NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

TESTING AND EVALUATION OF THE CONFIGURABLE FAULT TOLERANT PROCESSOR (CFTP) FOR SPACE-BASED APPLICATIONS

by

Charles A. Hulme

December 2003

Thesis Co-Advisors: Herschel H. Loomis, Jr.
Alan A. Ross

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Testing and Evaluation of the Configurable Fault Tolerant Processor (CFTP) For Space-Based Applications			**5. FUNDING NUMBERS**
6. AUTHOR(S) Hulme, Charles A.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

With the complexity of digital systems, reliability considerations are important. In many digital systems it is desirable to continuously monitor, exercise and test the system in order to determine whether the system is performing as desired. Such monitoring may enable automatic detection of failures via periodic testing or through the use of codes and checking circuits (e.g., built-in self-testing). While any complex system requires testing to ensure satisfactory performance, satellite systems require extensive testing for two additional reasons: they operate in an environment considerably different from that in which they were built, and after launch they are inaccessible to routine maintenance and repair. Because of these unique requirements, a specific solution is required such as a self-contained, autonomous, self-testing circuit. The focus of this thesis is on the design and development of a series of Built-In Self-Tests (BISTs) for use with the Configurable Fault Tolerant Processor (CFTP). The results of this thesis are two detailed designs for a Random Access Memory (RAM) BIST and a Read-Only Memory (ROM) BIST, as well as a conceptual design for a Field Programmable Gate Array (FPGA) BIST. These designs are stored on board the CFTP and are made to operate remotely and autonomously. Together, these BISTs provide a means to monitor, exercise, and test the CFTP components and thus facilitate a reliable design.

14. SUBJECT TERMS Field-Programmable Gate Array (FPGA), Built-In Self-Test (BIST), FPGA testing, Read-Only Memory (ROM) testing, Random Access Memory (RAM) testing, system diagnosis, system reliability			**15. NUMBER OF PAGES** 269
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

TESTING AND EVALUATION OF THE CONFIGURABLE FAULT TOLERANT PROCESSOR (CFTP) FOR SPACE-BASED APPLICATIONS

Charles A. Hulme
Captain, United States Marine Corps
B.S., Texas A&M University, 1995
M.S., Old Dominion University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2003

Author: Charles A. Hulme

Approved by: Herschel H. Loomis, Jr.
Thesis Co-Advisor

Alan A. Ross
Thesis Co-Advisor

John P. Powers
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

With the complexity of digital systems, reliability considerations are important. In many digital systems it is desirable to continuously monitor, exercise and test the system in order to determine whether the system is performing as desired. Such monitoring may enable automatic detection of failures via periodic testing or through the use of codes and checking circuits (e.g., built-in self-testing). While any complex system requires testing to ensure satisfactory performance, satellite systems require extensive testing for two additional reasons: they operate in an environment considerably different from that in which they were built, and after launch they are inaccessible to routine maintenance and repair. Because of these unique requirements, a specific solution is required such as a self-contained, autonomous, self-testing circuit. The focus of this thesis is on the design and development of a series of Built-In Self-Tests (BISTs) for use with the Configurable Fault Tolerant Processor (CFTP). The results of this thesis are two detailed designs for a Random Access Memory (RAM) BIST and a Read-Only Memory (ROM) BIST, as well as a conceptual design for a Field Programmable Gate Array (FPGA) BIST. These designs are stored on board the CFTP and are made to operate remotely and autonomously. Together, these BISTs provide a means to monitor, exercise, and test the CFTP components and thus facilitate a reliable design.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....1

II. CONFIGURABLE FAULT TOLERANT PROCESSOR DESIGN....5
- A. BACKGROUND....5
  - 1. Effects....6
  - 2. Solution....6
- B. CONCEPT....7
- C. COMPONENTS....11
- D. ARCHITECTURE....13
- E. CFTP STATUS....15
- F. CHAPTER SUMMARY....16

III. BUILT-IN SELF-TEST....17
- A. AN INTRODUCTION TO BIST....17
  - 1. What is BIST?....17
  - 2. Basic Architecture....18
  - 3. Advantages and Disadvantages....18
  - 4. The CFTP Self-Tests....19
- B. RANDOM ACCESS MEMORY TESTING....21
  - 1. Memory Problems....22
    - *a. Electrical Wiring Problems....23*
    - *b. Chip Connection Problems....25*
  - 2. Developing a Test Strategy....25
  - 3. Data-Bus Test....26
  - 4. Address-Bus Test....27
  - 5. Memory-Chip Test....28
  - 6. Designing the RAM Test....29
    - *a. Overview....29*
    - *b. Circuit Under Test (RAM)....32*
    - *c. Test Pattern Generator (Pattern)....32*
    - *d. Output Response Analyzer (Comparator)....34*
    - *e. State Machine....35*
    - *f. Address Counter (Counter)....40*
    - *g. Test Controller (Top-Level Control Logic)....41*
  - 7. Testing the Test....47
  - 8. Conclusions and RAM BIST Implementation....47
- C. READ-ONLY MEMORY TESTING....48
  - 1. State the Problem to be Solved....50
  - 2. Determine the Inputs and Outputs for the Test Device....52
  - 3. Define the States, Transitions and Outputs of Each State....53
  - 4. Determine the Computational Modules....54
  - 5. Develop a Data Subsystem Module....54

a. Registers *55*
b. Multiplexers *55*
6. Develop the System Module 57
7. Develop the Top Level Module 58
8. Testing the Test 59
9. Conclusions and ROM BIST Implementation 59
D. FIELD-PROGRAMMABLE GATE ARRAY TESTING 60
1. Introduction 61
2. Interfacing with the Test 61
3. The Test Process 62
4. The CLB Tests 64
5. The Interconnect Test 67
6. Conclusions and FPGA BIST Implementation 69

IV. CONCLUSIONS AND FOLLOW-ON RESEARCH 71
A. OVERVIEW 71
B. CONCLUSIONS 71
C. FOLLOW-ON RESEARCH 73

APPENDIX A: COMPLETE SCHEMATICS, VHDL CODES AND TEST-BENCH WAVEFORMS FOR SDRAM TEST 75
A. COMPLETE DESIGN 76
1. Schematic Diagram 76
2. Test-Bench Waveform 77
B. PATTERN MODULE 153
1. VHDL Code 153
2. Test-Bench Waveform 165
C. COMPARATOR MODULE 170
1. Schematic Diagram 170
2. Test-Bench Waveform 171
D. STATE MACHINE MODULE 172
1. VHDL Code 172
2. Test-Bench Waveform 196
E. ADDRESS COUNTER MODULE 203
1. Schematic Diagram 203
2. Test-Bench Waveform 204
F. COUNTER-CONTROL MODULE 212
1. VHDL Code 212
G. COUNTER-DECODE MODULE 215
1. VHDL Code 215
H. COMPARE-ENABLE MODULE 218
1. VHDL Code 218
I. PASS-COUNTER MODULE 220
1. Schematic Diagram 220
J. STATUS MODULE 221
1. VHDL Code 221
K. TOP LEVEL CONTROL LOGIC MODULE 222

1. Schematic Diagram 222
APPENDIX B: COMPLETE SCHEMATICS, VHDL CODES AND TEST-BENCH WAVEFORMS FOR EPROM/PROM TEST 223
A. MULTIPLEXER MODULE 224
1. VHDL Code 224
2. Test-Bench Waveform 225
B. ADDER MODULE 226
1. VHDL Code 226
2. Test-Bench Waveform 227
C. DATA MODULE 228
1. Schematic Diagram 228
2. Test-Bench Waveform 229
D. CONTROL MODULE 230
1. VHDL Code 230
2. Test-Bench Waveform 233
E. SYSTEM MODULE 234
1. Schematic Diagram 234
2. Test-Bench Waveform 235
F. COMPARATOR MODULE 236
1. Schematic Diagram 236
2. Test-Bench Waveform 237
G. TOP LEVEL MODULE 238
1. Schematic Diagram 238
2. Test-Bench Waveform 239
LIST OF REFERENCES 241
INITIAL DISTRIBUTION LIST 245

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. Solar Radiation. (From Ref. [1].) 5
Figure 2. Example of Satellite Capture for On-Orbit Servicing. (After Ref. [10].) 8
Figure 3. CFTP Conceptual Illustration. (After Ref. [4].) 10
Figure 4. Example Xilinx RADHARD Device Numbering. (From Ref. [18].) 11
Figure 5. Example Intel Flash Memory Device Numbering. (From Ref. [20].) 12
Figure 6. Example Xilinx ISP EPROM Device Numbering. (From Ref. [22].) 13
Figure 7. Example Xilinx OTP PROM Device Numbering. (From Ref. [23].) 13
Figure 8. CFTP Architecture. (From Ref. [4].) 14
Figure 9. ESPA Configuration. (After Ref. [25].) 15
Figure 10. Basic BIST Architecture. (After Ref. [26].) 18
Figure 11. BIST Conceptual Illustration. (After Ref. [4].) 20
Figure 12. Basic Memory Structure. 23
Figure 13. Possible Wiring Problems. 24
Figure 14. Proper Order of Memory-test Components. 26
Figure 15. Block Diagram of the RAM Test. 31
Figure 16. Pattern Module. 33
Figure 17. Pattern Test-Bench Waveform. 33
Figure 18. Comparator Module. 34
Figure 19. Comparator Test-Bench Waveform. 35
Figure 20. RAM Test State Machine 36
Figure 21. State Machine Module. 38
Figure 22. State Machine Test-Bench Waveform. 39
Figure 23. Counter Module. 41
Figure 24. Counter Test-Bench Waveform. 41
Figure 25. Counter-Control Module. 42
Figure 26. Counter-Decode Module. 43
Figure 27. Compare-Enable Module. 43
Figure 28. Pass-Counter Module. 44
Figure 29. Status Module. 44
Figure 30. Control Logic Module 45
Figure 31. Complete RAM Test Design. 46
Figure 32. General ROM Test Structure. (After Ref. [30].) 49
Figure 33. ROM Test Module. (After Ref. [29].) 52
Figure 34. ROM Test State Diagram 53
Figure 35. Control Module. 53
Figure 36. Control Module Test-Bench Waveform. 54
Figure 37. Adder Module. 54
Figure 38. Data Subsystem Details. 56
Figure 39. Data Module 56
Figure 40. Data Module Test-Bench Waveform. 57
Figure 41. System Module. 57

Figure 42. System Module Test-Bench Waveform. 58
Figure 43. Checksum Module. 58
Figure 44. Checksum Module Test-Bench Waveform. 59
Figure 45. FPGA Architecture Overview. (From Ref. [21].) 60
Figure 46. Configuration Logic Block. (After Ref. [21].) 63
Figure 47. Basic Architecture for CLB Test. (After Ref. [26].) 65
Figure 48. Basic CLB Test Architecture across FPGA array 66
Figure 49. CLB Test Configurations for FPGAs. (After Ref. [26].) 66
Figure 50. Basic Architecture for Interconnect Test Configuration. (After Ref. [26].) 68

LIST OF TABLES

Table 1. Radiation Effects and Mitigation. (After Ref. [2].) 6
Table 2. Radiation Effects and Mitigation Solutions Selected. 7
Table 3. Summary of Advantages and Disadvantages of BIST. (From Ref. [26].).....19
Table 4. Consecutive Data Values for the Sliding 1's Test. 27
Table 5. "Power-of-Two" Addresses. 28
Table 6. Data Values for an Increment Test. 29
Table 7. RAM Test Patterns. 32
Table 8. Description of the states used in the Memory Test. 37
Table 9. Implementation Approaches for Control Subsystems. (After Ref. [30].).....49
Table 10. Example ROM contents. 50
Table 11. External Data and Control Signals. 51
Table 12. Fault Table for the XOR Gate. (After Ref. [3].) 64
Table 13. Propagation D Cube for the XOR Gate. (After Ref. [3].).....65

THIS PAGE INTENTIONALLY LEFT BLANK

GLOSSARY

ASIC	Application Specific Integrated Circuit
BIST	Built-In Self-Test
BUT	Block Under Test
C&DH	Command and Data Handler
CFTP	Configurable Fault-Tolerant Processor
CLB	Configurable Logic Block
COTS	Commercial-Off-The-Shelf
CPU	Central Processing Unit
CUT	Circuit Under Test
EDAC	Error Detection And Correction
ELV	Expendable Launch Vehicle
EPROM	Erasable Programmable Read-Only Memory
ESPA	Expendable Launch Vehicle (ELV) Secondary Payload Adapter
FPGA	Field-Programmable Gate Array
I/O	Input/Output
IOB	Input/Output Block
ISP	In System Programmable
LC	Logic Cell
LUT	Look Up Table
MeV	Mega-electron Volt
MidSTAR-1	Midshipmen Science and Technology Application Research Mission 1
Mux	Multiplexer
NPS	Naval Postgraduate School
NPSAT1	Naval Postgraduate School Satellite Mission 1
ORA	Output Response Analyzer
OTP	One-Time Programmable
PCB	Printed Circuit Board
PROM	Programmable Read-Only Memory
RADHARD	Radiation Hardened
rads	Radiation Absorbed Dose

RAM	Random Access Memory
ROM	Read-Only Memory
SDRAM	Synchronous Dynamic Random Access Memory
SEE	Single Event Effect
SEL	Single Event Latchup
SERB	Space Experiments Review Board
SEU	Single Event Upset
SOC	System-On-A-Chip
TID	Total Ionizing Dose
TMR	Triple Modular Redundant
TPG	Test Pattern Generator
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WUT	Wire Under Test

ACKNOWLEDGMENTS

I would like to thank all of the professors, engineers, technicians, and students of the Naval Postgraduate School who made not only this research possible but also made my studies here at the Naval Postgraduate School a true educational experience. Many of them may not realize the magnitude of their efforts, but the author does.

Special thanks are owed to these individuals for their support:

To Professors Loomis and Ross, your knowledge in Computer Engineering and Space Systems has been inspiring.

To First Lieutenant Rong Yuan for your friendship, your willingness to answer my never-ending questions, and for continuously reminding me that just because I speak English doesn't mean I know the English language.

To Professor Butler, for your impressive teaching methods and skills.

To Jim Lucier, thank you for the time we spent climbing the granite walls of Yosemite, the crumbling rock of the Pinnacles, the death defying Tyrolean traverse, and our epic adventure up the Hobbit Book.

And most importantly, I wish to thank my family: Mari, Addy Mae, and Baxter. Your patience, understanding, and support throughout this process have been a blessing. Finally, I owe an extra thanks to my loving wife, for without her none of this would have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

With the complexity of digital systems, reliability considerations are important. Being physical devices, digital circuits are subject to failures caused by faults. A fault is defined as any change in a system that causes it to behave differently from the original system. In digital systems, typical maintenance deals with detection, location, and repair of any such faults.

In many digital systems it is desirable to continuously monitor, exercise and test the system in order to determine whether the system is performing as desired. Such monitoring may enable automatic detection of failures via periodic testing or through the use of codes and checking circuits (e.g., built-in self-testing). In such systems, special hardware and software must be incorporated in the system to obtain these reliability objectives. This thesis will address problems associated with testing a digital system to detect and locate faults. Additionally, the thesis addresses the design of self-testing digital systems in which faults can be automatically detected for key components of the digital system.

While any complex system requires testing to ensure satisfactory performance, satellite systems require extensive testing for two additional reasons: they operate in an environment considerably different from that in which they were built and after launch they are inaccessible to routine maintenance and repair. Because of these unique requirements, a specific solution is required such as a self-contained, autonomous, self-testing circuit. This Built-In Self-Test (BIST) provides a means to monitor and maintain specific components remotely.

The focus of this thesis was on the design and development of a series of BISTs for use with the Configurable Fault Tolerant Processor (CFTP) in space applications. The results of this thesis are detailed designs for a Random Access Memory (RAM) BIST and a Read-Only Memory (ROM) BIST, as well as a conceptual design for a Field Programmable Gate Array (FPGA) BIST. These designs are stored on board the CFTP and are made to operate remotely and autonomously. Together, these BISTs provide a means to monitor, exercise, and test the CFTP components and thus facilitate a reliable design.

The CFTP is an experiment being conducted at the Naval Postgraduate School (NPS) that attempts to demonstrate flexibility achieved through reprogrammable and reconfigurable technology. The CFTP design incorporates Field Programmable Gate Arrays (FPGAs) as a foundation for this flexibility. The CFTP is designed to combat the three principal errors experienced in space environments: Total Ionizing Dose (TID), Single Event Upset (SEU), and Single Event Latchup (SEL). Through careful component selection, TID and SEL can be prevented, while a fault tolerant scheme is needed to mitigate SEUs. The CFTP's fault-tolerant architecture is accomplished with a Triple-Modular-Redundant design. In this design, three processors operate in lock step with a majority voter evaluating each output, and correcting any errors induced from SEUs.

The BIST design is made up of multiple sub-designs, each designed to test a specific component of the CFTP. These additional designs are stored on board the CFTP and are made to operate remotely and autonomously. In typical approaches of this fashion, this additional circuitry translates to an overhead in the overall system design and functionality. Fortunately, due to the reprogrammability of FPGA devices, the additional circuitry required to incorporate a BIST can be maintained in the system's configuration-storage as one of many possible system configurations. Therefore, the area overhead and performance penalties typically associated with traditional BIST approaches can be avoided.

One of the components to be tested is the System-Memory, RAM, on board the CFTP. Because it is necessary for the CFTP to store and retrieve information accurately from its memory, proper physical and electrical operation of the memory components and their interconnect is required. The RAM BIST uses a method of writing, reading and verifying specific patterns to and from locations in the System-Memory.

Other components to be tested are the configuration-storage components, EPROM/PROM and Flash Memory, on board the CFTP. When using FPGAs as a means to replicate architecture and functionality, back-ups and variations of these configurations need to be maintained in some form of a configuration-storage device. These devices can be one of many forms of Read-Only Memory (ROM); the three selected for the CFTP are EPROM, PROM, and Flash Memory. The ROM BISTs are identical and use a method of

calculating a checksum signature particular to the data stored in the respective devices and comparing it to the correct signature.

The final components discussed are the FPGAs themselves. Because they are the core of the CFTP design, proper operation of their internal and external characteristics is critical. In this portion of the thesis, research is presented which explains the process of dividing up the FGPA into groups and allowing one group to test another in a round-robin fashion until all the groups have been tested. Furthermore, this chapter discusses the building blocks of the FPGA and testing the Configurable Logic Blocks (CLBs), the elements used in constructing representative logic, and testing the interconnect surrounding these logic blocks.

Further research is required to incorporate these designs into the board-level designs for the CFTP development board, qualification board, and flight models. Each system may require slight modifications for specific protocol requirements (e.g., handshakes, timing, etc.). These BISTs are an excellent means to provide a functional baseline prior to environmental testing for space flight qualification.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

With the complexity of digital systems, reliability considerations are important. Being physical devices, digital circuits are subject to failures caused by faults. A fault is defined as any change in a system that causes it to behave differently from the original system. In digital systems, typical maintenance deals with detection, location, and repair of any such faults.

In many digital systems it is desirable to continuously monitor, exercise and test the system in order to determine whether the system is performing as desired. Such monitoring may enable automatic detection of failures via periodic testing or through the use of codes and checking circuits (e.g., built-in self-testing). In such systems, special hardware and software must be incorporated in the system to obtain these reliability objectives. This thesis will address problems associated with testing a digital system to detect and locate faults. Additionally, the thesis addresses the design of self-testing digital systems in which faults can be automatically detected for key components of the digital system.

While any complex system requires testing to ensure satisfactory performance, satellite systems require extensive testing for two additional reasons: they operate in an environment considerably different from that in which they were built and after launch they are inaccessible to routine maintenance and repair. Because of these unique requirements, a specific solution is required such as a self-contained, autonomous self-testing circuit. This Built-In Self-Test (BIST) provides a means to monitor and maintain specific components remotely.

The focus of this thesis was on the design and development of a series of BISTs for use with the Configurable Fault Tolerant Processor (CFTP) in space applications. The results of this thesis are detailed designs for a Random Access Memory (RAM) BIST and a Read-Only Memory (ROM) BIST, as well as a conceptual design for a Field Programmable Gate Array (FPGA) BIST. These designs are stored on board the CFTP and

are made to operate remotely and autonomously. Together, these BISTs provide a means to monitor, exercise, and test the CFTP components and thus facilitate a reliable design.

The CFTP is an experiment being conducted at the Naval Postgraduate School (NPS) that attempts to demonstrate flexibility achieved through reprogrammable and reconfigurable technology. The CFTP design incorporates Field Programmable Gate Arrays (FPGAs) in conjunction with Erasable Programmable Read-Only Memory (EPROM), Programmable Read-Only Memory (PROM), and Flash Memory as a foundation for this flexibility.

The CFTP is designed to combat the three principal errors experienced in space environments: Total Ionizing Dose (TID), Single Event Upset (SEU), and Single Event Latchup (SEL). TID effects can be addressed by hardening or shielding the device against radiation, while SELs can be dealt with by through fabrication processes. The CFTP has taken a unique approach to mitigating SEUs by incorporating a Triple Modular Redundancy (TMR) design. In this design, three processors operate in lock step with a majority voter evaluating each output, and correcting any errors induced from SEUs. Chapter 2 continues this discussion of the CFTP's background, the underlying principles behind its inception, as well as a description of the specific components selected and the architecture which ties them together.

Chapter 3 begins the discussion of the BISTs and how they are incorporated into the CFTP design, while the sections within describe in detail the individual BIST designs. The BIST design is made up of multiple sub-designs, and each is designed to test a specific component of the CFTP. These additional designs are stored on board the CFTP and are made to operate remotely and autonomously. In typical approaches of this fashion, this additional circuitry translates to an overhead in the overall system design and functionality. Fortunately, due to the reprogrammability of FPGA devices, the additional circuitry required to incorporate a BIST can be maintained in the system's configuration-storage as one of many possible system configurations. Therefore, the area overhead and performance penalties typically associated with traditional BIST approaches can be avoided.

The first BIST design described in Chapter 3 is the RAM BIST. One of the CFTP components to be tested is the System-Memory, RAM, on board the CFTP. Because it is necessary for the CFTP to store and retrieve information accurately from its memory, proper physical and electrical operation of the memory components and their interconnect is required. The RAM BIST uses a method of writing, reading and verifying specific patterns to and from locations in the System-Memory.

The next section of Chapter 3 describes the ROM BIST. After the System-Memory, other components to be tested are the configuration-storage components, EPROM/PROM and Flash Memory, on board the CFTP. When using FPGAs as a means to replicate architecture and functionality, back-ups and variations of these configurations need to be maintained in some form of a configuration-storage device. These devices can be one of many forms of Read-Only Memory (ROM); the three selected for the CFTP are EPROM, PROM, and Flash Memory. The ROM BISTs are identical and use a method of calculating a checksum signature particular to the data stored in the respective devices and comparing it to the correct signature.

The final BIST discussed in Chapter 3 is the FPGA BIST. Because the FPGAs are the core of the CFTP design, proper operation of their internal and external characteristics is critical. In this portion of the thesis, research is presented which explains the process of dividing up the FGPA into groups and allowing one group to test another in a round-robin fashion until all the groups have been tested. Furthermore, this chapter discusses the building blocks of the FPGA and testing the Configurable Logic Blocks (CLBs), the elements used in constructing representative logic, and testing the interconnect surrounding these logic blocks.

Finally, Chapter 4 brings to light some conclusions as well as needed research. Further research is required to incorporate these designs into the board-level designs for the CFTP development board, qualification board, and flight models. Each system may require slight modifications for specific protocol requirements (e.g., handshakes, timing, etc.). These BISTs are an excellent means to provide a functional baseline prior to environmental testing for space flight qualification.

THIS PAGE INTENTIONALLY LEFT BLANK

II. CONFIGURABLE FAULT TOLERANT PROCESSOR DESIGN

In order to give context to this thesis, a brief discussion of the Configurable Fault Tolerant Processor (CFTP) design is required. This discussion includes key issues with space environments and their source for errors in logical circuits, as well as the method in which CFTP deals with these concerns. Additionally, this chapter describes the design framework, solution, and the state of the CFTP to date.

A. BACKGROUND

Close to the earth's surface, within the atmosphere, electrical circuits are shielded from many of the effects of space, like radiation. Leaving the earth's atmosphere, a circuit is exposed to an environment heavily populated by high-energy ions [1].

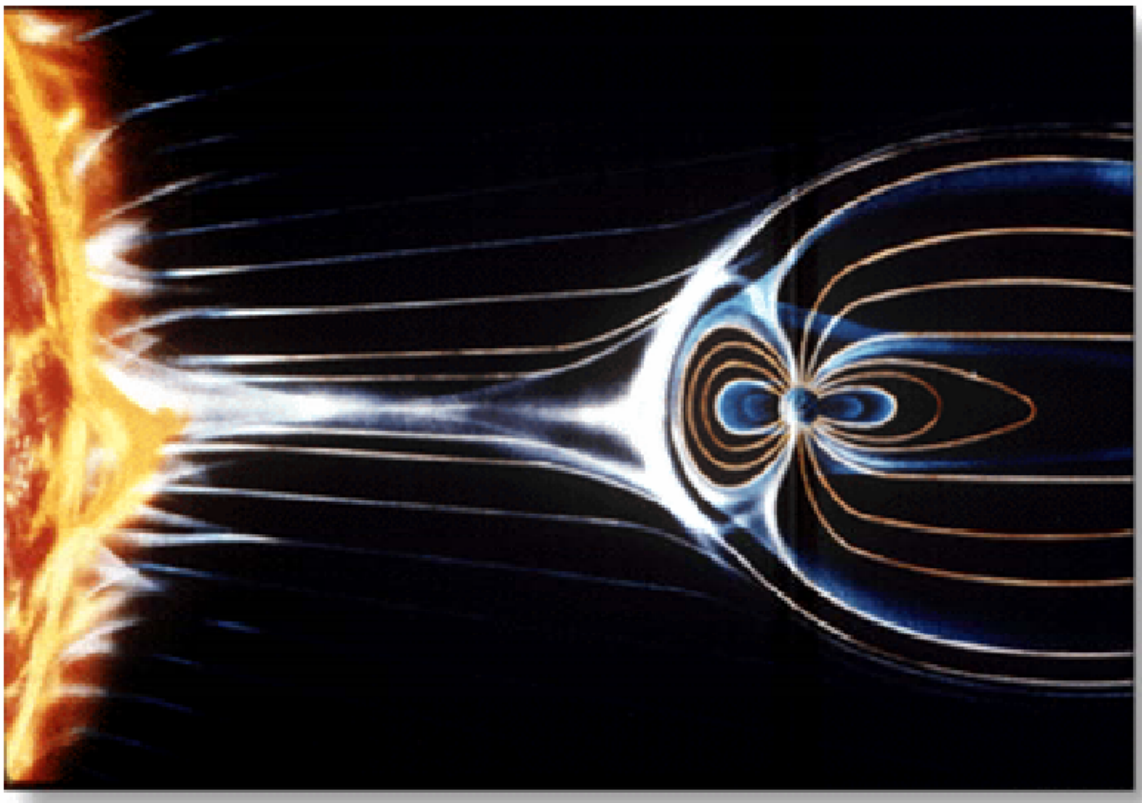


Figure 1. Solar Radiation. (From Ref. [1].)

1. Effects

These high-energy ions cause several types of errors to occur in electrical circuits. Two principal types of errors or failures the CFTP addresses are Total Ionizing Dose (TID) and Single Event Effects (SEE) [2]. Total Ionizing Dose effects contribute to the deterioration of a device over time. A Single Event Effect is any measurable effect to a circuit due to a single ion strike. Single Event Effects include Single Event Latchup (SEL) and Single Event Upset (SEU) [2]. Single Event Latchup is a condition that may result in the potentially permanent loss of device functionality due to a single-event-induced high-current state. SEU is a condition that may result in the unwanted change of value in a memory cell due to a charge absorbed into the device body.

2. Solution

Table 1 summarizes the principal effects of radiation on electronic circuits along with several methods that can be employed to combat those effects. First, TID effects are typically addressed with Radiation-Hardened (RADHARD), Radiation-Tolerant, or shielded devices [2]. Second, SELs can be dealt with by either fabricating devices on an epitaxial substrate or incorporating guard-ring or dielectric-isolation processes [2, 3]. Third, SEUs can be addressed through many different fault-tolerant schemes such as Triple Modular Redundancy (TMR), Quadded Logic, or Software Fault Tolerance [3].

Radiation Effect	Mitigation Techniques
Total Ionizing Dose	➢Radiation Hardening ➢Shielding
Single Event Latchup (SEL)	➢Epitaxial Substrate ➢Guard-Ring ➢Dielectric-Isolation
Single Event Upset (SEU)	➢TMR ➢Quadded Logic ➢Software Fault Tolerance

Table 1. Radiation Effects and Mitigation. (After Ref. [2].)

Table 2 provides the solution that the CFTP uses to mitigate all these effects: Radiation Hardening for key devices, fabrication of these on epitaxial substrates, and incorporating Triple Modular Redundancy [4].

Radiation Effect	Mitigation Techniques
Total Ionizing Dose	➢Radiation Hardening
Single Event Latchup (SEL)	➢Epitaxial Substrate
Single Event Upset (SEU)	➢TMR

Table 2. Radiation Effects and Mitigation Solutions Selected.

While these techniques provide a solution, there are trade-offs in the form of performance, cost, and availability [4]. The following section will discuss these trade-offs as well as the basis of the CFTP technology, re-programmability.

B. CONCEPT

The primary objectives in developing the Configurable Fault Tolerant Processor (CFTP) were three-fold [5, 6]. First, the CFTP was to demonstrate the applicability of a reconfigurable, fault-tolerant System-On-a-Chip (SOC)[1] for space-based applications. Second, the CFTP was to demonstrate the use of reprogrammable technology in spacecraft architectures. Third, the CFTP must provide a reliable design to accomplish these reprogramming and reconfiguration objectives. The CFTP design was centered on a reconfigurable system instead of an Application-Specific Integrated Circuit (ASIC). Custom built, ASICs are expensive and inflexible. Furthermore, TID and SEL mitigation must be applied after the ASIC is designed. By utilizing reprogrammable devices that offer design flexibility, CFTP can provide faster design cycles and lower costs [7, 8]. These benefits in time, money, and effort are tremendous, especially by providing a system with on-orbit upgradeability and reconfigurability. Using this technology in spacecraft architectures decreases development time, decreases costs, and increases reliability as well as flexibility in hardware development and implementation [7, 8].

Radiation-Hardened (RADHARD) parts, due to the exacting fabrication requirements and the processes involved to harden the devices are by their very nature slower, larger, and more expensive than their commercial equivalents [4]. Because of this, RADHARD parts lag current technology and are possibly years old by the time of launch. While this holds true to some extent for RADHARD Field-Programmable Gate

[1] An integrated embedded computer system on a single chip. In the context of SRAM-based FPGAs, this usually discounts an external FPGA configuration ROM and external RAM.

Arrays (FPGAs) too, there is an exception. Because FPGAs are designed to be reprogrammable, they are generic in their design and not application specific. So while the process to fabricate a RADHARD FPGA may be involved and costly, they are still somewhat readily available and technologically current. Therefore, a system designed to incorporate RADHARD FPGAs can harness the radiation mitigation benefits of a RADHARD device as well as the availability and technology of non-RADHARD devices. Additionally, the inherent reprogrammability and reconfigurability FPGAs bring to the CFTP provide a medium for updateable architectures in space applications.

Today, once a satellite is in orbit making hardware changes is almost always impossible. Because of this, many research facilities have been searching for methods to provide on-orbit satellite servicing. Solutions range from physically visiting the satellite with an autonomous servicing satellite, as with the Orbital Express program and seen in Figure 2, to utilizing proven, reliable uplink communications, as with the CFTP [9, 10]. The CFTP program is breaking ground in on-orbit servicing using reconfigurable logic and uploading programmable modifications via command and control communications. If successful, tremendous flexibility will be achieved. The CFTP could be reconfigured on-orbit to correct errors, meet dynamic mission requirements, upgrade, or serve as back-up devices to several on-board systems.

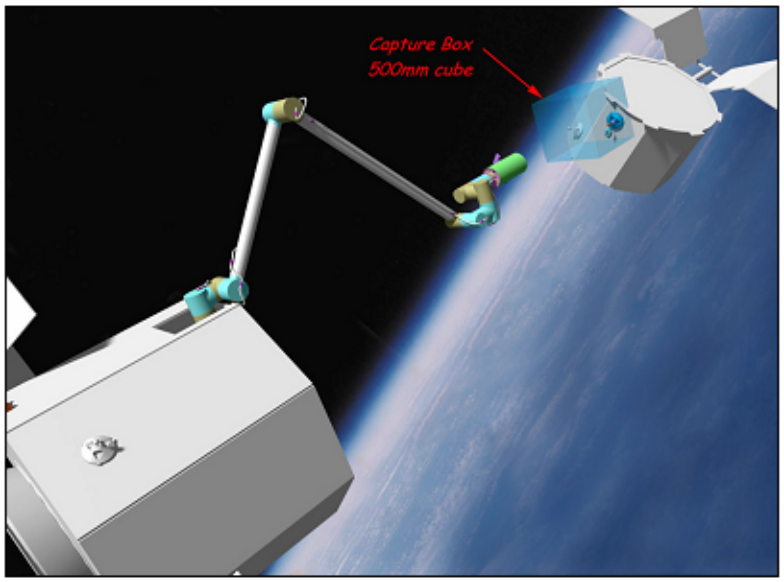


Figure 2. Example of Satellite Capture for On-Orbit Servicing. (After Ref. [10].)

The CFTP program has evolved over the years and across numerous graduate researchers. It has evolved from its beginnings as an investigation into fault-tolerant computing techniques into an exploration of the applicability, design and testing of using reprogrammable and reconfigurable technology in space-based applications and is detailed in References [2, 4, 11-16]. These individual theses have sought to solve unique research questions, and have cumulatively established a significant number of design constraints. The design was framed around using Field-Programmable Gate Arrays (FPGA) as a basis for a SOC design, implementing a TMR-voting scheme for fault-tolerance, using a 16-bit or 32-bit processor softcore[2], maximizing the use of Commercial Of The Shelf (COTS) technology, and introducing real-time on-orbit reconfigurability. The physical design constraints are a Printed Circuit Board (PCB) 5.3 x 7.3 inches, with a slightly modified PC104 Bus interface, using FPGA and COTS devices, all with a targeted maximum power consumption of 11 Watts or less [17].

Figure 3 is a conceptual illustration of the CFTP design on a PCB and, while it is not a true SOC, it is close with a total chip count of 13. Of these 13 chips, eight are memory chips, two are FPGAs, two are power converters, and one is an oscillator [4]. The Processor FPGA is the large block in the top left (FPGA 2 - containing the TMR scheme outlined within it) and the Controller FPGA is in the lower left (FPGA 1 - containing the configuration control, command and status registers, bus interface logic, etc.). On the right side of the image are the System-Memory, configuration memory, and left-over discrete components such as capacitors, resistors, etc. On-orbit, the CFTP will operate with either a triple-redundant softcore or multiple other programmable modules to test the configurability and reconfigurability of the system.

[2] Softcore describes the concept of implementing electronic hardware in computer code.

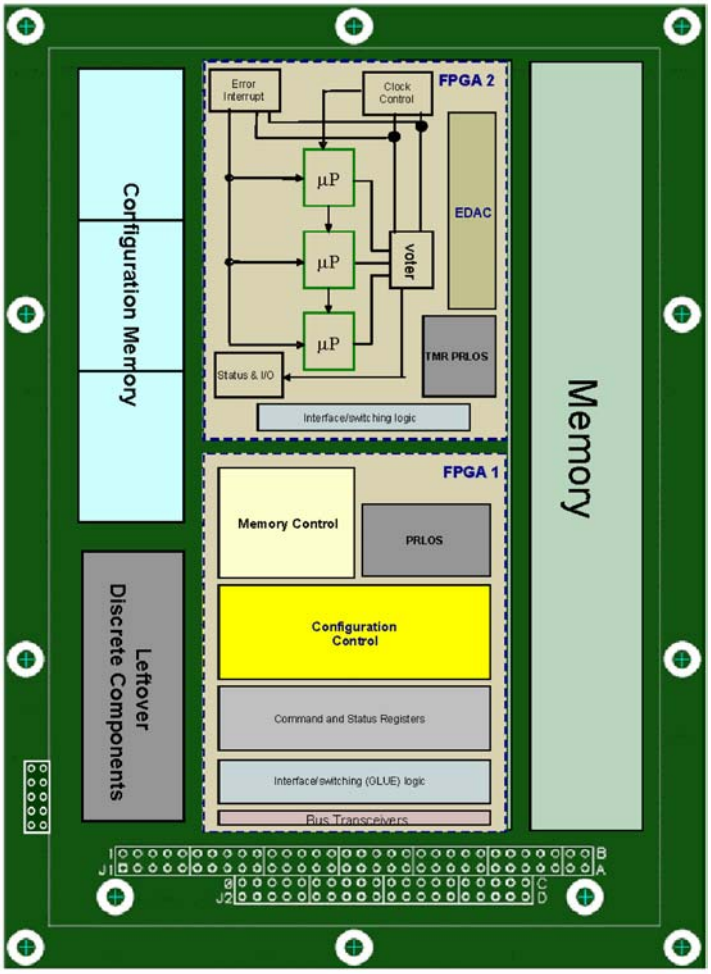


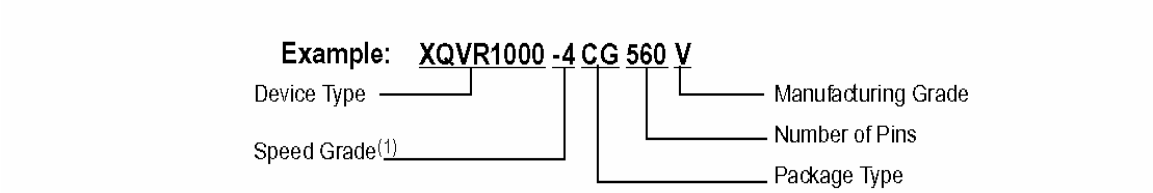
Figure 3. CFTP Conceptual Illustration. (After Ref. [4].)

The CFTP's radiation tolerance and latchup mitigation are accomplished by selecting RADHARD FPGAs which are fabricated on an epitaxial substrate for both the Processor and Controller FPGAs, RADHARD Read-Only Memory (ROM) for the Controller FPGA's configuration-storage device, Flash Memory with a proven radiation performance for the Processor FPGA's configuration storage-device, Synchronous Dynamic RAM (SDRAM) from a tested and qualified batch for the System-Memory, and performance-proven devices commonly used in the space industry for the power converters, oscillator, and discrete components [4]. While the FPGAs are hardened to the effects of TID and SEL, these mitigation efforts do not protect the system from SEUs. As previously mentioned, SEUs must be addressed through some form of a fault-tolerant scheme.

The CFTP's fault-tolerant architecture is accomplished with a Triple-Modular-Redundant design. In this design, three processors operate in lock step with each output voted on. If a conflict is found among the three processors (in this case due to an SEU-driven error), an interrupt routine saves and reloads the faulty processor with the correct data from the other two, as opposed to traditional methods of resetting the processors to a "trusted" state (re-initialize/re-boot/re-format) which results in a potentially significant amount of data loss [15]. Error-Detection-And-Correction (EDAC) circuitry is used for single-bit-error correction and double-bit-error detection of data errors in the System-Memory [4]. By incorporating both TMR and EDAC into the CFTP architecture, the system can correct for any errors due to SEUs.

C. COMPONENTS

The main components of the CFTP design are the Processor FPGA, Controller FPGA, Configuration Storage, and System-Memory [4]. The devices selected for the Processor and Controller FPGAs are the Xilinx XQVR600-4CB228M FPGA (Figure 4 shows the part number information) [4, 18]. This device provides a gate count of 661,111, is guaranteed RADHARD for 100 krad of TID, SEL immune, and comes in a 228-pin ceramic quad flat package [18, 19]. The total number of bits required to configure each FPGA is 3,607,968 bits [21].



Device Ordering Options

Device Type
XQVR300
XQVR600
XQVR1000

Package	
CB228	228-pin Ceramic Quad Flat Package
CG560	560-column Ceramic Column Grid Package

Grade		
M	Military Ceramic	T_C = –55°C to +125°C
V	QPro Plus	T_C = –55°C to +125°C
Q	MIL-PRF-38535(2)	T_C = –55°C to +125°C

Notes:
1. -4 only supported speed grade.
2. Class Q must be ordered with SMD number.

Figure 4. Example Xilinx RADHARD Device Numbering. (From Ref. [18].)

For their radiation performance, the Intel TE28F320C3BA 32-Mbit Flash Memory was chosen to store all of the Processor FPGAs configurations (Figure 5 shows the part number information) [4, 20]. At 32-Mbits and 3,607,968 bits per configuration, the Intel Flash Memory is capable of holding up to eight configurations [21].

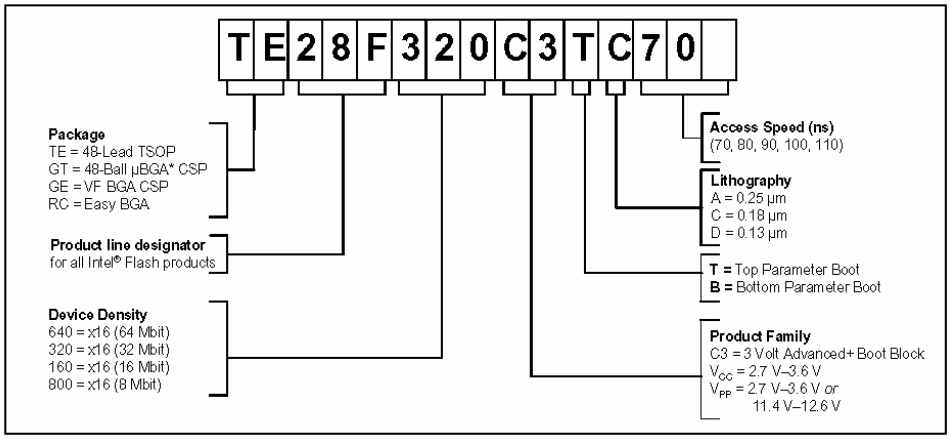


Figure 5. Example Intel Flash Memory Device Numbering. (From Ref. [20].)

Two arrangements were chosen for the Controller FPGA's configuration-storage device, an Erasable Programmable Read-Only Memory (EPROM) and a Programmable Read-Only Memory (PROM). The former device, a Xilinx XCV18V04 ISP EPROM that is In-System Programmable, is used during development to allow for changes to be made to the Controller FPGA's configuration (Figure 6 shows the part number information) [4, 22]. At 4-Mbits and 3,607,968 bits per configuration, the ISP PROM is capable of holding one configuration [21]. The latter, a Xilinx XQR17V16 OTP PROM which is RADHARD and One-Time Programmable, will be used in the final Flight model (Figure 7 shows the part number information) [4, 23]. At 16-Mbits and 3,607,968 bits per configuration, the OTP PROM is capable of holding up to four configurations [21].

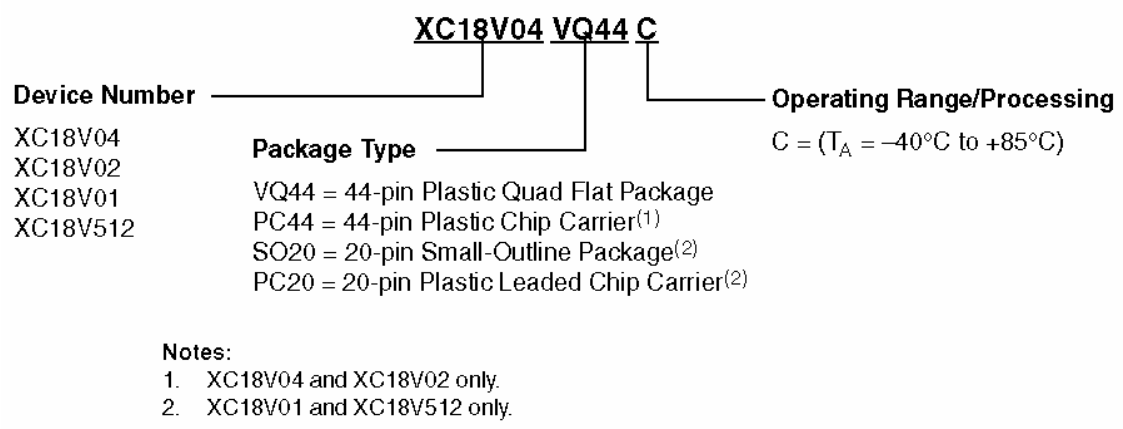


Figure 6. Example Xilinx ISP EPROM Device Numbering. (From Ref. [22].)

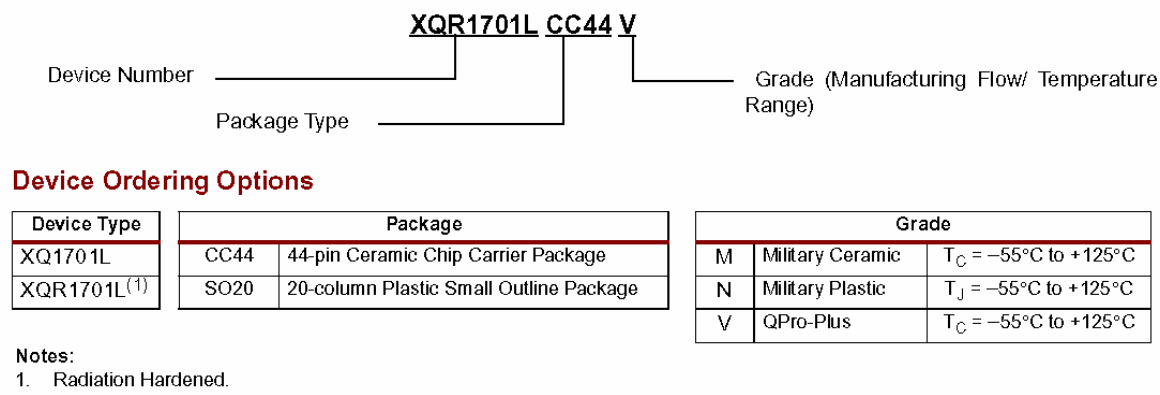


Figure 7. Example Xilinx OTP PROM Device Numbering. (From Ref. [23].)

A batch of Hitachi (now Elpida) HM5225405B-75/A6/B6 Synchronous Dynamic RAM (SDRAM), with a proven record of greater than 40 krad TID and an SEL threshold of 46.5 MeV-cm^2/mg, was provided by SEAKR Engineering [4]. These memory chips are used for the CFTP System-Memory and provide 256-Mbits of SDRAM organized as 16,777,216 words by 4-bits by 4 banks in a standard 54-pin plastic TSOP II [24].

D. ARCHITECTURE

Putting it all together, Figure 8 shows the basic architecture of the CFTP [4]. In its default configuration, the data paths flow as follows. First, power on/reset initiates configuring the Controller FPGA from the EPROM/PROM. Second, the Controller sends status through the PC104 onto the bus and receives commands and data. The Controller initiates configuring the Processor FPGA from the Flash Memory through the

Controller. From the Processor FPGA, data can be stored and retrieved from the System-Memory. Additional data paths have been incorporated to allow for future design flexibility. These allow for additional flow of data between FPGAs, into the EPROM and from the PC104 Bus [4].

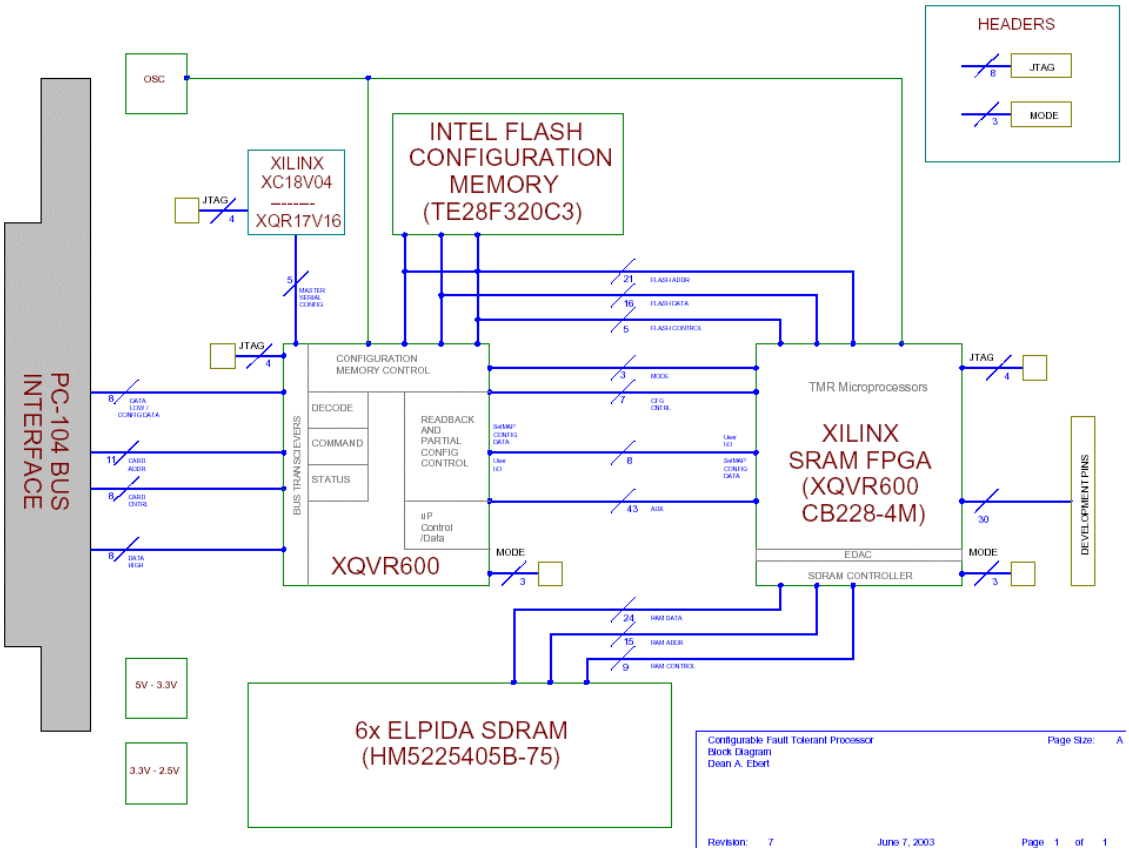


Figure 8. CFTP Architecture. (From Ref. [4].)

E. CFTP STATUS

In 2002, the CFTP project was presented to both the Navy and DoD Space Experiments Review Board (SERB) and was selected for space flight and manifested on two satellites: Midshipman Space Technologies Applications Research (MidSTAR-1) and Naval Postgraduate School Satellite (NPSAT1). Both satellites are currently scheduled to launch in September of 2006 into a Low Earth Orbit. Figure 9 shows the launch configuration, with both satellites mounted via an Expendable Launch Vehicle (ELV) Secondary Payload Adapter (ESPA) [25]. As a result, both satellites will release into relatively similar orbits and are expected to provide very comparable data. In 2003, the CFTP Project has already attended the Navy SERB and the DoD SERB seeking a flight that will subject CFTP to a higher degree of radiation exposure [5, 6].

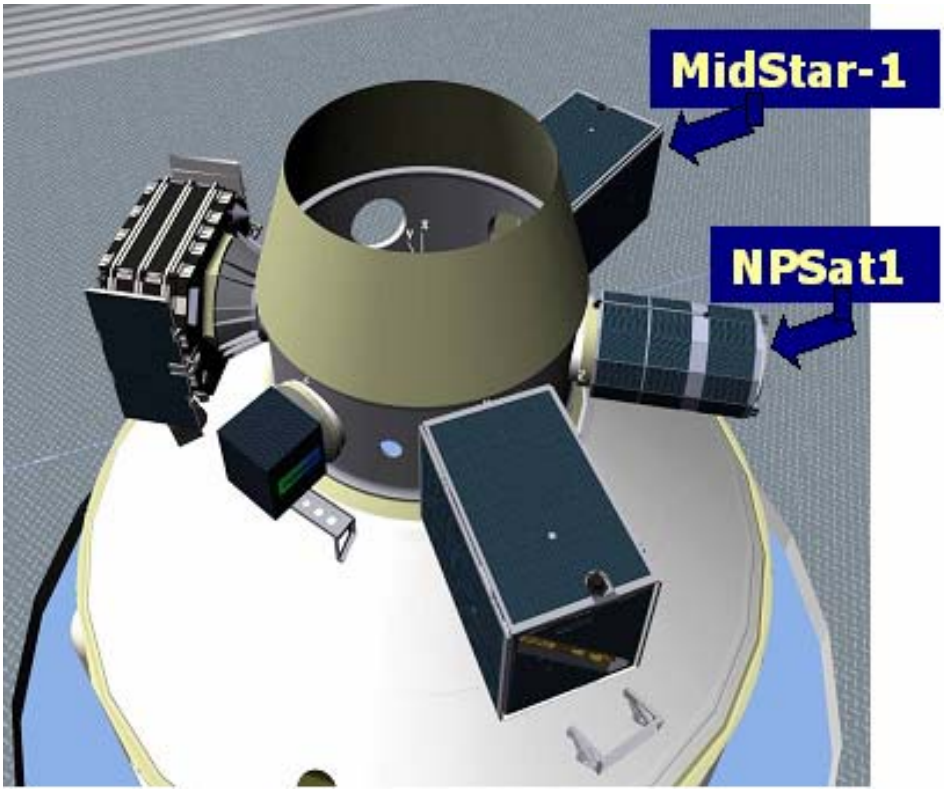


Figure 9. ESPA Configuration. (After Ref. [25].)

F. CHAPTER SUMMARY

This Chapter provided background information significant to the design process of the CFTP. Fundamental to the design goals for the CFTP are the concepts of immunity from the effects of a space environment, designing a system with a reconfigurable and reprogrammable architecture, and maximizing reliability.

With an understanding of the design and the components involved, the next chapter focuses on the system's functionality. Chapter III explores the design of a hardware self-test which the CFTP can utilize to test components and their interconnections in order to determine whether a device or the system as a whole is faulty or fault-free.

III. BUILT-IN SELF-TEST

This chapter presents an introduction to the Built-In Self-Test (BIST), including what it is and how it works, as well as the actual tests themselves. The intent is to establish the basic principles used in designing the BIST and then present the resulting self-test.

A. AN INTRODUCTION TO BIST

This Section begins with an explanation of the BIST concept, its basic architecture, as well as the primary advantages and disadvantages of incorporating this self-test design into a system.

1. What is BIST?

BIST, in its simplest form, is a circuit that tests itself to determine if it is fault-free or faulty [3]. Usually this entails incorporating additional circuitry and functionality into the design of the circuit in order to accomplish the self-testing aspect. This additional circuitry and functionality must generate test patterns and provide a means to analyze the output response [26]. The output responses of the Circuit Under Test (CUT) must correspond to the test patterns in order to pass as a fault-free circuit.

Fortunately, due to the reprogrammability of FPGA devices, the additional circuitry required to incorporate a BIST can be maintained in the system's configuration-storage as one of many possible system configurations (also known as off-line testing) [26]. Therefore, the area overhead and performance penalties typically associated with traditional BIST approaches can be avoided. As discussed in Chapter II, the CFTP design includes Flash Memory and EPROM/PROM that are programmed with a variety of configurations. Therefore when it is desired to operate the BIST, an FPGA can be reconfigured with the BIST configuration. Once the testing is complete, the CFTP can be reprogrammed to operate with one of multiple system functionalities. In doing so, the BIST is achieved with no area overhead or performance penalty to the system's functionality.

2. Basic Architecture

The block diagram in Figure 10 represents the basic architecture of the BIST circuitry. The BIST architecture includes the Test Pattern Generator (TPG), the Output Response Analyzer (ORA), and the Test Controller. The TPG produces a sequence of patterns, the ORA tests the output responses and produces a Pass/Fail signal, and the Test Controller initializes the BIST and maintains the operation of the self-test. Additionally, the BIST may require signals for starting the sequence (BIST Start), reporting the results (Pass/Fail), or reporting the completion of the sequence (BIST Done).

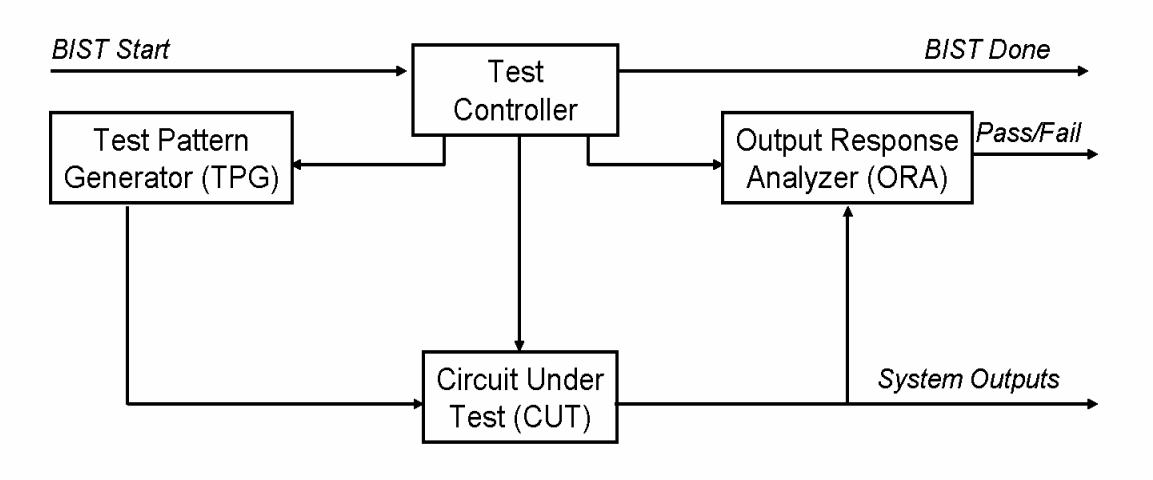


Figure 10. Basic BIST Architecture. (After Ref. [26].)

3. Advantages and Disadvantages

Table 3 lists some advantages and disadvantages typical to most BIST applications. Case studies have confirmed, however, that the advantages of BIST routinely make up for the disadvantages [26]. The studies have shown that these results are particularly true when the BIST is an off-line test routine (negating the increase in chip area and performance penalties). The remaining savings must address the additional design effort and project risk. The studies show that once the products are in the field and operational, the device may undergo repeated testing over time; the development testing accounts for only a small portion of the total tests run during the system's life-cycle [26].

Advantages	Disadvantages
Vertical Testability (wafer to system)	Area overhead
High diagnostic resolution	Performance penalties
At-speed testing	Additional design time and effort
Reduced need for external test equipment	Additional risk to project
Reduced test development time and effort	
More economic burn-in testing	
Reduced manufacturing test time and cost	
Reduced time-to-market	

Table 3. Summary of Advantages and Disadvantages of BIST. (From Ref. [26].)

Additionally, BISTs provide an advantage specific to FPGAs and other programmable devices. That is, if the locations of faults in the FPGA can be determined, then the FPGA can be reconfigured to avoid the faults when it is later reprogrammed. However, BISTs also have a disadvantage specific to these devices. Systems with these components rely on stored programs to define them. Incorporating BISTs require dedicating one or more of these stored programs to testing, leaving less for the overall system functions.

4. The CFTP Self-Tests

As discussed in Chapter II (and shown in Figure 11), the CFTP consists of 13 chips: eight memory chips, two FPGAs, two power converters, and one oscillator. The focus of the BIST developed here is on the eight memory chips and two FPGAs. Of the eight memory chips, six are System-Memory (i.e., RAM) and the remaining two are configuration-storage devices for the Controller and Processor FPGAs (i.e., PROM/EPROM and Flash Memory). The BISTs contain operations specific to each type of component and are divided into three independent tests: RAM BIST (System-Memory), ROM BIST (configuration-storage memory), and FPGA BIST (both FPGAs). The test sequence for

these BISTs follows the order: FPGA BIST, ROM BIST, and then RAM BIST. This will allow for assurance in the FPGAs prior to loading them with the ROM BIST. Finally, once the PROM/EPROM and Flash Memory functionalities are confirmed, the RAM BIST configuration can be loaded.

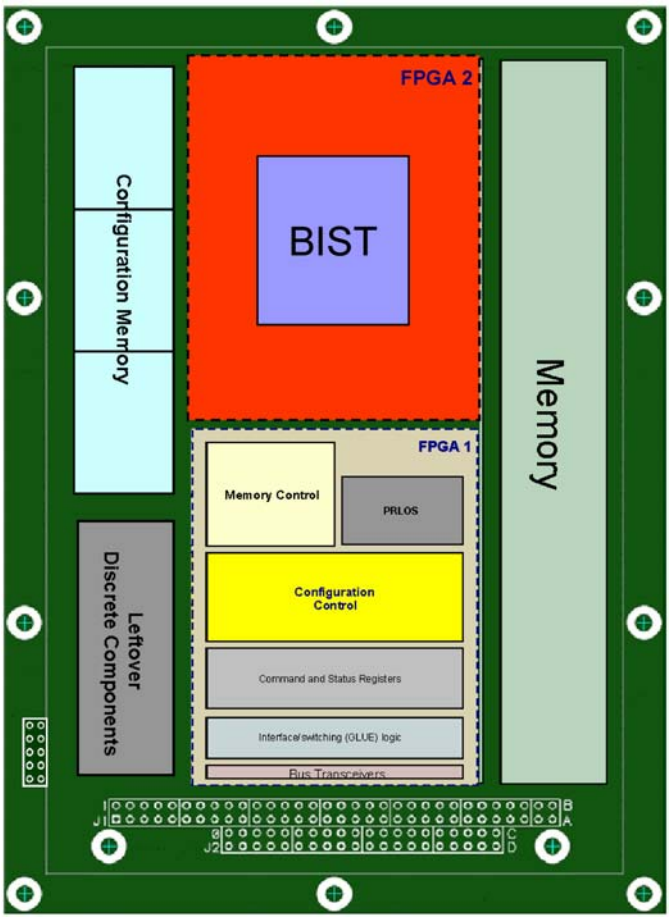


Figure 11. BIST Conceptual Illustration. (After Ref. [4].)

The objective behind creating the BIST is two-fold. First, in development, the CFTP needs a functional test to verify its mechanical and electrical performance. This functional test is performed at ambient temperature and pressure conditions prior to any simulated space environmental tests in order to establish a performance baseline. After being subjected to the required environments, additional functional tests are conducted to determine the impact of the environments on the CFTP. This process is used to qualify the CFTP design for spaceflight [27]. Second, once the CFTP is on orbit and separated

from the development laboratory environment, a means is needed to diagnose the hardware and its interconnect. Because the CFTP is based on a reprogrammable, reconfigurable design, the physical condition of the system is critical to CFTP's functionality. Therefore, while on-orbit at power-on/reset or when the system becomes suspect, the BIST provides a method to diagnose any faults that occur due to launch or on-orbit environmental conditions.

The following sections detail each portion of the BIST design and how they address the hardware self-test for each component; specifically, determining if the CUT (i.e., RAM, PROM, EPROM, Flash Memory, or FPGA) is faulty or fault-free.

B. RANDOM ACCESS MEMORY TESTING

Once the prototype CFTP is ready, assurance that RAM components are wired correctly is needed. Additionally, during initial operations in space, confirmation that the various memory chips are working properly is also needed, as the launch environment or radiation effects may have altered their operation. It may also be desired to test the RAM each time the system is powered-on or reset. Correct operation of the RAM components is needed to provide reliable System-Memory for the CFTP system.

At first, developing a Random Access Memory (RAM) test seems like a fairly simple task. However, a look at the problem more closely proves that it can be difficult to detect subtle memory problems with a simple test. In fact, it is tempting to mistakenly test only for internal memory failures and neglect other possible connections to memory problems.

The purpose of RAM testing is to confirm that each storage location in a memory chip is working. In other words, if a value is stored at a particular address, it is expected to find that value stored there until another value is written to that same address. The idea behind the memory test, then, is to write some set of data to each address in the memory chip and verify the data by reading it back. If all the values read back are the same as those that were written, then the memory chip is said to pass the test. Careful selection of the set of data values must be made to insure that a passing result is meaningful.

Unfortunately, this type of Memory-test is destructive. The process of testing the memory causes one to overwrite its contents. Since it is not desired to overwrite the contents of nonvolatile memories (i.e., configuration-storage), these tests can only be used for RAM testing.

1. Memory Problems

Before implementing any test algorithms, the types of memory problems that are likely to occur need to be identified. Because the manufacturers of memory chips perform a variety of post-production tests on each batch of chips, if there is a problem with a particular batch, it is extremely unlikely that one of the bad chips will make its way into our system.

The one type of memory chip problem that may be encountered is one due to launch or space environment conditions. On-orbit and space-based applications must consider the effects that the space environment may have on electrical components. If a high-energy charged particle penetrates a susceptible portion of the memory structure shown in Figure 12, it may produce adverse affects on the expected functionality. In short, a Single Event Upset (SEU) could affect the memory chip itself or the memory controller in the FPGA. Additionally, the memory chip's physical connection to the Printed Circuit Board (PCB) may be altered as a result of the launch or on-orbit environment also affecting the expected functionality.

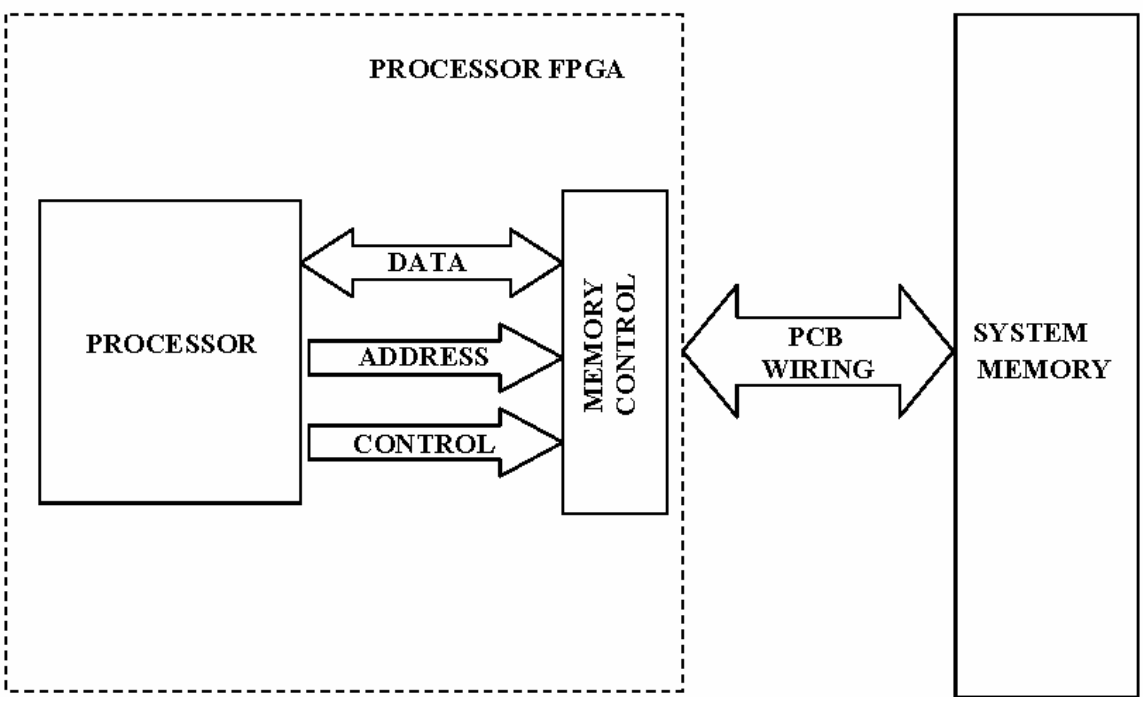


Figure 12. Basic Memory Structure.

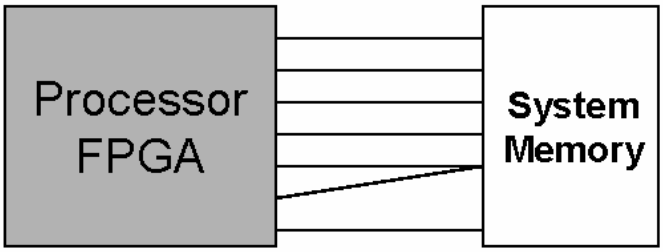
A problem encountered with the internal working of the memory chip itself is a catastrophic failure. Any sort of physical or electrical damage to the chip would cause this. An internal memory failure will affect a large portion of the chip and therefore should be detected by any decent test algorithm. Therefore, the goal of the Memory-test is to be able to detect internal memory failures without specifically looking for them.

A more probable source of actual memory problems will be the memory-to-FPGA interconnect. The following is a look at the interconnect problems in more detail.

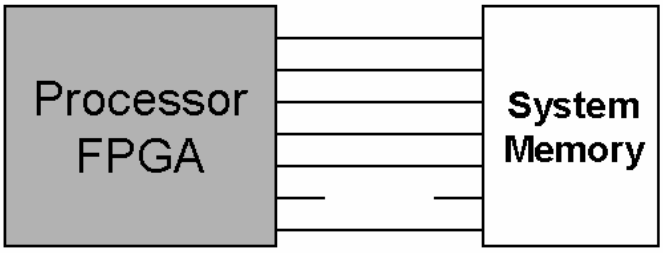
a. Electrical Wiring Problems

An electrical wiring problem can be caused by an SEU altering the interface configuration or actual data being transferred. Each of the wires that connect the memory chip to the processor is one of three types: an address line, a data line, or a control line. The address and data lines are used to select the memory location and to transfer the data, respectively. The control lines tell the memory chip whether the processor wants to read or write the location and precisely when the data will be transferred.

Unfortunately, one or more of these wires could be altered in such a way that it is either shorted (i.e., connected to another wire in the circuit) or open (not connected to anything). Both cases are illustrated in Figure 13.



(a) Shorted Wire



(b) Open Wire

Figure 13. Possible Wiring Problems.

Problems with the electrical connections to the processor will cause the memory chip to behave incorrectly. Data may be stored incorrectly, stored at the wrong address, or not stored at all. Each of these symptoms can be explained by wiring problems on the data, address, and control lines, respectively.

If the problem is with a data line, either several data bits may appear to be "stuck together" (i.e., two or more bits always contain the same value, regardless of the data transmitted) or a data bit may be "stuck-at-one" (always 1) or "stuck-at-zero" (always 0). These problems can be detected by writing a sequence of data values designed to test that each data pin can be set to 0 and 1, independently of all the others.

If the problem is with an address line, the contents of two memory locations may appear to overlap. In other words, data written to one address will actually overwrite the contents of another address instead. This happens because an address bit that is shorted or open will cause the memory chip to see a different address than the one selected by the processor.

Another possibility is that one of the control lines is shorted or open. Unfortunately, if there is a problem with a control line, the memory will probably not work at all, and this will be detected by all of the memory tests.

b. Chip Connection Problems

If a memory chip connection is affected, the system will usually behave as though there is a wiring problem or a missing chip. In other words, some number of the pins on the memory chip will either not be connected to the PCB at all or will be connected at the wrong place. These pins will be part of the data bus, address bus, or control wiring. So as long as the test checks for wiring problems, any improperly connected chips will be detected automatically.

2. Developing a Test Strategy

RAM testing must be able to detect both internal and interconnect errors. Internal errors will probably be catastrophic in nature and will be detected by any test. A more likely source of problems is the memory interconnect, where a wiring problem may occur or a memory chip may be improperly connected.

By carefully selecting the pattern and the order in which the addresses are tested, it will be possible to detect all of the memory problems described above. By also breaking the RAM testing into small pieces, the efficiency of the overall test and the diagnosibility of the schematics/code will be improved.

Three individual RAM tests will be used: a Data-Bus test, an Address-Bus test, and a Memory-Chip test. The first two test for electrical wiring problems, while the third is intended to detect catastrophic failures. As an unintended consequence, the Memory-Chip test will also uncover problems with the control bus wiring, though it cannot provide useful information about the source of such a problem.

The order in which these three tests are executed is important. As depicted in Figure 14, the proper order is: Data-Bus test first, followed by the Address-Bus test, and then the Memory-Chip test. This is because the Address-Bus test assumes a working data bus, and the Memory-Chip test results are meaningless unless both the address and data buses are known to be good. If any of the tests fail, the data value or address at which the test failed will help isolate the problem.

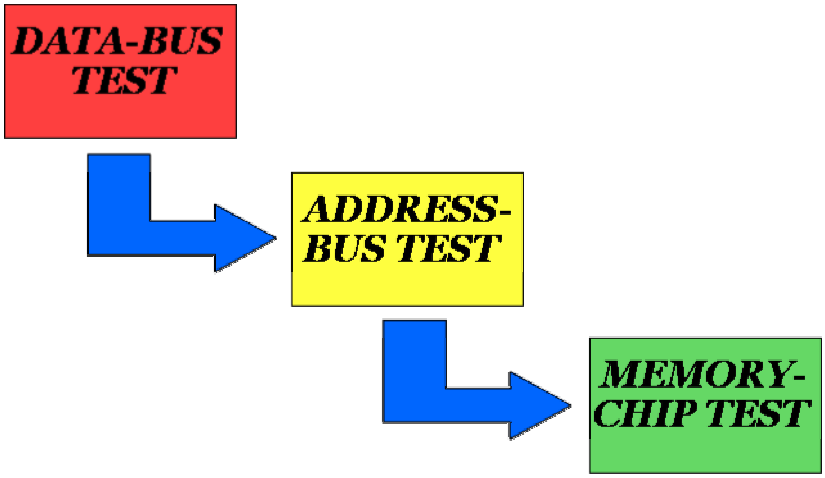


Figure 14. Proper Order of Memory-test Components.

3. Data-Bus Test

The first test is the Data-Bus test. This test should confirm that the memory chip correctly receives any value placed on the data-bus by the processor. The most obvious test is to write all possible data values and verify that the memory chip stores each one successfully. However, that is not the most efficient way to test the data bus. A faster method is to test the bus one bit at a time. The data-bus passes the test if each data bit can be set to 0 and 1, independently of the other data bits.

To test each bit independently a method called the “sliding ones test” will be performed [28]. Table 4 shows the data patterns used in 8-bit and 24-bit versions of this test. The name of this test, sliding ones, comes from the fact that a single data bit is set to one and it “slides” through the entire data word. The number of data values to test is the

same as the width of the data bus. This reduces the number of test patterns from 2^n to n, where n is the width of the data bus.

Test Pattern	8-Bit Binary Pattern	24-Bit Hex Pattern
1	0000 0001	0000 0001
2	0000 0010	0000 0002
3	0000 0100	0000 0004
4	0000 1000	0000 0008
5	0001 0000	0000 0010
6	0010 0000	0000 0020
7	0100 0000	0000 0040
8	1000 0000	0000 0080
…	…	…
24	NA	0080 0000

Table 4. Consecutive Data Values for the Sliding 1's Test.

Since only the data-bus is being tested at this point, all of the data values can be written to the same address. Any address within the memory chip will do.

To perform the sliding ones test, the first data value in the table is written, read back and verified, the second value is written, read and verified, etc. When the end of the table is reached, the test is completed. If the data test fails, it will return the data value for which the test failed. The bit that is set in the returned value corresponds to the first faulty data line, if any.

4. Address-Bus Test

After confirming that the data-bus works properly, the next test is the Address-Bus test. As mentioned earlier, address-bus problems lead to overlapping memory locations. There are many possible addresses that could overlap. However, it is not necessary to check every possible combination. Instead, following the example of the Data-Bus test, each address bit will be isolated during testing. Then the test will verify that each of the address pins can be set to zero and one without affecting any of the others.

The smallest set of addresses that will cover all possible combinations is the set of "power-of-two" addresses [28]. These addresses are analogous to the set of data values

used in the sliding ones test. The corresponding memory locations are 00001h, 00002h, 00004h, 00008h, 00010h, 00020h, etc. (see Table 5). In addition, address 00000h must also be tested. The condition of overlapping locations makes the Address-Bus test harder to implement. After writing to one of the addresses, the test must check that none of the others have been overwritten.

8-bit Hex Address	8-bit Binary Address
00h	0000 0000
01h	0000 0001
02h	0000 0010
04h	0000 0100
08h	0000 1000
10h	0001 0000
20h	0010 0000

Table 5. "Power-of-Two" Addresses.

To confirm that no two memory locations overlap, the test will first write some initial data value at each power-of-two address within the memory chip (e.g., 1010 1010 or AAh). Then a new value, an inverted copy of the initial value, is written to the first test address (e.g., 0101 0101 or 55h), and verified that the initial data value is still stored at every power-of-two address. If a location is found, other than the one just written to, that contains the new data value, a problem with the current address bit has been found. If the Address-Bus test fails, the address at which the error was detected will be returned.

5. Memory-Chip Test

Once the address and data-bus wiring are verified as working, it is necessary to test the integrity of the memory chip itself. Every bit in the memory chip must be tested to see if it is capable of holding both zero and one. This is a fairly straightforward test to implement, but takes significantly longer to execute than the previous two.

For a complete Memory-Chip test, the test must visit (write and verify) every memory location twice [28]. Any data value can be chosen for the first pass, so long as the test inverts that value during the second. A simple example is an "increment test" and "decrement test."

The 24-bit data values for an increment test are shown in the first two columns of Table 6. The third column shows the inverted data values used during the second pass of the test. The latter represents a decrement test.

Note that the memory locations are offset from the increment values. Because each memory location is verified/read immediately following the corresponding write, it is possible that the data read back would be just the voltage remaining on the data-bus from the previous write [28]. If this occurs, it will appear as though the data has been correctly stored in memory. In fact, the memory chip could be completely disconnected and the test would still appear successful. By selecting a set of data that changes with the address but is not equivalent to that address, this can be prevented. If the Memory-Chip test fails, the address containing an incorrect data value is returned.

Memory Offset	Binary Value	Inverted Value
00h	000001	111110
01h	000010	111101
02h	000011	111100
03h	000100	111011
...	...	...
3Eh	111111	000000
3Fh	000000	111111

Table 6. Data Values for an Increment Test.

6. Designing the RAM Test

With the basic theory behind the RAM self-test presented previously, the remainder of this section will discuss the development of the programmable System-Memory self-test circuit implemented within an FPGA.

a. Overview

The block diagram in Figure 15 is the RAM test circuit which is implemented within an FPGA for testing of CFTP's multiple SDRAM (System-Memory) chips. The precise location and mode of a detected memory failure is stored in a Status Register, which is routed directly to output pins. These output pins are connected from the Processor FPGA to the PC104 Bus through the Controller FPGA. This data is avail-

able immediately after an Output Response Analyzer (ORA) module (e.g., the Comparator) triggers the fail signal. Any time the expected data does not match the actual data registered in the Comparator, the device asserts a fail flag, FLAG; this fail signal is also ported to an output pin.

A single clock is used to toggle through the System-Memory addresses and control the reading, writing and evaluation of data.

Once invoked, an external reset signal, RESTART, will restart the Memory-test into the first test state. During the reset state, the test State Machine will remain in a wait state with all internal registers set to zero.

A pass signal will be directly connected to an output pin. The pass signal, PASS_ENABLE, will become enabled upon completion of the entire test cycle. A counter will store the number of Memory-test cycles that the memory has been able to pass without failure.

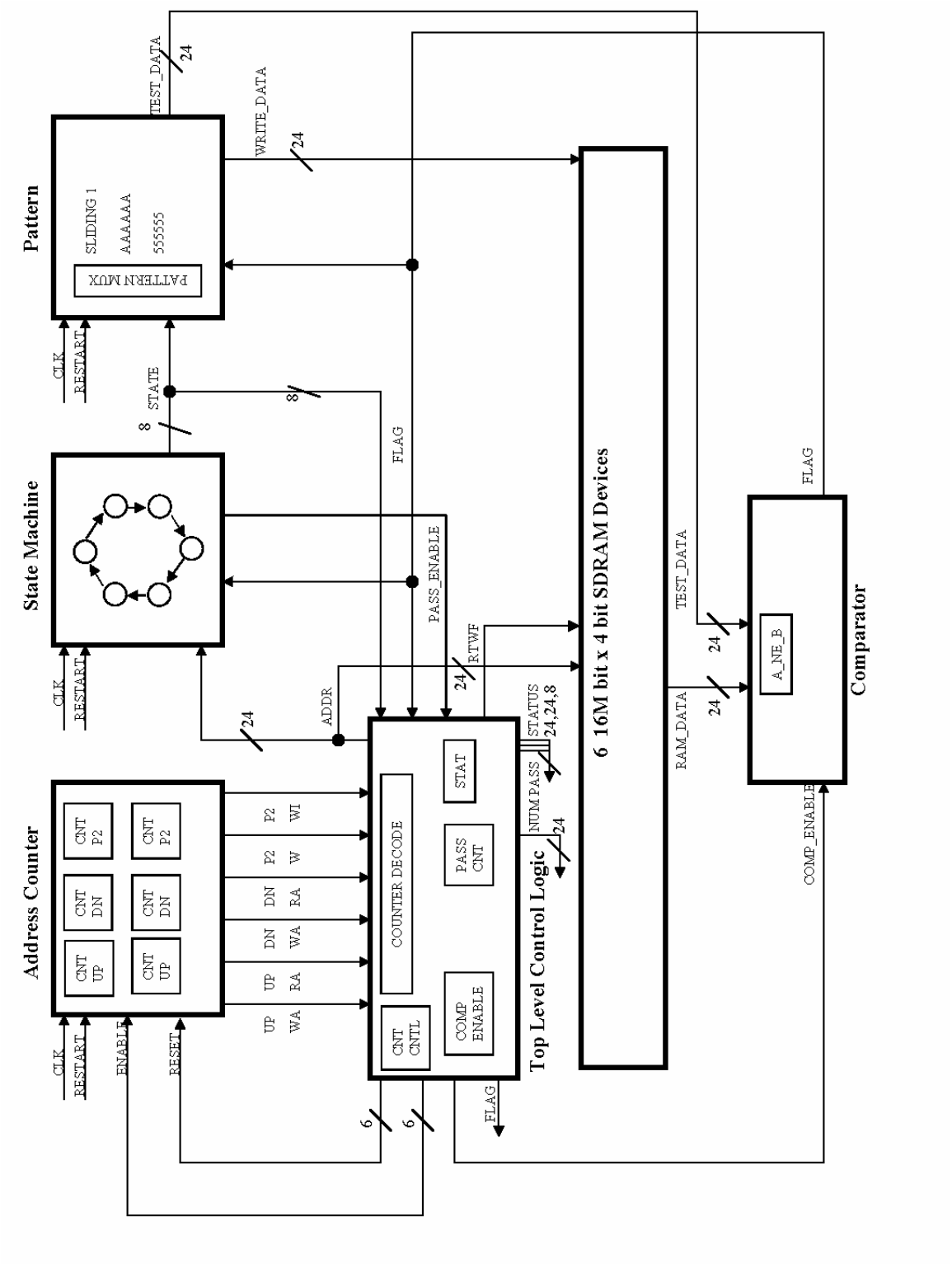


Figure 15. Block Diagram of the RAM Test.

b. Circuit Under Test (RAM)

All six of the 16-Mbit x 4-bit SDRAM banks are cascaded into a 16-Mbit x 24-bit RAM module. All data-multiplexing logic is contained in this module. In each test, data is written to specific addresses in the SDRAM array and checked some clock cycles later (depending on the test) for correctness in the Comparator.

c. Test Pattern Generator (Pattern)

The three different 24-bit words listed in Table 7 are written to the RAM array and read back some time later at various times during the RAM Test. The different patterns are generated in a Test Pattern Generator (TPG) module called Pattern and are selectively applied to the RAM array depending on the current test state.

Pattern 1	Sliding 1s
Pattern 2	AAAAAAh
Pattern 3	555555h

Table 7. RAM Test Patterns.

The Pattern module, shown in Figure 16 (see Appendix A for complete internal schematics and VHDL code), consists of three inputs (*clock, state, restart,* and *flag*) and two outputs (*test_data* and *write_data*). The *clock* pin is connected to the single clock used for the whole system and the *restart* pin is connected to system's external reset signal. The *flag* pin is connected to the Comparator and is used to notify the Pattern module that a test has failed. The 8-bit *state* bus is connected to the State Machine module and is used by the Pattern module to base decisions on which pattern from Table 7 to output. The two outputs *test_data* and *write_data* are connected to the Comparator and RAM, respectively, and carry the generated output pattern.

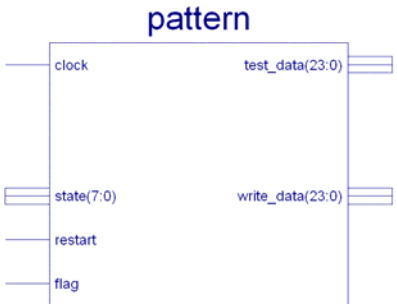
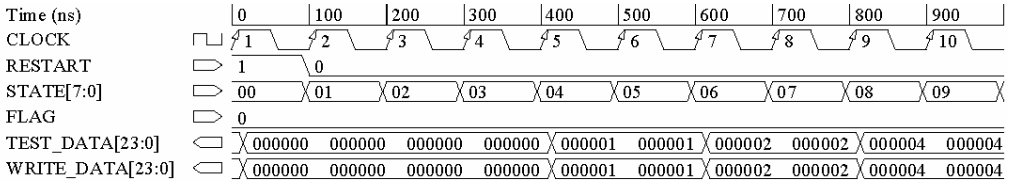
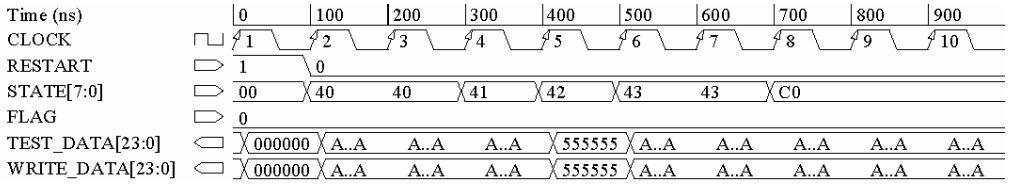


Figure 16. Pattern Module.

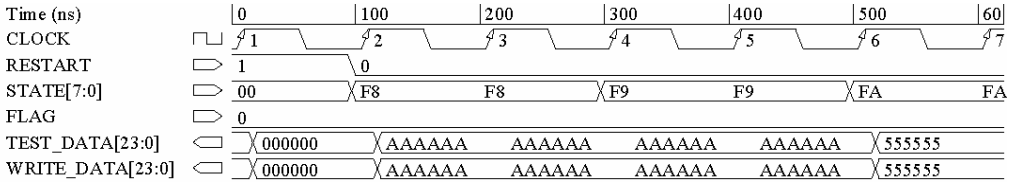
The three partial Test-Bench waveforms shown in Figure 17 (see Appendix A for a complete Test-Bench waveform) demonstrate the module's ability to output varying patterns for varying tests based on the current state. Figure 17(a) shows the sliding-ones pattern during the Data-Bus test being written and read back, then written and read back, etc. Figure 17(b) demonstrates the module writing an initial pattern to memory, then an inverted pattern, and finally a pattern to compare when data is read back from memory. Lastly, Figure 17(c) shows the module writing and reading a pattern then writing and read another pattern; therefore, performing a write and read twice at each location.



(a) Data-Bus-Test Example



(b) Address-Bus-Test Example



(c) Memory-Chip-Test Example

Figure 17. Pattern Test-Bench Waveform.

d. Output Response Analyzer (Comparator)

The module entitled Comparator is designed to compare a 24-bit word which is read from the RAM to the same test data word which was written to the RAM address some clock cycles earlier (again depending on the test). Whenever a fault is detected during one of the test states, the fail flag is asserted. As long as the test data from the data generator matches that which was read from RAM, the fail flag is not asserted. Testing continues until the clock is disabled or the external reset is asserted.

The Comparator module, shown in Figure 19 (see Appendix A for complete internal schematics and VHDL code), consists of three inputs (*ram_data, test_data,* and *comp_enable*) and one output (*fail*). The 24-bit bus *ram_data* is connected to the System-Memory and its input is compared by the Comparator module to the *test_data-bus* input from the Pattern module. Likewise, the 24-bit bus *test_data* is connected to the Pattern module and is compared by the Comparator module to the System-Memory output. The *comp_enable* pin is connected to the Top-Level Control Logic module and its input controls when the Comparator should conduct a comparison. The *fail* pin outputs the result of the comparison, and generates a signal, *flag*, if the comparison fails.

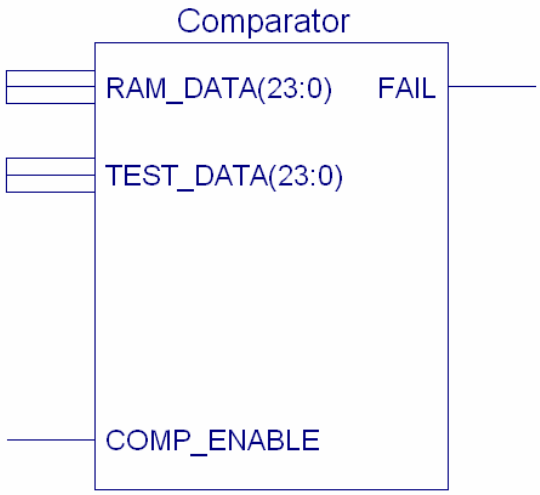


Figure 18. Comparator Module.

A partial Test-Bench waveform shown in Figure 19 (see Appendix A for a complete Test-Bench waveform) demonstrates the module's ability to compare two 24-bit signals and output a *fail* signal if any bit(s) does not match.

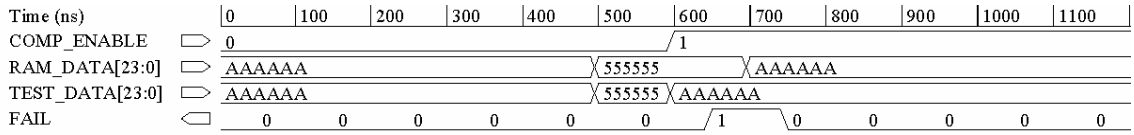
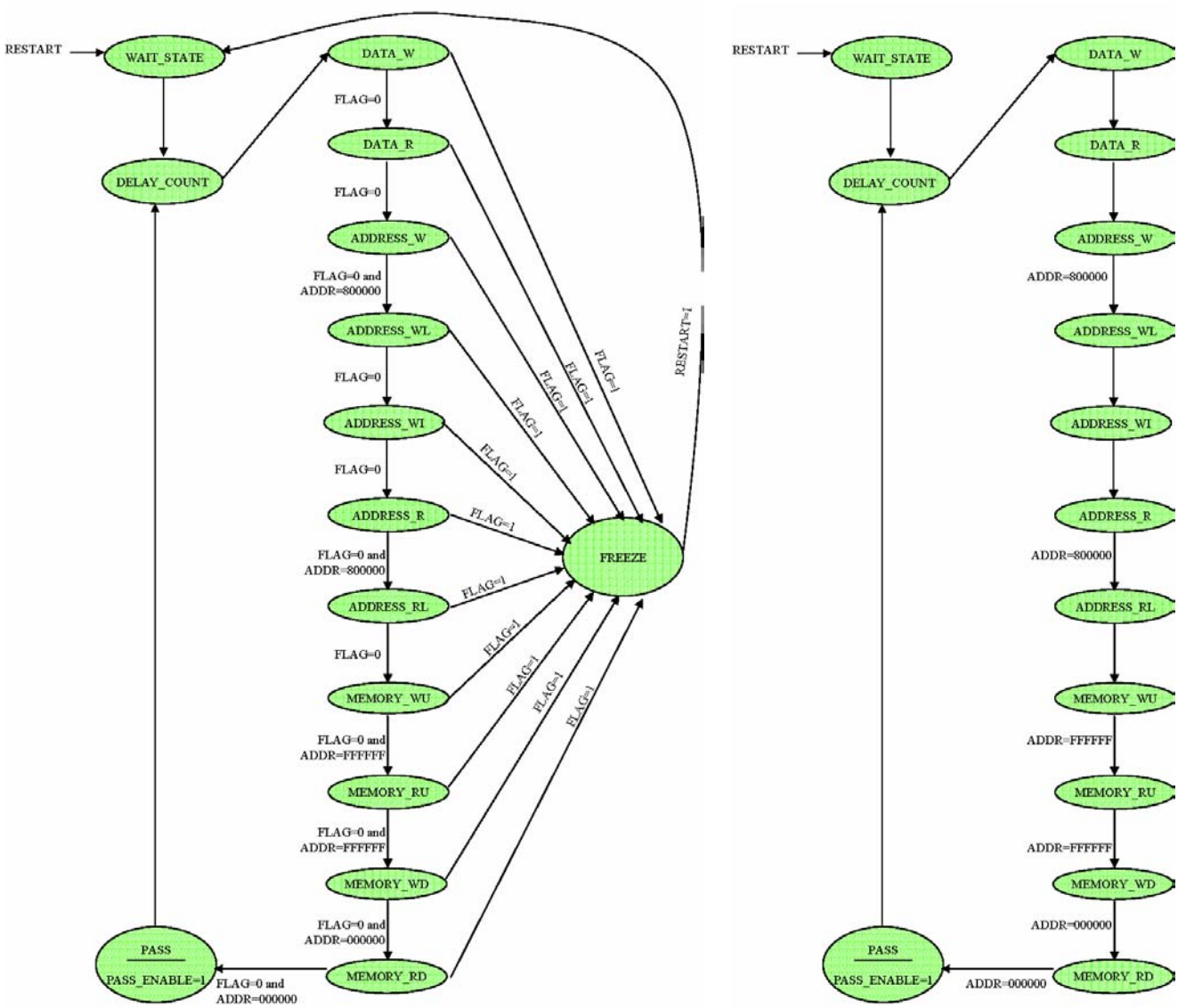


Figure 19. Comparator Test-Bench Waveform.

e. State Machine

Initially, the State Machine was designed with a very low threshold for faults. As shown in Figure 20(a), if a single fault was detected in any test state the state machine would transition to a freeze state (halting the execution of the test) and remain there until the external reset signal was asserted. This low threshold design provides very little data for diagnosis. With the one fault, a single capture of the test's status is provided for diagnosis in this RAM test design. Conversely, a design with a high threshold for faults provides a large amount of data for diagnosis. Because of the remote nature of the CFTP on board a satellite, providing as much information as possible allows the CFTP to take advantage of its reprogrammability/reconfigurability and design around the potential faults. Figure 20(b) illustrates the chosen state machine with a higher threshold. With each fault experienced, a capture of the test's state is stored and the test is continued.



(a) Low Threshold State Machine (b) High Threshold State Machine

Figure 20. RAM Test State Machine.

Fourteen different tests (shown in Table 8) must be conducted in order to implement the Data, Address and Memory-Chip tests that were described earlier. Within each individual test, there are specific operations that are synchronized to a variety of signals, such as enables, resets and the write and read address. Since these tests and operations must be conducted in a specific order and must transition to subsequent operations depending on the outcome of each test, a State Machine module is used for arbitrating the Memory Test. There are fourteen different states that are controlled by the state machine logic. Also included in the state machine are delay counters. These counters set the wait intervals that are necessary for ensuring clean test transitions as well as to create the delay needed for data retention testing.

State Name	Operation Performed	Address Counter	Pattern
WAIT	Wait	N/A	N/A
DELAY_ COUNT	Count TBD Clock cycles	N/A	N/A
DATA_W	Write Sliding 1s (at base address)	N/A	Sliding 1s
DATA_R	Read Sliding 1s (at base address)	N/A	Sliding 1s
ADDRESS_W	Write AAAAAAh	P2_W	AAAAAAh
ADDRESS_WL	Write AAAAAAh (to last address 800000h)	P2_W	AAAAAAh
ADDRESS_WI	Write 555555h (inverted pattern)	P2_WI	555555h
ADDRESS_R	Read AAAAAAh	P2_W	AAAAAAh
ADDRESS_RL	Read AAAAAAh (from last address 800000h)	P2_W	AAAAAAh
MEMORY_WU	Write UP	UP_WA	AAAAAAh
MEMORY_RU	Read UP	UP_RA	AAAAAAh
MEMORY_WD	Write DOWN	DN_WA	555555h
MEMORY_RD	Read DOWN	DN_RA	555555h
PASS	Pass state. Continue testing until clock stops	N/A	N/A

Table 8. Description of the states used in the Memory Test.

The State Machine module, shown in Figure 21 (see Appendix A for complete internal schematics and VHDL code), consists of four inputs (*clock, restart, flag,* and *addr*) and two outputs (*pass_enable* and *state*). The *clock* pin is connected to the single clock used for the whole system and the *restart* pin is connected to system's external reset signal. The *flag* pin is connected to the Comparator and is used to notify the State Machine module that a test has failed. The 24-bit *addr* bus is connected to the Top-Level Control Logic module and feeds the current memory address to the state machine. The state machine uses this address signal to help determine state transitions. The *pass_enable* pin is asserted by the state machine when it completes all of the states required to test the RAM. This is used by the Top-Level Control Logic module to count the number of times the BIST completes a full RAM test. The 8-bit *state* bus outputs a number sequence for each state. This output is used by the Pattern and Top-Level Control Logic modules to execute specific tasks during specific states.

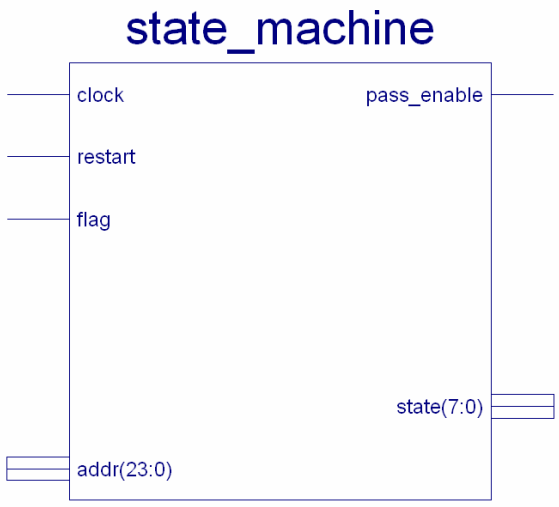
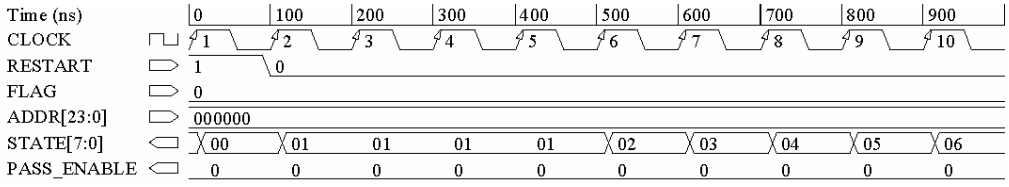
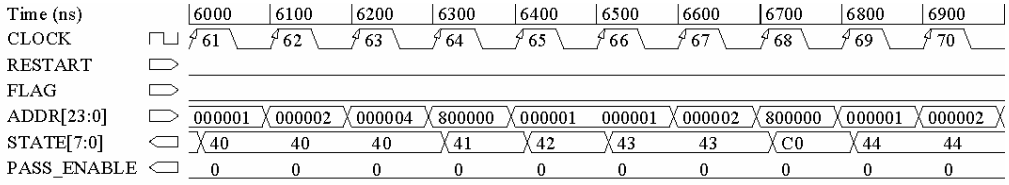


Figure 21. State Machine Module.

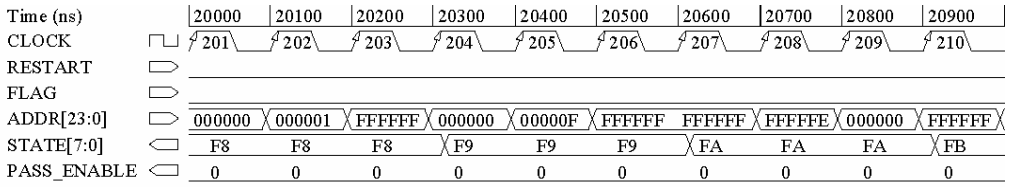
The three partial Test-Bench waveforms shown in Figure 22 (see Appendix A for a complete Test-Bench waveform) demonstrate the module's ability to take in address inputs and make transitions between states. Figure 22(a) shows the module incrementing through the different write and read states during the Data-Bus test. Figure 22(b) demonstrates the module executing the state which writes to all the power-of-two addresses, until the last address (i.e., 800000h) is reached and the state transitions from state 40 to 41. In state 41 the state writes to the last address. In state 42, the inverted pattern is written to memory. Finally, during state 43 and C0, the patterns are read and verified from each power-of-two address (C0 reads the last address, 800000h). Figure 22(c) shows the module transitioning between states which incrementally write and read (i.e., state F8 and F9 respectively) and states which decrementally write and read (i.e., FA and FB respectively).



(a) Data-Bus Test Example



(b) Address-Bus Test Example



(c) Memory-Chip Test Example

Figure 22. State Machine Test-Bench Waveform.

f. Address Counter (Counter)

Every address in the RAM array under test is evaluated many times; the use of counters to supply encoded address bits to the memory address decoders provide an easy way to synchronously address the embedded RAM array during self-testing. Address Counter-Control logic is contained in the Test Controller module (i.e., the Top-Level Control Logic) where the counters are instantiated. Some of the RAM tests require testing to begin at the first address line and count up, begin at the last address line and count down, or count in powers-of-two. Therefore, an up-counter module, a down-counter module, and a power-of-two module are used. Six different types of 24-bit address counters (16M RAM addresses map to 2^{24} bits) are instantiated in the Top-Level-Control Logic module: read-address up counter (UP_RA), write-address up counter (UP_WA), read-address down counter (DN_RA), write-address down counter (DN_WA), write power-of-two address counter (P2_W), and write-inverted[3] power-of-two address counter (P2_WI).

The Counter module, shown in Figure 23 (see Appendix A for complete internal schematics and VHDL code), consists of four inputs (*clock, enable, reset*, and *restart*) and two outputs (*pass_enable* and *state*). The *clock* pin is connected to the single clock used for the whole system and the *restart* pin is connected to system's external reset signal. The 6-bit *enable* bus is connected to the Top-Level Control Logic module that controls which of the six address counters is enabled. The 6-bit *reset* bus is also connected to the Top-Level Control Logic module that controls when the counters are reset. The 24-bit buses *UP_WA, UP_RA, DN_WA,* and *DN_RA* are used during the Memory-Chip Test to provide the appropriate incrementing/decrementing address counter. During the Address Test, the 24-bit buses *P2_W* and *P2_WI* provide the addresses for writing a pattern (i.e., AAAAAAh) and an inverted pattern (i.e., 555555h) respectively. During the Data Test, a hardwired memory address is used.

[3] Write-inverted refers to the fact that this counter is used specifically for the Address Test states where an inverted copy of the pattern is written to memory.

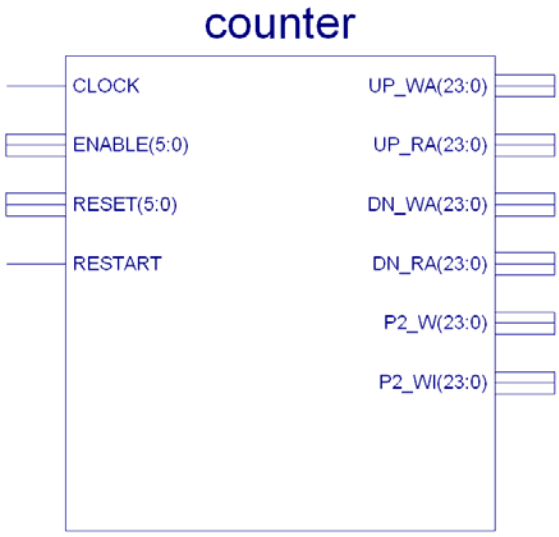


Figure 23. Counter Module.

A partial Test-Bench waveform shown in Figure 24 (see Appendix A for a complete Test-Bench waveform) demonstrates the module's ability to enable and reset specific counters in any combination.

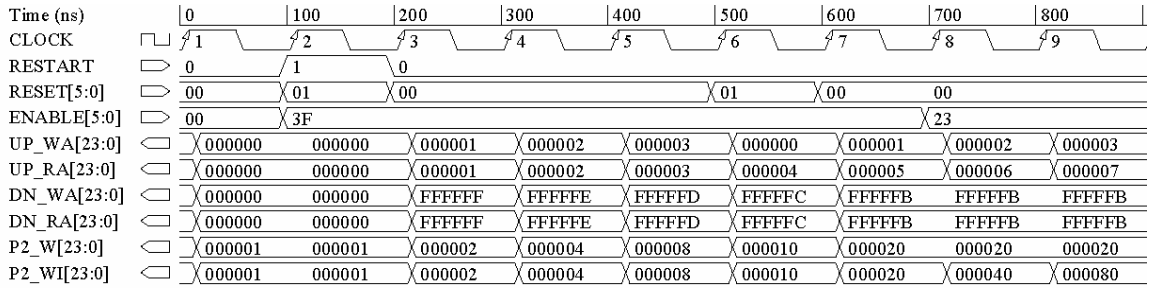


Figure 24. Counter Test-Bench Waveform.

g. Test Controller (Top-Level Control Logic)

All of the modules above are instantiated and connected together in the Top-Level Control Logic module. The Top-Level Control Logic module also contains control logic to handle the following tasks:

- Synchronously control the read/write address counter enables and resets with inputs from the state machine.
- Arbitrate which counters (up or power-of-two) are used to send encoded address bits to RAM.

- Dictate to the Comparator when it is to test.
- Create a counter for keeping track of the number of consecutive passing test cycles. The PASS_ENABLE signal for this counter is asserted by the state machine module at the completion of each passing test cycle.
- Assign internal test signals to output pins in order to enhance the observability of the RAM self-testing.

Five modules are used to accomplish these different tasks; they are: Counter-Control, Counter-Decode, Compare-Enable, Pass-Counter, and Status.

(1) Counter-Control Module. The Counter-Control module, shown in Figure 25 (see Appendix A for complete internal schematics and VHDL code), consists of one input (*state*) and two outputs (*reset* and *enable*). Based on the current state fed to the module, the Counter-Control module determines the appropriate reset and enable commands for the Counter module.

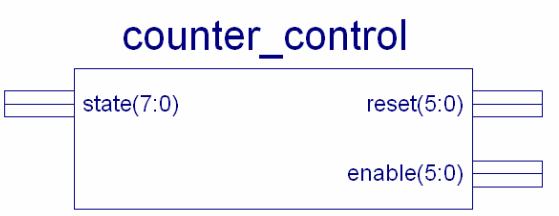


Figure 25. Counter-Control Module.

(2) Counter-Decode Module. The Counter-Decode module, shown in Figure 26 (see Appendix A for complete internal schematics and VHDL code), consists of seven inputs (*state, address1, address2, address3, address4, address5,* and *address6*) and two outputs (*rtwf* and *addr*). Based on the current state fed to the module, the Counter-Decode module determines which address-bus from the Counter module should be routed to the RAM and State Machine module.

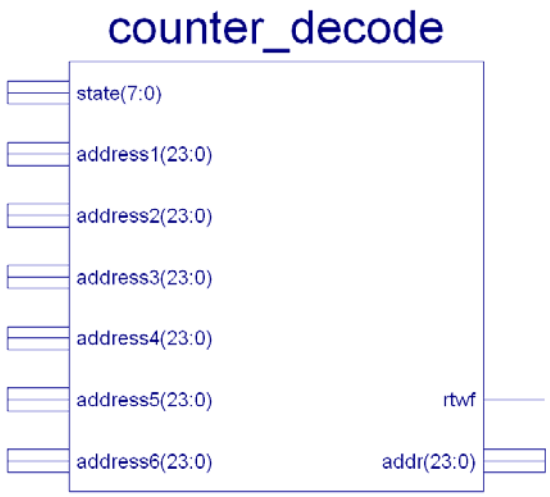


Figure 26. Counter-Decode Module.

(3) Compare-Enable Module. The Compare-Enable module, shown in Figure 27 (see Appendix A for complete internal schematics and VHDL code), consists of one input (*state*) and one output (*comp_enable*). Based on the current state fed to the module, the Compare-Enable module asserts the *comp_enable* signal to enable the Comparator module to compare the output of the memory location (i.e., *ram_data*) with the expected pattern (i.e., *test_data*).

Figure 27. Compare-Enable Module.

(4) Pass-Counter Module. The Pass-Counter module, shown in Figure 28 (see Appendix A for complete internal schematics and VHDL code), consists of three inputs (*enable, clock,* and *reset*) and one output (*Num_passes*). The Pass-Counter module is a simple counter. The module is clocked by the *pass_enable* signal generated by the State Machine Module. Each time the state machine completes the test sequence and asserts the *pass_enable* signal, the Pass-Counter module will be clocked and therefore increment its counter.

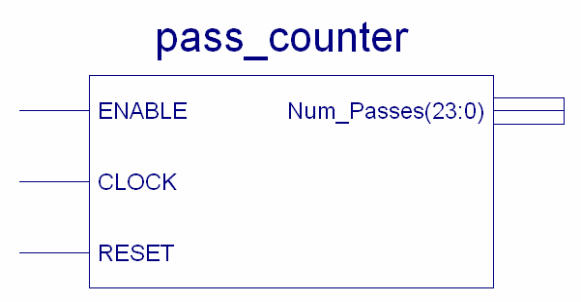


Figure 28. Pass-Counter Module.

(5) Status Module. The Status module, shown in Figure 29 (see Appendix A for complete internal schematics and VHDL code), consists of four inputs (*flag, state, addr* and *test_data*) and three outputs (*location, mode,* and *data*). When the Status module detects that the *flag* signal has been asserted, it captures the current state, address and pattern.

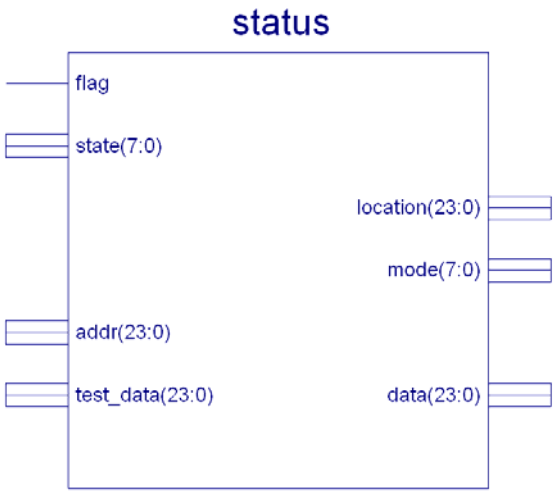


Figure 29. Status Module.

The combination of the last four modules make up the Control Logic Module shown in Figure 30 (see Appendix A for complete internal schematics and VHDL code).

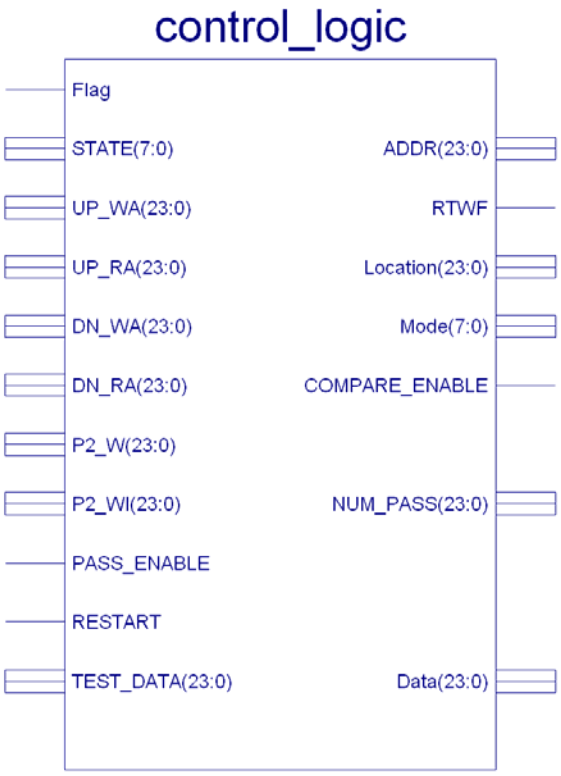


Figure 30. Control Logic Module.

Finally, Figure 31 shows the results of combining all of the modules into the completed design (see Appendix A for complete internal schematics, VHDL code, and Test-Bench waveforms).

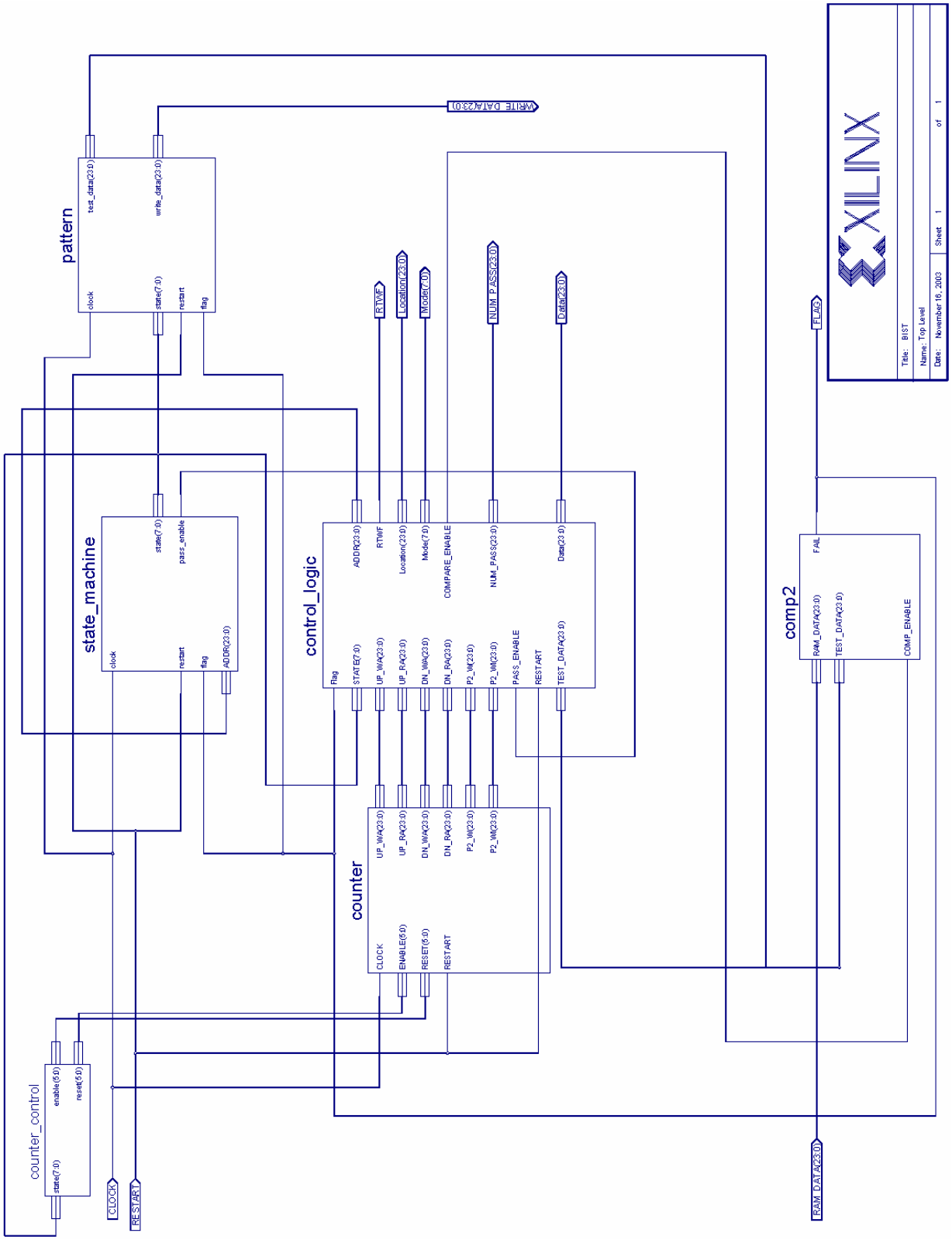


Figure 31. Complete RAM Test Design.

7. Testing the Test

The RAM BIST design shown in Figure 31 does not include a RAM module. During development of the RAM BIST, a place-holder for the CFTP System-Memory was needed but the actual test is connected to memory components. The absence of these memory components in the RAM BIST provides a means to inject errors. One of the *Comparator* module's inputs, *RAM_DATA* which is normally data read from RAM, is now an input to the RAM BIST. In the Test-Bench, this input is used to insert both correct and erroneous data values in order to test the RAM BIST's ability to detect errors.

8. Conclusions and RAM BIST Implementation

This section has provided a detail description of the RAM BIST design and how it addresses the hardware self-test for CFTP's System-Memory components. The RAM BIST is accomplished in a single configuration and is implemented in the Processor FPGA. Access to CFTP's System-Memory is only through the Processor FPGA; therefore, access to RAM from the Controller FPGA requires configuring the Processor FPGA to allow inputs to pass through to outputs. This means implementing the RAM BIST in the Controller FPGA requires two configurations. So, in order to reduce the number of required configurations and maintain the simplest design, the RAM BIST is implemented in the Processor FPGA.

The next section focuses on the design of a data-checksum device used to ensure that data is correctly maintained in the EPROM/PROM and Flash Memory components. It discusses the analysis, design, and implementation of the checksum.

C. READ-ONLY MEMORY TESTING

Once the prototype CFTP is ready, assurance that the Erasable Programmable Read-Only Memory (EPROM), Programmable Read-Only Memory (PROM), and Flash Memory[4] components are wired correctly is needed. Additionally, during initial operations in space, confirmation that the various ROM chips are working properly is also needed, as the launch environment or radiation effects may have altered their operation. It may also be desired to test the EPROM/PROM or Flash Memory each time the system is powered-on or reset. Stored in ROM are the different CFTP configurations. Errors in these will more that likely make a configuration unusable. A method is needed, as with the RAM test, to verify that each storage location in the ROM devices is working. However, unlike the RAM components the EPROM/PROM and Flash Memory are nonvolatile memories, so the method of writing some set of data and verifying the data by reading it back will not work. Instead, the CFTP will use the suspect configuration to make a unique *signature* which can be compared to the *signature* from the correct configuration. The ROM test will utilize a checksum device to produce a sum, the *signature*. This *signature* can then be compared to the stored expected result to determine whether the device has experienced any hardware faults.

The checksum device uses a state machine architecture which is divided into three modules: *System*, *Data*, and *Control*. The *System* module is the overall checksum device, and connects the *Data* and *Control* modules. The *Data* module is the part of the system that stores, moves, and transforms data, using registers to hold data values and multiplexers when multiple inputs are possible [30]. The *Control* module controls the data transfers, the transformations, and the sequencing [30]. Inputs to the *Control* module are the conditions generated by the *Data* module plus the **external** control inputs. The outputs are control signals that are distributed to the corresponding control points in the *Data* module (i.e., multiplexers and registers). There are several common approaches for the implementation of the *Control* module [30]. First, it can be implemented as a *hardwired* controller in the sense that it consists of a fixed *state transition* and *output* definition and

[4] Because Flash Memory is non-volatile memory and is used in the CFTP design similarly to an EPROM device, further mention in this section of ROM will include Flash Memory.

any changes to the controller's behavior would require modifying the state transitions or output definitions. A more general approach is to implement the controller as a *micro-programmed* device, one that has a fixed state transition and output definition part but its actions are programmed much like a regular computer Central Processing Unit (CPU) [29, 30]. The typical implementation methods for controllers are listed in Table 9.

Fixed	**a.**	Register (or counter or shift register) + gates
↑	**b.**	Register (or counter) + multiplexers
↑	**c.**	Register (or counter) + ROM or PLA
↑	**d.**	Programmable Sequential Array
↑	**e.**	Microprogrammed Controller
↑	**f.**	Microprogrammable Controller
Programmable	**g.**	Microprocessor as controller

Table 9. Implementation Approaches for Control Subsystems. (After Ref. [30].)

A general controller architecture is shown in Figure 32. Note that the *System* has **external** inputs and outputs that connect to either of the two **internal** *Data* or *Control* subsystems. As mentioned earlier, the main purpose of the *System* is to connect the *Data* and *Control* subsystems.

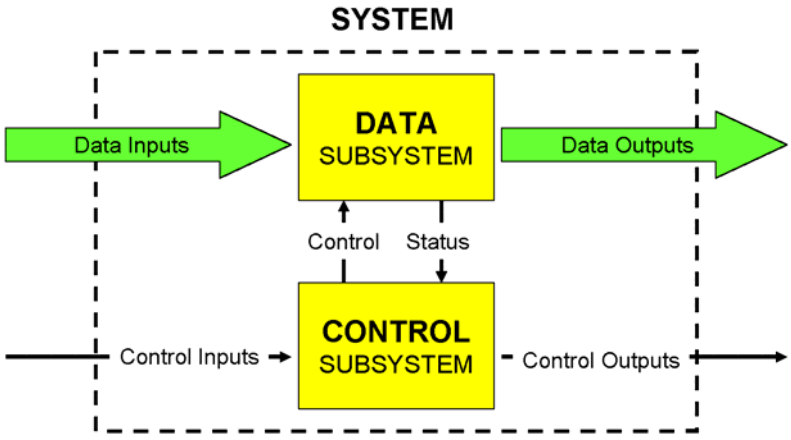


Figure 32. General ROM Test Structure. (After Ref. [30].)

The design methodology includes the following steps.

1. State the Problem to be Solved

The checksum device maintains a running sum of data received over the ROM data bus. The PROM/EPROM and Flash Memory devices selected for CFTP all operate with an 8-bit, parallel interface [22, 31, 24]. Therefore, the data is sent as many bytes of data. The checksum device sums all data presented to it. To operate, the checksum device is set to *run*, given data, and notified that there is *new data* to sum. For example, if the ROM contains the data stored in Table 10, the checksum device is given data inputs of 01, 02, 04, 08 in hexadecimal.

00000001
00000010
00000100
00001000
00010000
…

Table 10. Example ROM contents.

This produces a checksum output of 15, or 0F in hexadecimal. The sequence of **external** *Data* and *Control* signals given to the checksum device follow steps 1–8 from Table 11.

Step	Data(hex)	Run	Newdata	Result(hex)	done
1	01	0	0	00	1
2	01	1	0	00	0
3	01	1	1	01	0
4	02	1	1	03	0
5	04	1	1	07	0
6	08	1	1	0F	0
7	08	1	0	0F	0
8	08	0	0	0F	1
9	10	1	0	0F	0
10	10	1	1	10	0

Table 11. External Data and Control Signals.

Note that step 9 begins new data, indicated by *run* changing from 0 back to 1. The *computed checksum* accumulates as 0, 1, 3, 07, 15 (0, 1, 3, 7, 0F in hexadecimal). In use, the stored checksum and the computed checksum are then compared to determine whether the data had been correctly stored and maintained in ROM. The checksum device must indicate when the checksum has been computed (it is *done)* so that an external comparator can compare the computed and the received checksum for equality. In the example, if both the computed and received checksum are 0F, then the received data is assumed correct.

2. Determine the Inputs and Outputs for the Test Device

From the above problem statement, the required inputs and outputs are determined. It is useful to look at the connections/interface for the CFTP from a black-box perspective. For the checksum device the inputs and outputs are as in Figure 33 where *Data* and the checksum *Result* are *n*-bit inputs and outputs, while *run*, *newdata*, and *done* are single-bit control signals.

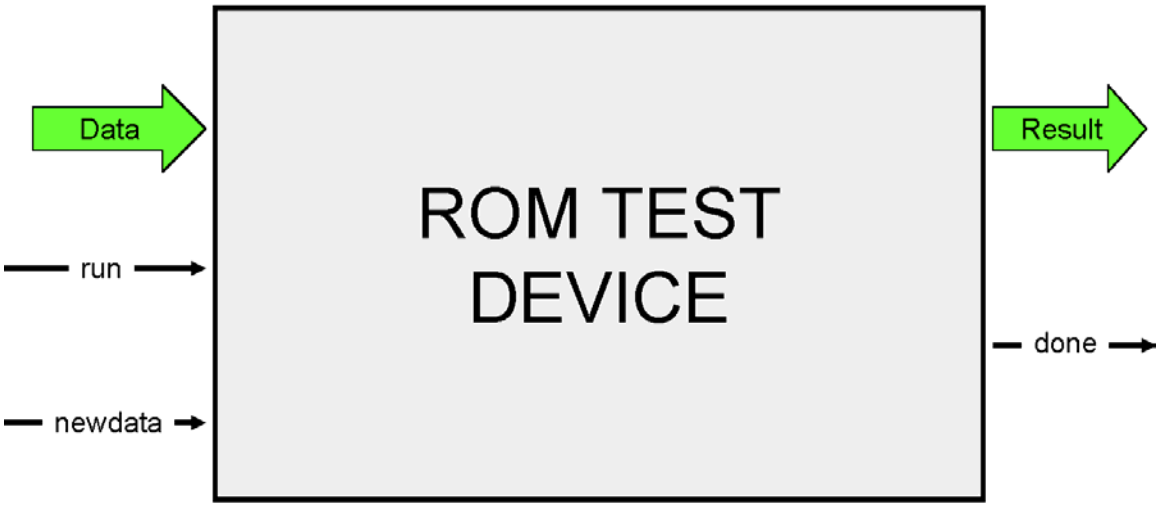
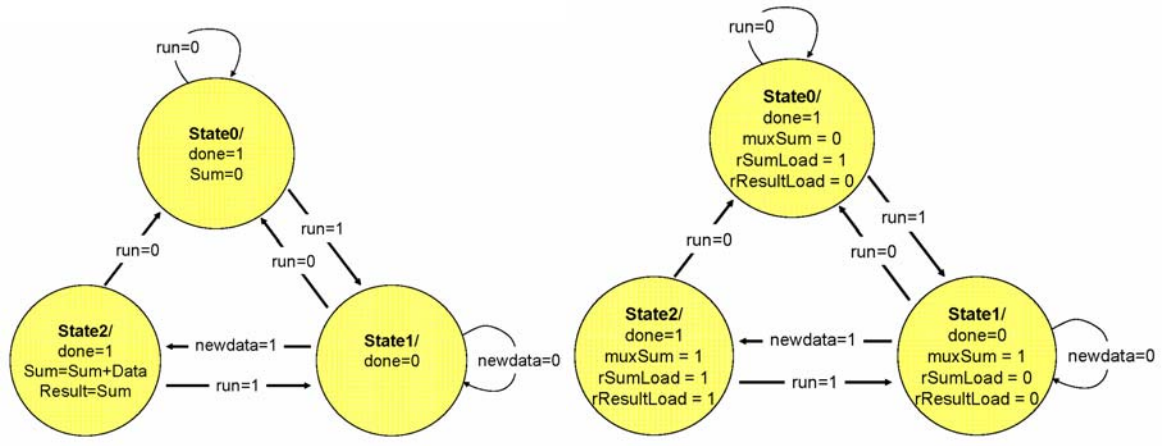


Figure 33. ROM Test Module. (After Ref. [29].)

3. Define the States, Transitions and Outputs of Each State

A state diagram is useful to help determine high-level states and transition conditions [30]. Figure 34 illustrates first the high-level operations necessary (see Figure 34(a)). These high-level operations are further defined as control outputs of multiplexers and registers (see Figure 34(b)).



(a) High Level State Diagram (b) Multiplexer/Register Level State Diagram

Figure 34. ROM Test State Diagram.

This state diagram defines the *Control* module shown in Figure 35 (see Appendix B for complete internal schematics and VHDL code).

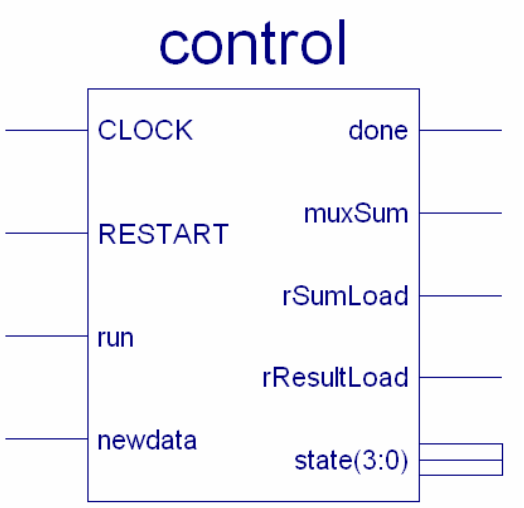


Figure 35. Control Module.

A partial Test-Bench waveform shown in Figure 36 (see Appendix B for a complete Test-Bench waveform) demonstrates the module's ability to take in inputs and make transitions between states.

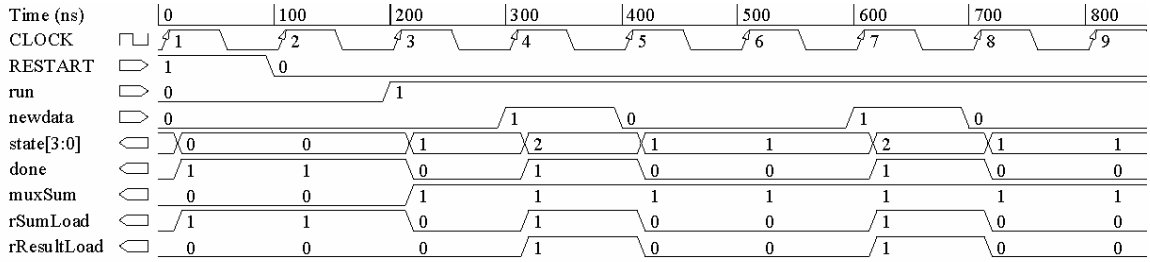


Figure 36. Control Module Test-Bench Waveform.

4. Determine the Computational Modules

The checksum device needs a method to add a registered sum to incoming data. This can be implemented using a simple *Adder* module seen in Figure 37 (see Appendix B for complete internal schematics and VHDL code).

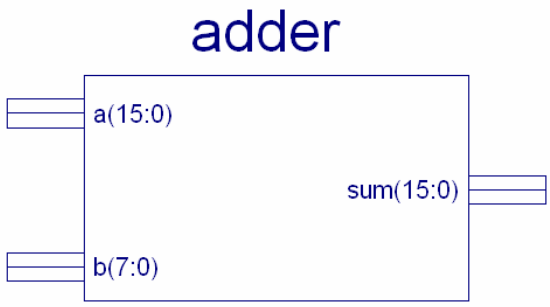


Figure 37. Adder Module.

5. Develop a Data Subsystem Module

To accomplish steps 3–5 of Table 11, data registers are required to hold state information and multiplexers are needed to select between data sources. The following paragraphs describe these.

a. Registers

The general rule of thumb for determining what requires a register (e.g., a flipflop[5]) is: if a value must be maintained through multiple states, make it a register [32]. The other option is to set the value in each state. This includes any internal values and outputs. For the checksum device, there is *Sum, Result,* and *done* that must be maintained through multiple states or set in each state. *Sum* and *Result* must be in *n*-bit registers, *done* in a *single-bit* register. To simplify the design, only the *n*-bit values (*Sum* and *Result)* will be treated as registers (i.e., *rSum* and *rResult*). When to assign the *rSum* register a value is controlled by *rSumLoad*. If it is asserted, the register value changes on the clock edge. The same is true for *rResult.* The purpose of *rResult* is to hold the result after the checksum is computed and *run=0* (execution of the checksum device is stopped), since *Sum=0* when *run=0.*

b. Multiplexers

The need for a multiplexer (mux) is determined by whether a variable has multiple assignments [30]. If a variable has only one assignment, no multiplexer is required. If two assignments, a two-input mux is required, four assignments requires a four input mux, etc. The checksum device has one variable, *Sum*, with two assignments (i.e., *Sum=0,* and *Sum=Sum+Data*). The multiplexer selects the input that the *rSum* register receives. When the control signal *muxSum* is 0, the input of "000000000000000" is selected and when *muxSum* is 1 the input *Sum+Data* is selected.

---

[5] An edge-triggered, clocked storage unit that stores a single bit of data. A low-to-high transition on the clock signal changes the output of the flipflop to the value of the data input(s). This value is maintained until the next low-to-high transition of the clock, or until the flipflop is reset [30].

A diagram of the devices and connections is shown in Figure 38 and can help visualize the data subsystem architecture.

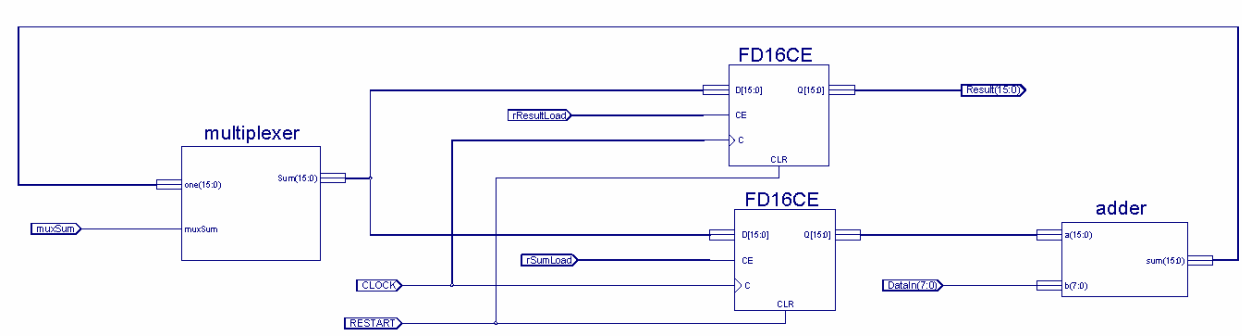


Figure 38. Data Subsystem Details.

Figure 39 shows the *Data* subsystem module (see Appendix B for complete internal schematics and VHDL code). The *Data* subsystem module receives three *Control* subsystem module inputs (i.e., *muxSum*, *rSumLoad*, and *rResultLoad*) and the *Data* input, then outputting the *Result.* Note the *CLOCK* and *RESTART* inputs; these tie the Data module's registers to the clock used for the whole system and the system's external reset signal.

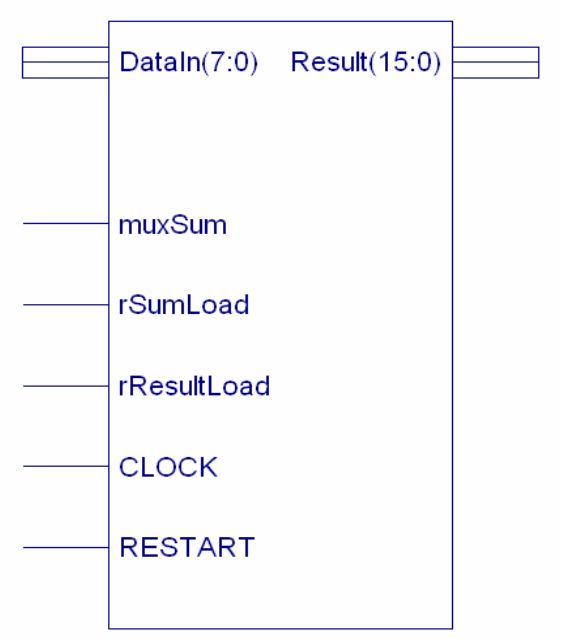


Figure 39. Data Module.

A partial Test-Bench waveform shown in Figure 40 (see Appendix B for a complete Test-Bench waveform) demonstrates the module's ability to store and sum appropriately.

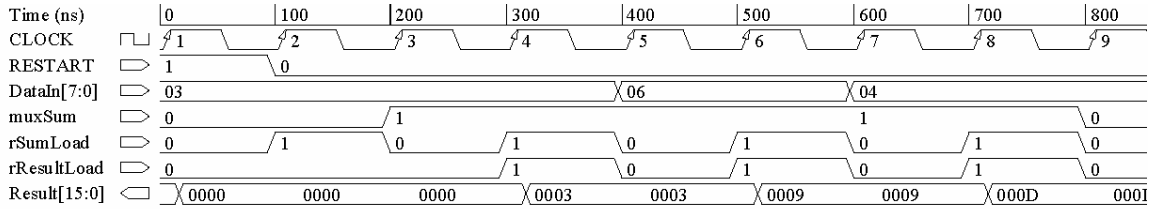


Figure 40. Data Module Test-Bench Waveform.

6. Develop the System Module

As mentioned earlier, the *System* module serves to connect the *Data* and *Control* subsystem modules. The *Control* subsystem is concerned only with generating control signals while the *Data* subsystem is concerned only with operations on data. From the general architecture shown in Figure 32, the checksum device is created and shown in Figure 41.

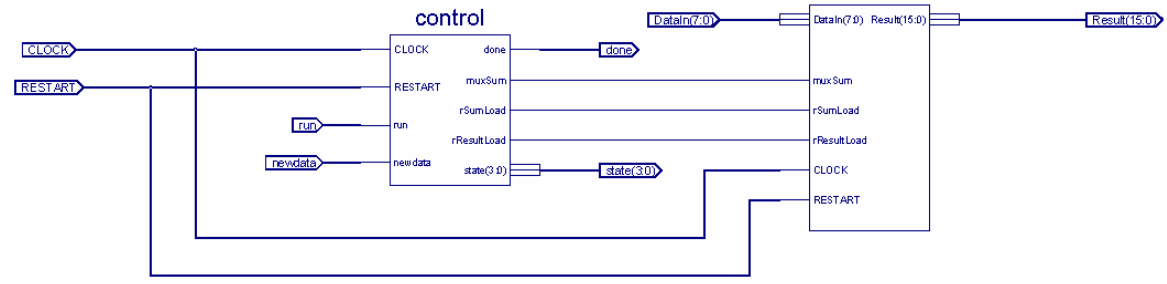


Figure 41. System Module.

A partial Test-Bench waveform shown in Figure 42 (see Appendix B for a complete Test-Bench waveform) demonstrates the module's ability to sum a consecutive series on data inputs.

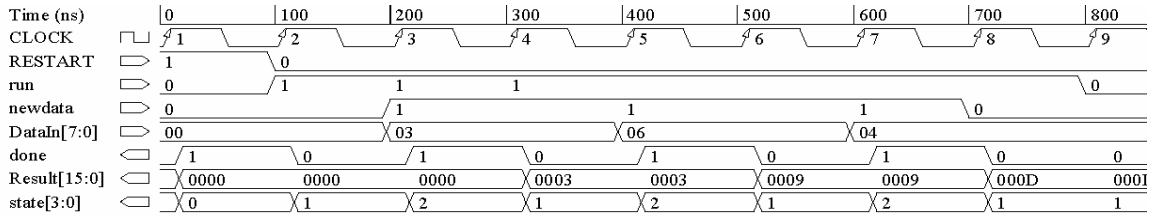


Figure 42. System Module Test-Bench Waveform.

7. Develop the Top Level Module

The *Top-Level* module serves to connect the *System* module with a *Comparator* module. The *System* module, shown in Figure 43, generates a checksum signature while the *Comparator* module compares the result to the stored expected result. The *Checksum* module shown in Figure 43 is a general design. For a design specific to either the EPROM/PROM or Flash Memory data, the *correctResult* bus is hardwired to be a stored signature specific to the configuration(s) stored in the device being tested. This signature is then compared to the checksum result from the System module.

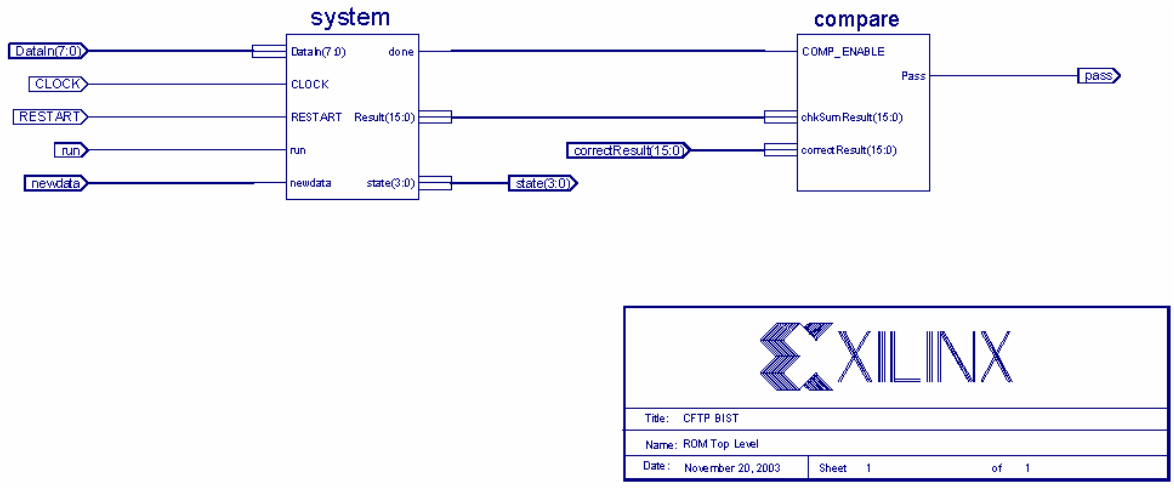


Figure 43. Checksum Module.

A partial Test-Bench waveform shown in Figure 44 (see Appendix B for a complete Test-Bench waveform) demonstrates the module's ability to execute a test on a set of data whose signature should be 00D.

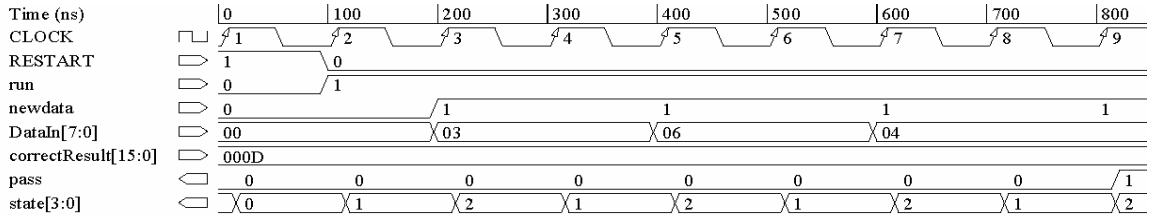


Figure 44. Checksum Module Test-Bench Waveform.

8. Testing the Test

Testing the ROM BIST's ability to detect errors is done by verifying the resulting sum it produces. Two methods to accomplish this are either to feed in a configuration with an erroneous bit or to compare the resulting sum with an incorrect hardwired signature.

The checksum signature output from the *System* module is a 16-bit value created by summing 8-bit values. While summing consecutive 8-bit values of configuration data, eventually the resulting sum will overflow the 16-bit *Result* bus. Once the overflow occurs, technically the signature is no longer unique; however, the probability that an error would cause an overflow and subsequently the correct signature is extremely low. If the CFTP is operating in an environment of high radiation exposure and sufficient concern exists that a duplicate signature will occur, the 16-bit *Result* bus can be expanded to an X-bit bus by modifying the *Data Subsystem* module. The multiplexer buses will need to be lengthened and the 16-bit zero changed to an X-bit zero. The 16-bit registers will need to be replaced with X-bit registers. Finally, the 16-bit buses of the *Adder* module will need to be expanded and the 8-bit data input padded with zeros to make an X-bit input for summing.

9. Conclusions and ROM BIST Implementation

This section has provided a detailed description of the ROM BIST design and how it addresses the hardware self-test for CFTP's configuration-storage components. In

the case of the ISP EPROM or the Flash Memory, where the stored data can be changed, anytime the data is modified by uploading and storing a new configuration(s), a new EPROM or Flash Memory test configuration with the new signature must also be stored. The OTP PROM cannot change its stored data, so the PROM test configuration and signature will never change.

The next section focuses on the method to make configurations to detect internal and external FPGA faults.

D. FIELD-PROGRAMMABLE GATE ARRAY TESTING

Once the prototype CFTP is ready, assurance that each Field-Programmable Gate Array (FPGA) device is wired correctly is needed. Additionally, during initial operations in space, confirmation that the two FPGA devices are working properly is also needed, as the launch environment or radiation effects may have altered their operation. It may also be desired to test FPGAs each time the system is powered-on or reset. As shown in Figure 45, FPGAs are made up of an array of configurable logic blocks (CLBs) surrounded by configurable input/output blocks (IOBs) [21, 26].

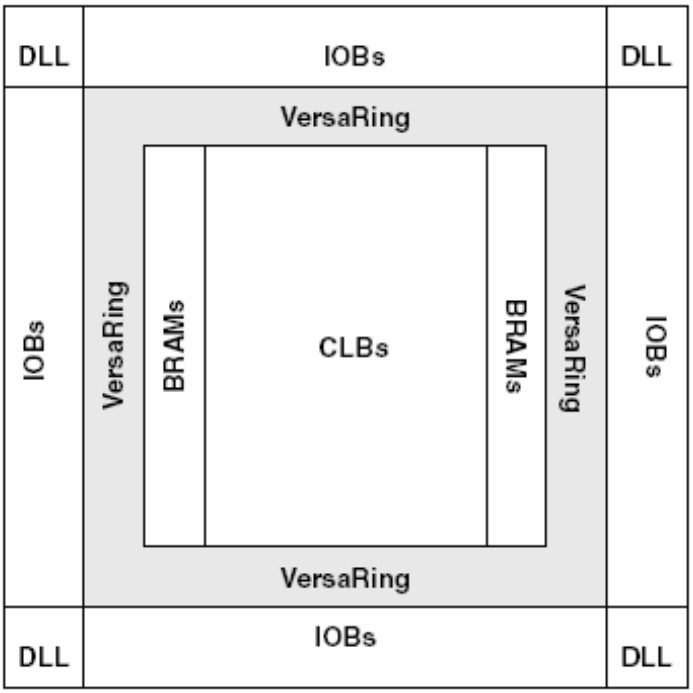


Figure 45. FPGA Architecture Overview. (From Ref. [21].)

This array of CLBs is interconnected by a programmable routing network, or general routing matrix (GRM) [21, 26]. The GRM is an array of switches which determine the routing of wire segments at the intersections of the rows and columns of the array of CLBs. Each CLB is made up of logic cells (LCs) which are the core elements used in constructing the representative logic [21]. IOBs provide the interface between off-chip signals and the CLBs [21]. The CLBs and their interconnect are susceptible to a range of events, such as SEUs or SELs, which may cause damage or faults to occur. In CLBs, this damage can result in changes to the circuit that the LCs were meant to represent. With interconnects, this damage can result in faulty connections between CLBs affecting the operation of the configured circuit. This section addresses the damage or faults that may occur in CLBs or interconnect with two tests: the CLB test of the LCs and the interconnect test of the wires surrounding the CLBs.

1. Introduction

As mentioned in previous sections, by configuring the test logic only during off-line testing, the reprogrammability of the FPGA can be exploited and testability is achieved without any overhead. This is due to the fact that the test logic disappears once the FPGA is reconfigured. This is all facilitated by means of configuration-storage, the amount available of which determines the number functions the FPGA(s) can operate as. Therefore, as will be discussed later, trade-offs must be balanced between the amount of storage space needed for configurations, and the number of configurations needed to perform the desired system functions. Additionally, in the CFTP architecture the two FPGAs can function independently so they can be tested concurrently and therefore reduce the test run-time.

2. Interfacing with the Test

When designing an FPGA test, it is tempting to follow the assumption that access to the test should be made through the multiple input/output (I/O) pins on the FPGA. First instincts are to use the I/O pins for initiation of the test sequence and obtaining the pass/fail status at the end of the sequence; however, this would not be the best method. This use of I/O pins for signals make the FPGA testing difficult since it is not known a priori which I/O pins will be inputs and which will be outputs. The most logical and

practical interface for FPGA testing access is the Boundary Scan Test Access Port (TAP) interface [26]. Because there exists an IEEE Boundary Scan standard, IEEE 1149.1, most FPGAs, including those chosen for the CFTP, support reconfiguration through the IEEE 1149.1 interface [21, 33, 34]. Therefore, Boundary Scan access can be used for downloading the test configurations, initiating the test sequence, retrieving the subsequent test results and reconfiguring the FPGA upon completion of off-line testing. Using a JTAG interface through the PC104 Bus interface with the satellites, initiation of FPGA tests other that at power-on/reset are signaled via a Boundary Scan TAP.

3. The Test Process

The FPGA test process is a sequence of *test phases* in which each phase consists of the following steps: 1) reconfigure the circuit, 2) initiate the test which includes generating test patterns and analyzing responses to produce a pass/fail indication, and 3) read the test results [26]. In step 1 of each *test phase*, the test controller must configure the FPGAs from their respective configuration-storage devices. In step 2, the test controller initiates the test sequence via the system's external reset signal. Finally, in step 3 the pass/fail results of the test are retrieved and the end of the test sequence is indicated by a done signal.

The underlying principle for both the CLB test and the interconnect test is to configure one group of CLBs as TPGs and another group of CLBs as ORAs. Recall from previous sections and Figure 10, the TPG produces a sequence of patterns for testing the CUT and the ORA compares the output responses from the CUT to produce a pass/fail indication. In the CLB test, an additional group of CLBs is configured as blocks under test (BUTs); while in the interconnect test a group of wire segments and configuration interconnect points (CIPs) are configured as wires under test (WUTs) [26]. By closing or opening CIPs, a configuration can control which wire segments are connected or disconnected between CLBs, thus isolating specific WUTs.

In the Xilinx Virtex FPGAs, each CLB is made up of two LCs that provide multiple modes of operation to the CLBs [21]. As shown in Figure 46, a 4-input function generator, carry logic, and storage element per LC allow each CLB to operate in a Look-Up Table (LUT) mode or RAM mode. Because there are multiple modes in which a CLB can operate, during the CLB test the BUTs require repeated reconfiguration in order to test all modes [26]. Similarly, during the interconnect test different groups of WUTs are configured to test all interconnect resources. Each reconfiguration of the FPGA makes up a *test phase*. A collection of *test phases* make up a *test session*. One *test session* completely tests the BUTs in all their modes of operation or tests for similar types of interconnect faults in WUTs. Multiple *test sessions* are needed to completely test all CLBs and their interconnect.

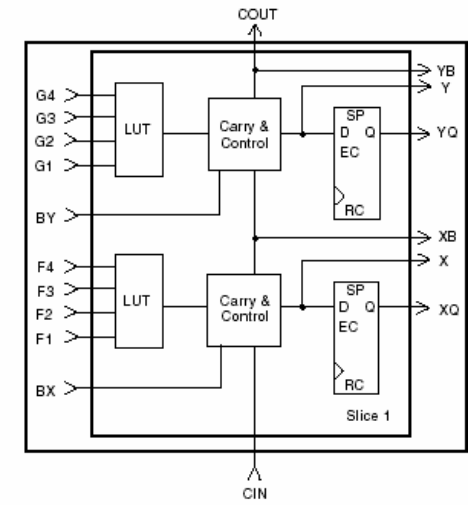
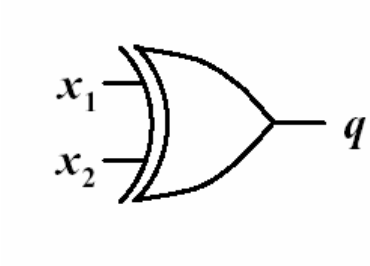


Figure 46. Configuration Logic Block. (After Ref. [21].)

4. The CLB Tests

As mentioned, each CLB can operate in either a RAM mode or a LUT mode. In order to exhaustively test a CLB, each of these modes needs to be tested. Because the RAM mode provides higher fault detection, it is tested first [26]. During the RAM mode, it is required to test that every storage location is working. This is done by writing some set of data to each address and verifying the data by reading it back, similar to the RAM BIST. During the LUT mode, inputs are tested for stuck-at faults. Utilizing the D Algorithm method, faults located at the input to the LUT can be propagated to the output of the LUT [3]. The XOR gate provides good controllability and observability properties both at the input and output. Shown in Table 12, the 2-input XOR gate detects stuck-at-zero (s-a-0) and stuck-at-one (s-a-1) faults on both lines of the input, for every input pattern. So for any test pattern, if a fault occurs on either input it is identified.



	x1 s-a-0	x1s-a-1	x2 s-a-0	x2 s-a-1	q s-a-0	q s-a-1
00		X		X		X
01		X	X		X	
10	X			X	X	
11	X		X			X

Table 12. Fault Table for the XOR Gate. (After Ref. [3].)

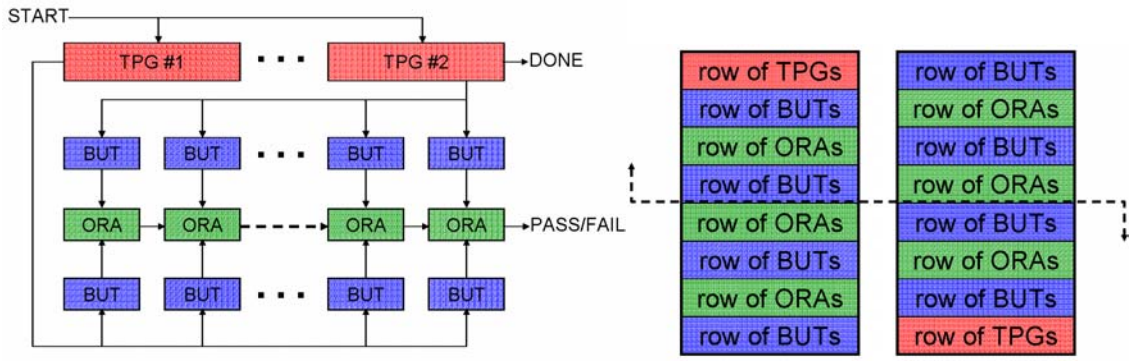
The Propagation D Cube, shown in Table 13, demonstrates how the stuck-at fault occurring at the input is propagated to the output, thus sensitizing the output to each of the input lines. A value D (0 when faulty and 1 when fault-free) sensitizes the output with a value D or D* (1 when faulty and 0 when fault-free). Programming the LUT as a

4-input XOR gate will allow the BUT to propagate a fault to the output. Because the LUTs are 4-input LUTs, the TPG will apply all 2^4, or 16, test vectors to the LUT inputs.

x1	x2	q
D	1	D*
D	0	D
1	D	D*
0	D	D

Table 13. Propagation D Cube for the XOR Gate. (After Ref. [3].)

Figure 47(a) is an example of a CLB *test session* configuration and Figures 47(b) and (c) outline the physical and functional assignments for each row of CLBs in the FPGA. Once the BUTs have been tested, the roles of the CLBs are changed so that in the next *test session* rows that were BUTs become TPGs or ORAs and vice versa. Clearly, if at least half the CLBs are BUTs during each *test session*, only two *test sessions* are needed [26]. This is accomplished by flipping the assignment table in Figure 47(b) about the horizontal axis, making the assignments as shown in Figure 47 (c).



(a) (b) (c)

Figure 47. Basic Architecture for CLB Test. (After Ref. [26].)

If one block architecture, as shown in Figure 47(a), does not cover the FPGA's entire array, multiples of this architecture are needed (see Figure 48). The number of

blocks will depend on the size of the FPGA. This decomposes the testing problem into many identical problems of a fixed size. This approach allows the test to be scalable to FPGAs of any size.

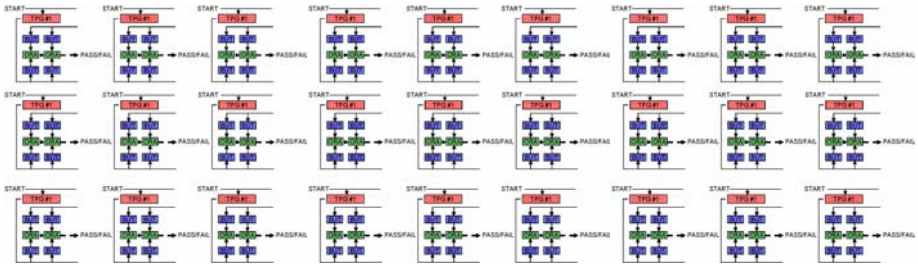
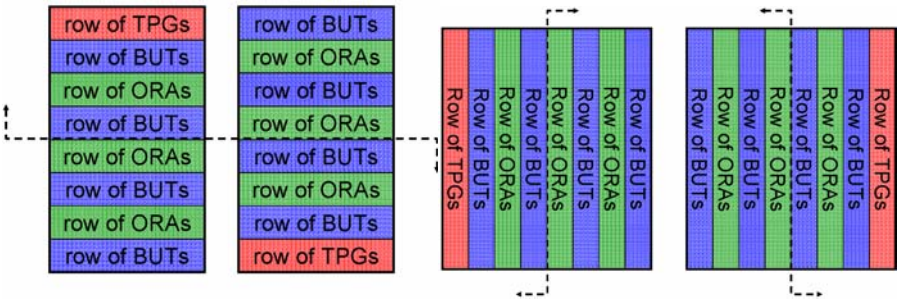


Figure 48. Basic CLB Test Architecture across FPGA array.

During each *test phase* and *test session*, all BUTs are configured to have identical representative logical functions and receive identical test patterns [26]. If each BUT were fault-free, it would produce the same output pattern as their neighboring BUT; therefore, comparator ORAs can be used to compare the outputs of neighboring BUTs and propagate the results through the row of ORAs (as seen in Figure 47(a)). The result of each comparison is stored and then shifted out at the end of the test [26]. By storing the pass/fail result from each row of ORAs, the test can identify the specific row in which a faulty CLB exists. Similarly, by rotating the test configuration 90 degrees into a new series of *test phases*, any column containing a faulty CLB can be identified. Incorporating these two test methods provide a means to uniquely identify a specific faulty CLB within the FPGAs. Figure 49 outlines the physical and functional assignments for each row of CLBs in this complete test method.



(a) Session #1 (b) Session #2 (c) Session #3 (d) Session #4

Figure 49. CLB Test Configurations for FPGAs. (After Ref. [26].)

Assuming two modes of operation, each row assignment requires two configurations or two *test phases*. At a minimum, each row assignment requires two *test sessions* to allow each BUT to become a TPG or ORA and vice versus. To identify a faulty CLB by row and column, an additional two *test sessions* are required. Therefore, the number of tests in this example required to completely test the CLBs is four test sessions of two test phases each or a total eight test phases/configurations. As mentioned in Chapter 2, the configuration-storage for the Processor FPGA holds eight configurations and the Controller FPGA holds either one for the ISP EPROM or four for the OTP PROM. So, one obvious cost incurred when implementing the FPGA test becomes the amount of external memory required to store the multiple test configurations. A single test configuration with total fault coverage would be ideal, and provides a good trade-off in situations where applying a full set of configurations requires too much system-level real estate. However, experiments of such methods have achieved lesser fault coverage with one example of only 56.5% fault coverage of single stuck faults [35]. Research is continuing and new test configurations are being designed with increased fault coverage and reasonable sizes.

5. The Interconnect Test

As in CLB testing, during interconnect testing the FPGA is reconfigured as various independent structures. Wire segments and CIPs are configured to form two groups of wires under test (WUTs) [26]. As with the BUTs in the CLB test, the WUTs will receive identical test patterns. ORAs will compare WUTs and produce a pass/fail indication. Figure 50 is an example of an interconnect test configuration. Similar to the CLB test, there may need to be many of these test blocks operating concurrently to cover the entire FPGA array. Several wire segments connected by closed CIPs make up the WUTs [26]. In Figure 50 there are two groups of WUTs shown. The first connects the wire segments S1, S2, S3, S4, S5, and S6. The second connects the wire segments S7, S8, S9, and S10. WUT may pass through CLBs that are configured as an identity function and thus the inputs are passed directly to the outputs (as shown in Figure 50 by the dashed line through the CLB boxes) [26].

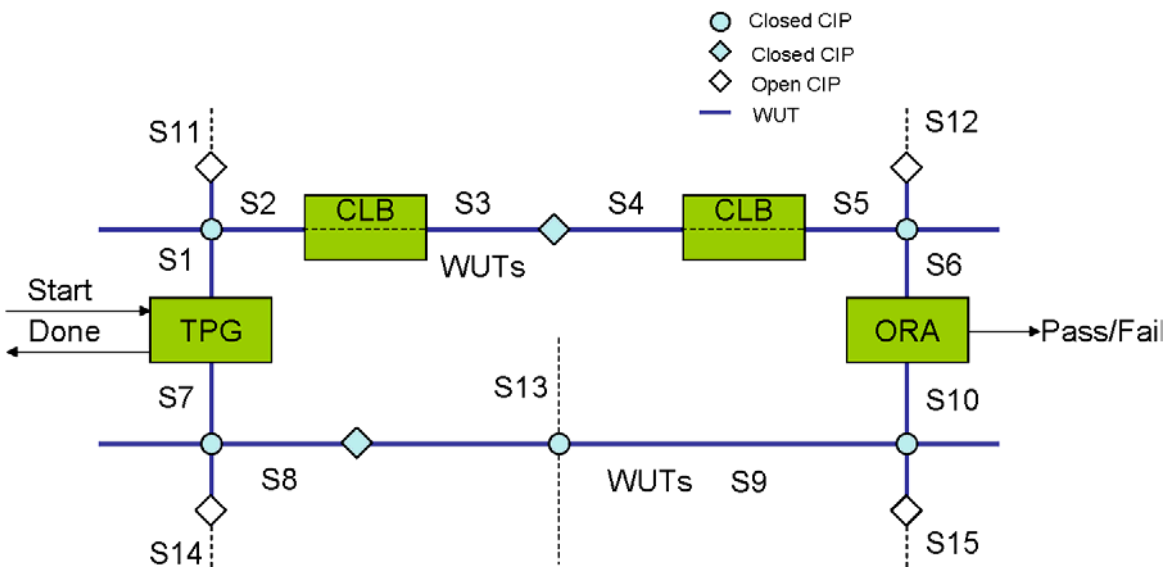


Figure 50. Basic Architecture for Interconnect Test Configuration. (After Ref. [26].)

The interconnect *test session* uses the same *test phases* as mentioned earlier: 1) reconfigure the circuit, 2) initiate the test, and 3) read the test results. During step 2, test patterns are generated and output responses are analyzed. The WUTs consist of *n* wire segments. Each wire segment must be tested for its ability to transmit both logic 0 and logic 1 which takes 2^n test vectors to complete an exhaustive test pattern. The TPG will apply these patterns to one end of the WUTs and the ORA will verify the patterns received at the other end of the WUTs. The open CIPs that isolate the WUTs from the rest of the interconnect must be tested for stuck-closed faults. To achieve this, the TPG controls any wire that may become shorted to some WUT (S11, S12, S13, S14, and S15 in Figure 50), such that when the TPG passes a logic 0 on the WUT it also sets S11, S12, S13, S14, and S15 to logic1 (and vice versus from logic 1 to logic 0) at least once during the interconnect test phase [26].

A potential problem occurs during step 2, when the *test phase* is analyzing the pass/fail indications from the ORAs during both the CLB test and the interconnect test. If two sets of BUTs/WUTs possess equivalent faults when compared by the ORA, they will escape detection [26]. To resolve this issue, every group of BUTs/WUTs must be com-

pared with multiple respective BUTs/WUTs. This will decrease the chances that a fault will go undetected, but unfortunately increase the number of *test sessions* and configurations.

6. Conclusions and FPGA BIST Implementation

In conclusion, it is apparent that the main cost in FPGA testing is the number of configurations required to be stored onboard the CFTP in order to completely test the FPGA devices. As mentioned earlier, the alternative is to use a single, total fault coverage test that will detect all the faults in one test sequence. While single, high fault coverage tests do not ensure the FPGA is completely fault-free, they do provide some means of reasonable assurance that the devices are operational. The amount of fault coverage and the degree of assurance a specific test can provide for FPGAs in specific system-level design is dependant on each application.

The limited configuration storage available in the current CFTP design brings this real estate question to reality. A decision needs to be made on what amount of fault coverage for the CFTP is adequate. Two key points can help focus this debate. First, the FPGA BIST for the CFTP is not intended as a means to provide fault tolerance to any of CFTP's configurations. It is assumed that each configuration designer will perform his/her own reliability analysis and provide for the needed level of fault tolerance. This helps relax any requirements on the BIST design to mitigate SEUs that may cause faults to occur in CLBs or their interconnect. Therefore, the use of the FPGA BIST can be isolated to monitoring the system health and reliability and not fault tolerance.

Second, past experiences with creating designs for implementation into the chosen FPGAs have shown that they have more than enough space to represent the desired circuit. One example of a TMR processor design showed that only 17% of the FPGA was used, even with three microprocessors in the design [15]. This provides a large amount of area within each FPGA for remapping the design around faulty CLBs or even a complete row containing a faulty CLB. This allows the BIST design to perform adequately without the additional configurations needed to narrow down the faulty CLB by column. Therefore, this reduces the required configurations to two test sessions of two

test phases each, or four configurations for the CLB test and two test sessions of one test phase each, or two configurations for the interconnect test. These six configurations are the minimum number of configurations needed to completely test each BUT and WUT for faults.

The only FPGA configuration-storage device that can store all of these test configurations is the Flash Memory. Fortunately, both FPGAs have access to this device without having to go through the other FPGA. So, both FPGAs can run the BIST independently and concurrently. During on-orbit operations, if the minimal FPGA BIST of six configurations is stored in Flash Memory, that leaves two remaining configurations in the Processor FPGA and three in the Controller FPGA (one of the four configurations for the Controller FPGA is dedicated to the Controller's default configuration). As mentioned in Chapter 2, one of the CFTP objectives is to demonstrate fault tolerance with a TMR processor configuration; therefore this will take up one of the remaining two configurations in the Processor FPGA. With another four configurations available for the RAM BIST, ROM BIST, and other potential configurations to assist the CFTP in demonstrating its reprogrammability and reconfigurability objectives, the minimal FPGA BIST is definitely a viable option with the available configuration-storage real estate. During development, when the configuration-storage devices can be easily and quickly updated, larger configurations can be used for the FPGA BIST.

IV. CONCLUSIONS AND FOLLOW-ON RESEARCH

A. OVERVIEW

This purpose of this thesis was to design and develop a series of Built-In Self-Tests (BISTs) for use with the Configurable Fault Tolerant Processor (CFTP) in space applications. The final stages of integration are still required once the development, qualification and flight boards are ready. In concert with the initial objectives of this project, the CFTP BIST has been designed as a self-contained, autonomous, self-testing circuit.

B. CONCLUSIONS

This thesis has described a BIST approach for the CFTP's System-Memory, configuration-storage, and Field-Programmable Gate Arrays that provides a means to monitor the health of the system and furnish reliability through BISTs. The BISTs operate on each component of the CFTP system: RAM BIST (System-Memory), ROM BIST (configuration-storage memory), and FPGA BIST (both FPGAs). The test sequence for these BISTs follows the order: FPGA BIST, ROM BIST, and then RAM BIST. This will allow for assurance in the FPGAs prior to loading them with the ROM BIST. Finally, once the PROM/EPROM and Flash Memory functionalities are confirmed, the RAM BIST configuration can be loaded. Together, this test suite forms a set of hardware diagnostics. If one or more of the diagnostics fail, action can be taken to diagnose the problem and repair or circumvent the faulty hardware.

The basic BIST architecture introduced has proven to be very adaptive, as it has been applied in every BIST design easily and without performance penalties. Additionally, the applicability of FPGAs in BIST implementations has proven to have a very important role in reducing on-chip testing hardware. The FPGA approach allows the test hardware to be removed from the device once the test is complete. However, FPGAs do incur other test costs, such as increased test generation and test application time. Instead

of a single configuration for fault detection, FPGAs require multiple configurations to test an assortment of CLB modes and programmable interconnect, with the attendant configuration storage cost.

Fortunately, the RAM BIST is accomplished in a single configuration and efficiently tests whether each storage location in the System-Memory is working correctly. Similarly, the ROM BIST is a single configuration test. An important consideration to remember when uploading new configurations to the CFTP is that this changes the signature of the ROM device and therefore will not match the signature hardwired into the ROM BIST checksum device. To allow for continuous autonomous operation of the ROM BIST, a new ROM BIST configuration should also be uploaded with the new correct signature. This should not be an issue when uploading configurations from the Ground Station to the satellite, as it is just as easy to send all the configurations as it is to send one[6]. An issue will arise when the CFTP is saving a configuration from itself into the Flash Memory device, possibly for download to the Ground Station.

Additionally, because the RAM BIST and ROM BIST operate on different components of the CFTP and have the same number of required configurations, their designs can be implemented in one RAM/ROM BIST that simultaneously conducts both tests. In the Xilinx ISE software used in creating CFTP's BISTs, this is easily done. By simply creating a module that represents each of the two overall BISTs, a new test can be created that contains these two modules. Some integration may be needed to deconflict signal names or other syntax that may be illegally shared between the modules. Finally, To further reduce the number of required configurations, the Processor and Controller FPGAs should be tested concurrently.

An underlying theme throughout the design process of the CFTP project was to maintain a very small footprint. This design constraint has led to a very efficient reprogrammable/reconfigurable system; however, it has created an obstacle in its ability to

[6] While the satellites that CFTP is currently manifested on have a modest data rate between ground station(s) and satellites (e.g. MidSTAR-1 can provide up to 9600 bps), files uploaded to the satellite will be compressed before transmitting and buffered upon receipt.

maintain system-reliability. Future CFTP designs should take this into consideration and provide for a means to store a larger number of configurations with either more configuration-storage devices or ones that can hold a much larger number of configurations. At a minimum, the CFTP should be able to store all 10 FPGA BIST configurations and the RAM/ROM BIST configuration and still have room for other configurations such as the TMR processor design.

C. FOLLOW-ON RESEARCH

Currently, the CFTP is manifested for launch into LEO orbit on two satellites in 2006 and efforts are in motion to acquire additional manifests into higher orbits. In order to accommodate these flights, there are many areas for follow on research that *must* be accomplished. First, the BISTs designed in this thesis must be integrated into the CFTP development board, qualification board, and flight boards. Assuming that these boards' designs prove valid, the qualification board must be qualified for space. This verification process should evaluate the suitability of the designs in numerous space environments (e.g., high vacuum, extreme temperatures, solar and particle radiation) and launch conditions (e.g., shock load, static load, and various frequency driven vibrations). While these environments will not necessarily duplicate the space environment or launch conditions, they merely need to approach it sufficiently so that any unit that passes the tests will survive the launch and operate successfully in its designed space environment. Once a qualified design has been achieved, flight boards must be built, tested, and then integrated into the host satellites.

The configuration for the Controller FPGA needs to be designed. This should contain at a minimum the configuration control, command and status registers, and bus interface logic. This is an essential component to the CFTP architecture and will have to be integrated and routed to the appropriate hardwired pins.

Other than the BIST designs in this thesis, the only other configuration written for the CFTP is a Triple Modular Redundant (TMR) microprocessor, KDLX. However, to date no programs have been written for it. Although extensive research has been done to perfect its interrupt handling capabilities in mitigating SEU induced errors and converting

its memory interface architecture to match that of the CFTP's, actual instantiation of the TMR design in an FPGA has not been attempted.

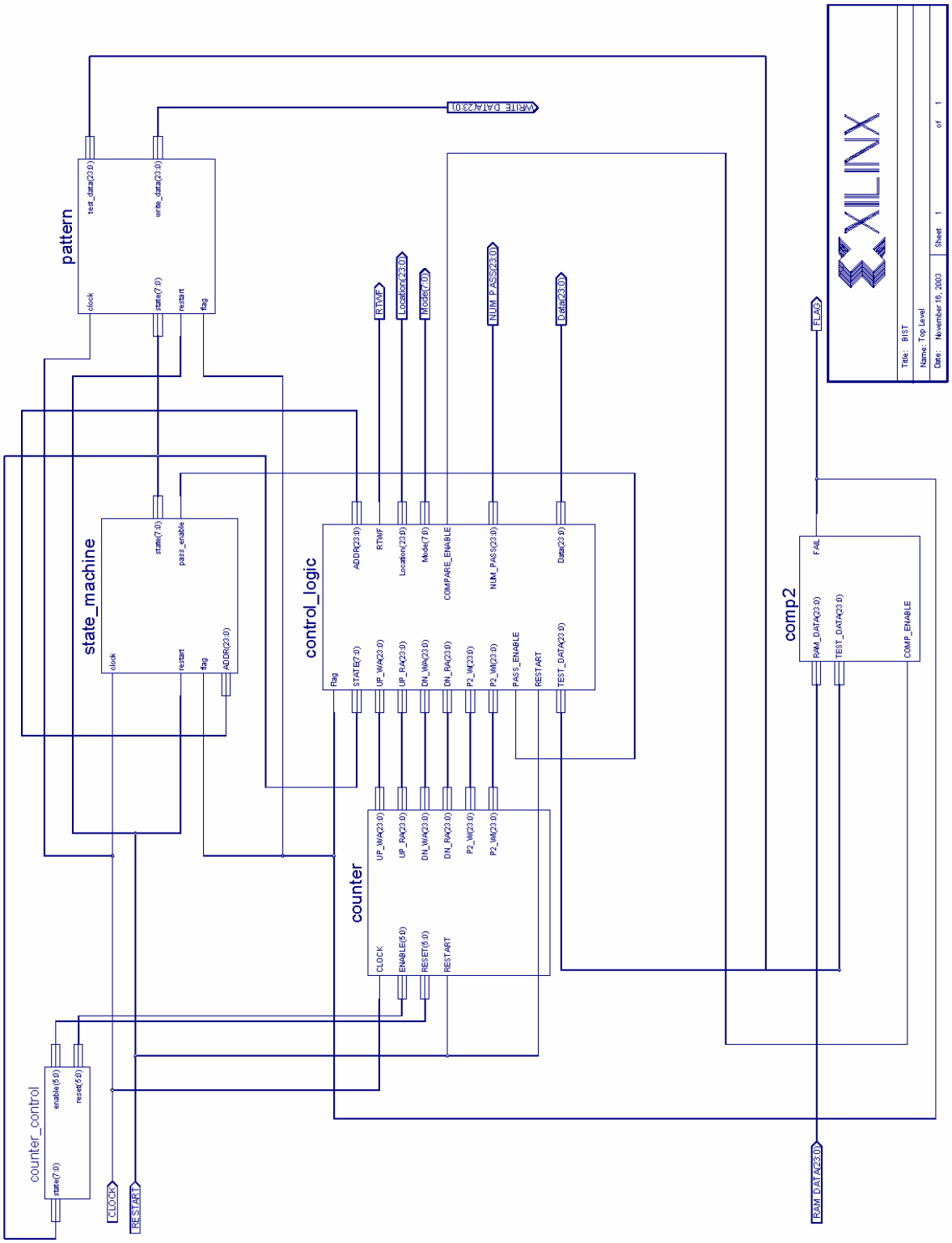
Finally, the fine details in the integration of the CFTP into the host satellites needs to be accomplished. While protocols, connectors and components have been agreed upon, the details of what commands need to be transmitted and received between the CFTP and the satellite Command and Data Handler have not been.

APPENDIX A: COMPLETE SCHEMATICS, VHDL CODES AND TEST-BENCH WAVEFORMS FOR SDRAM TEST

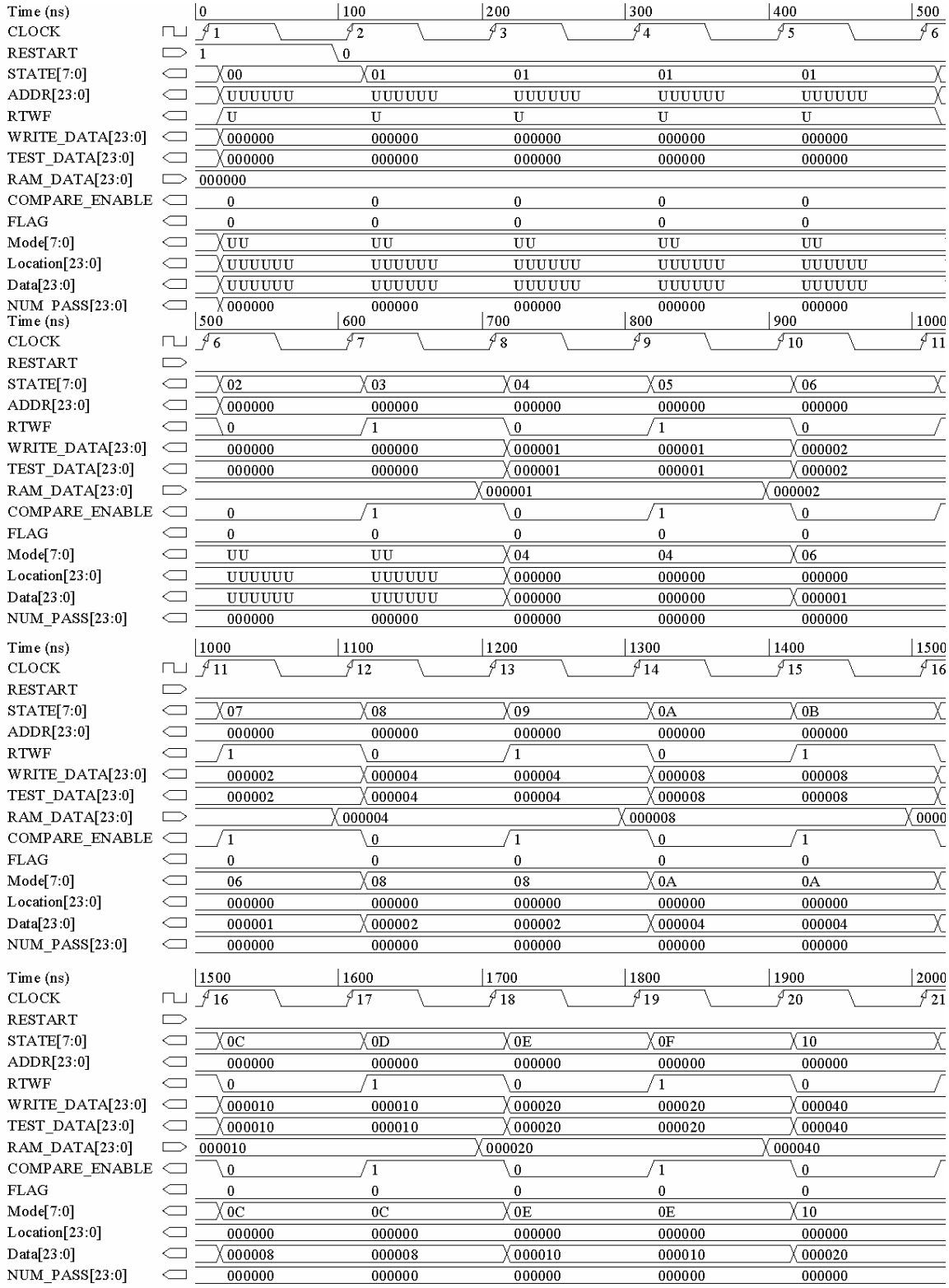
Appendix A contains the schematic diagrams, VHDL code, and Test-Bench waveforms for the complete SDRAM test design. The appendix is organized by module, so each section contains a module schematic/VHDL code and a Test-Bench waveform.

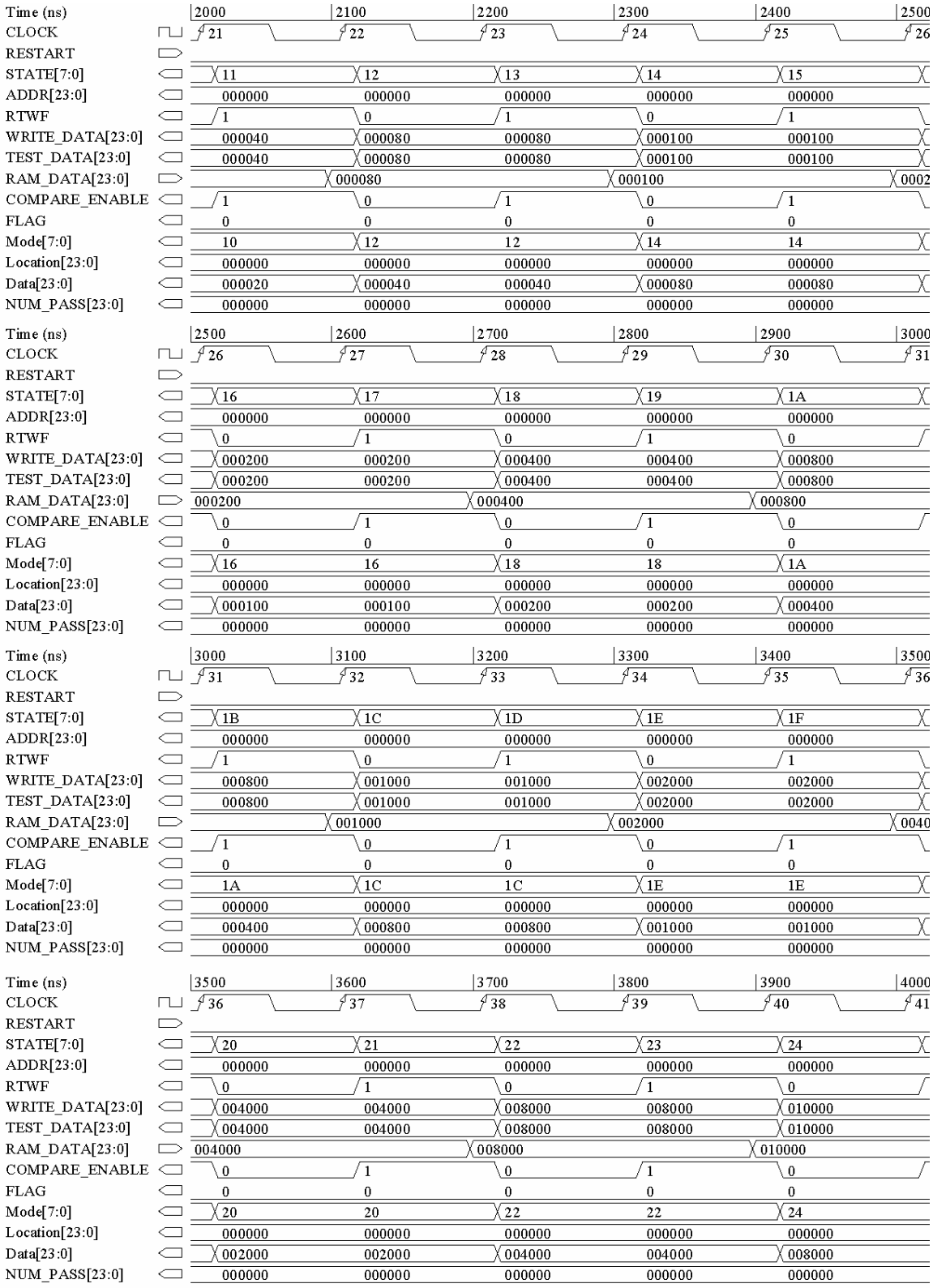
A. COMPLETE DESIGN

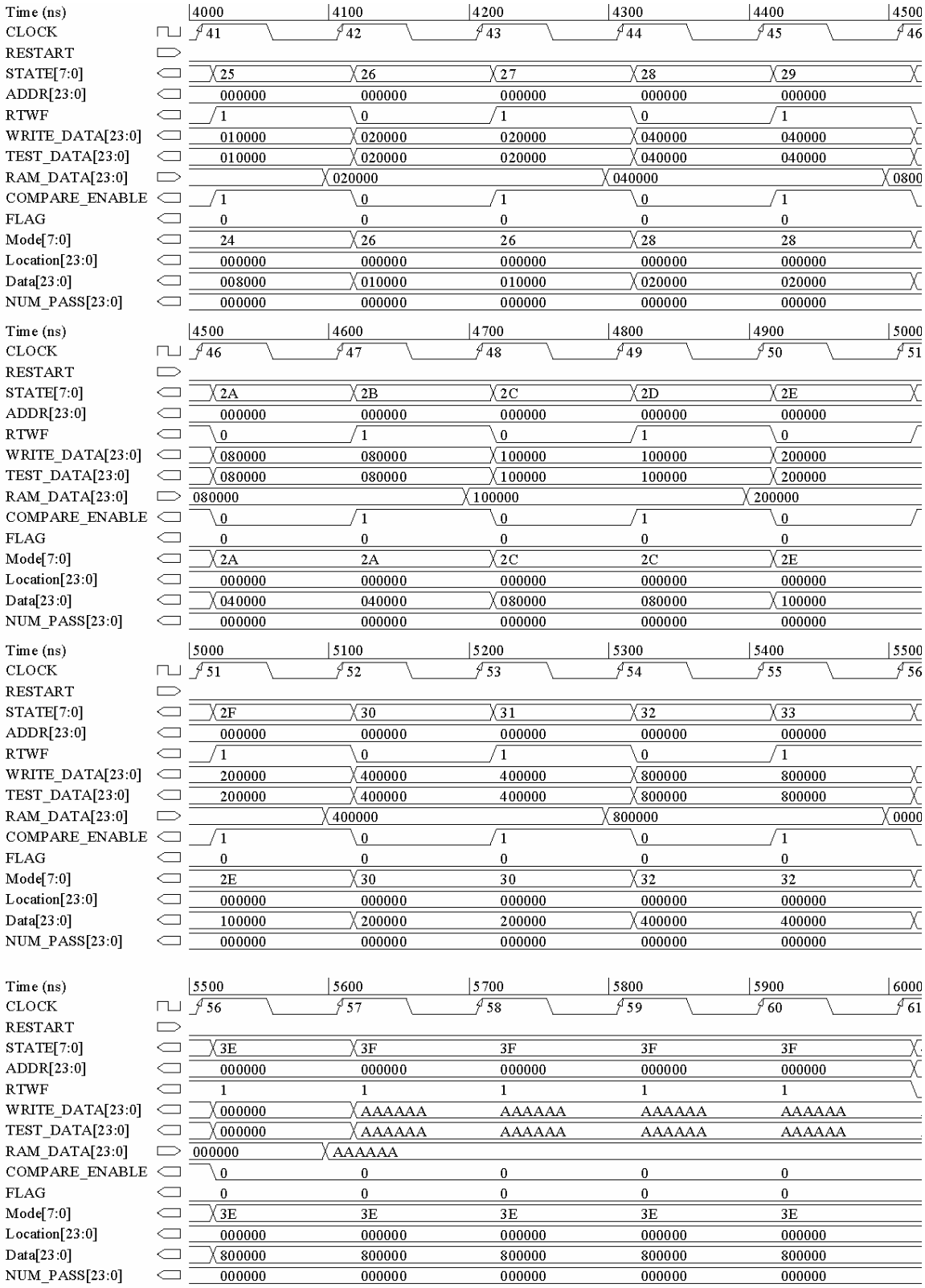
1. Schematic Diagram

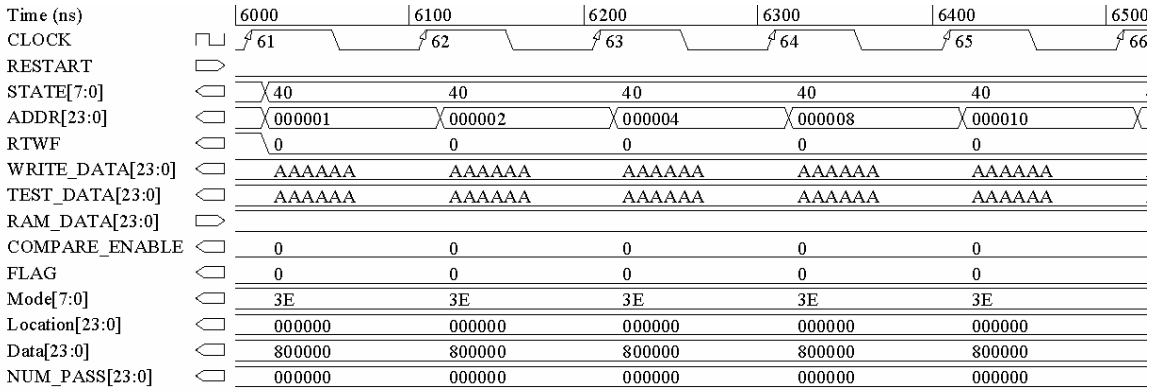


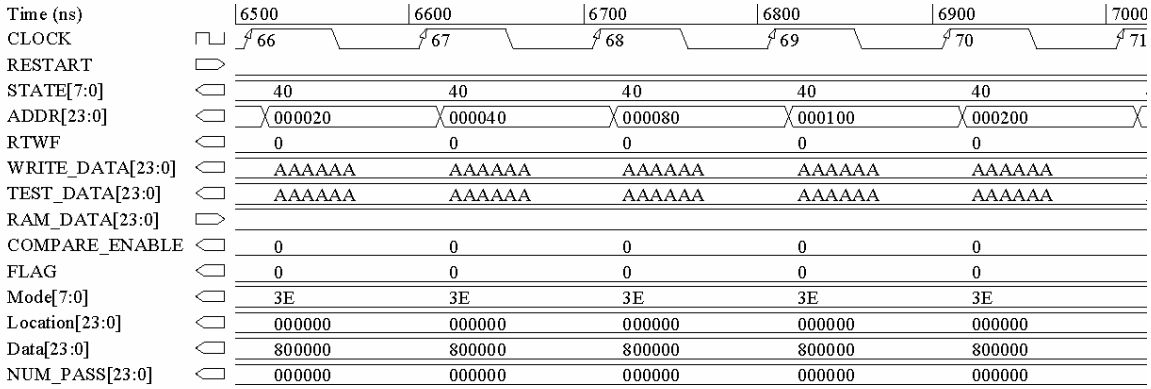
2. Test-Bench Waveform

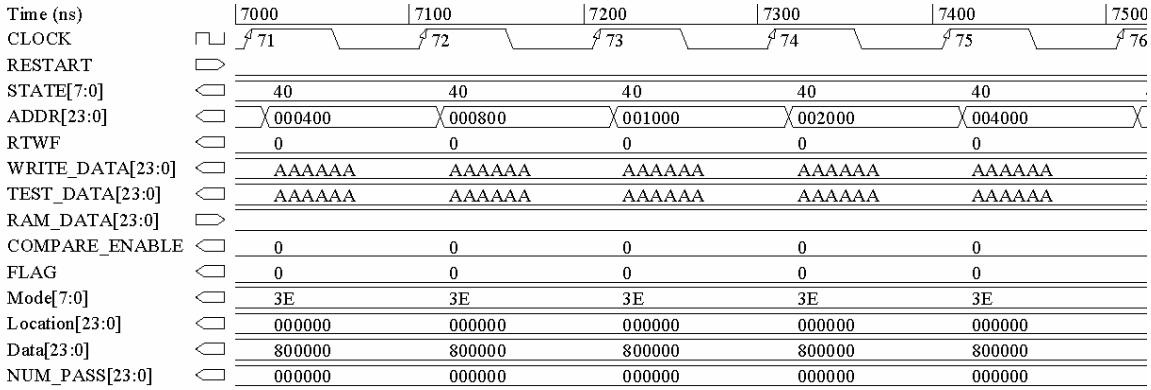


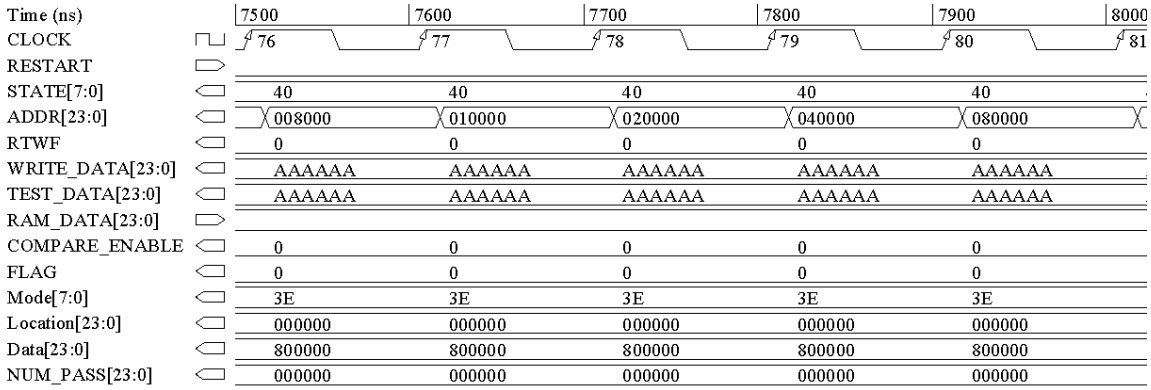

Time (ns) 2000 2100 2200 2300 2400 2500
CLOCK 21 22 23 24 25 26
RESTART
STATE[7:0] 11 12 13 14 15
ADDR[23:0] 000000 000000 000000 000000 000000
RTWF 1 0 1 0 1
WRITE_DATA[23:0] 000040 000080 000080 000100 000100
TEST_DATA[23:0] 000040 000080 000080 000100 000100
RAM_DATA[23:0] 000080 000100 0002
COMPARE_ENABLE 1 0 1 0 1
FLAG 0 0 0 0 0
Mode[7:0] 10 12 12 14 14
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 000020 000040 000040 000080 000080
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 2500 2600 2700 2800 2900 3000
CLOCK 26 27 28 29 30 31
RESTART
STATE[7:0] 16 17 18 19 1A
ADDR[23:0] 000000 000000 000000 000000 000000
RTWF 0 1 0 1 0
WRITE_DATA[23:0] 000200 000200 000400 000400 000800
TEST_DATA[23:0] 000200 000200 000400 000400 000800
RAM_DATA[23:0] 000200 000400 000800
COMPARE_ENABLE 0 1 0 1 0
FLAG 0 0 0 0 0
Mode[7:0] 16 16 18 18 1A
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 000100 000100 000200 000200 000400
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 3000 3100 3200 3300 3400 3500
CLOCK 31 32 33 34 35 36
RESTART
STATE[7:0] 1B 1C 1D 1E 1F
ADDR[23:0] 000000 000000 000000 000000 000000
RTWF 1 0 1 0 1
WRITE_DATA[23:0] 000800 001000 001000 002000 002000
TEST_DATA[23:0] 000800 001000 001000 002000 002000
RAM_DATA[23:0] 001000 002000 0040
COMPARE_ENABLE 1 0 1 0 1
FLAG 0 0 0 0 0
Mode[7:0] 1A 1C 1C 1E 1E
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 000400 000800 000800 001000 001000
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 3500 3600 3700 3800 3900 4000
CLOCK 36 37 38 39 40 41
RESTART
STATE[7:0] 20 21 22 23 24
ADDR[23:0] 000000 000000 000000 000000 000000
RTWF 0 1 0 1 0
WRITE_DATA[23:0] 004000 004000 008000 008000 010000
TEST_DATA[23:0] 004000 004000 008000 008000 010000
RAM_DATA[23:0] 004000 008000 010000
COMPARE_ENABLE 0 1 0 1 0
FLAG 0 0 0 0 0
Mode[7:0] 20 20 22 22 24
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 002000 002000 004000 004000 008000
NUM_PASS[23:0] 000000 000000 000000 000000 000000

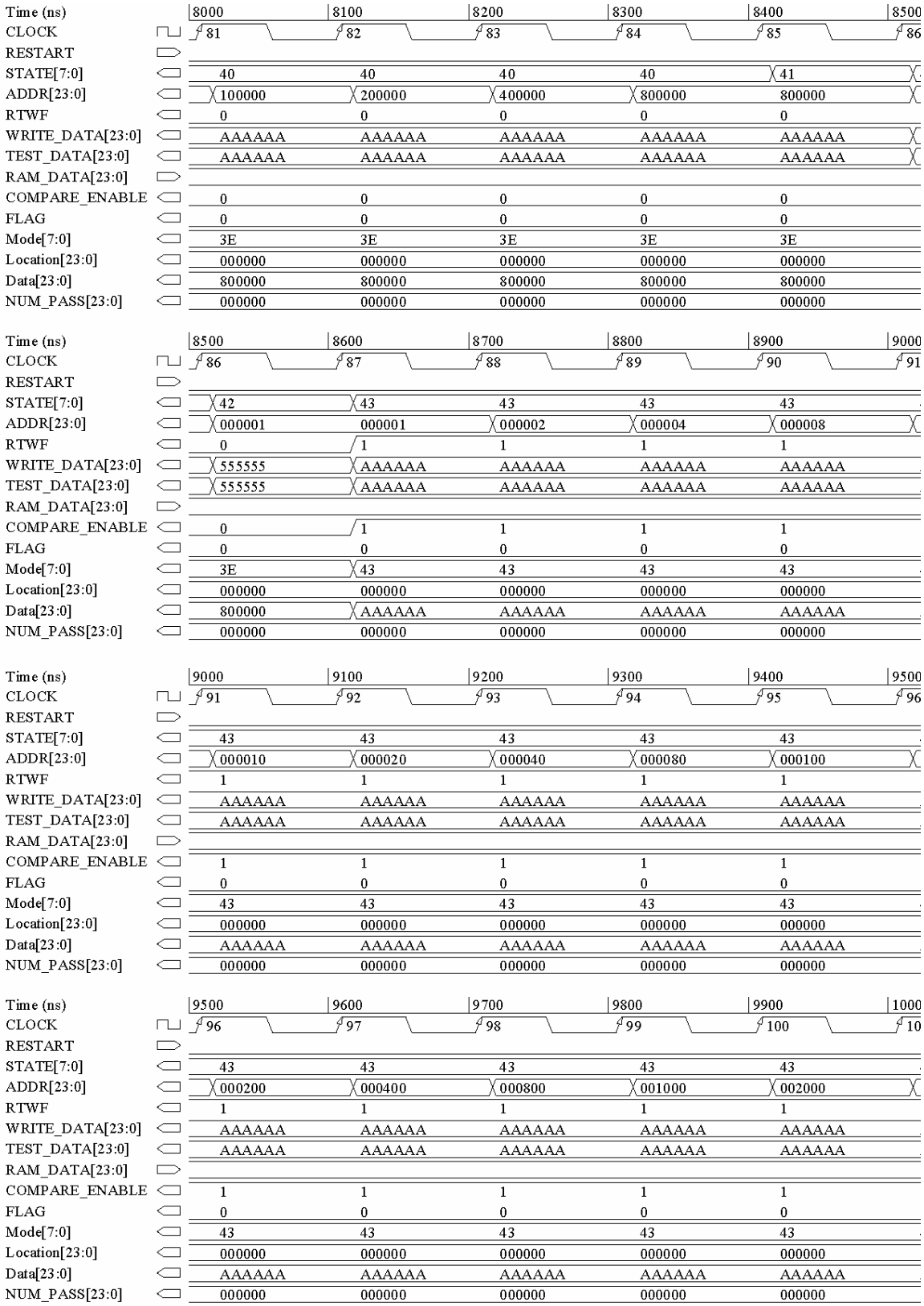

Time (ns) 4000 4100 4200 4300 4400 4500
CLOCK 41 42 43 44 45 46
RESTART
STATE[7:0] 25 26 27 28 29
ADDR[23:0] 000000 000000 000000 000000 000000
RTWF 1 0 1 0 1
WRITE_DATA[23:0] 010000 020000 020000 040000 040000
TEST_DATA[23:0] 010000 020000 020000 040000 040000
RAM_DATA[23:0] 020000 040000 0800
COMPARE_ENABLE 1 0 1 0 1
FLAG 0 0 0 0 0
Mode[7:0] 24 26 26 28 28
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 008000 010000 010000 020000 020000
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 4500 4600 4700 4800 4900 5000
CLOCK 46 47 48 49 50 51
RESTART
STATE[7:0] 2A 2B 2C 2D 2E
ADDR[23:0] 000000 000000 000000 000000 000000
RTWF 0 1 0 1 0
WRITE_DATA[23:0] 080000 080000 100000 100000 200000
TEST_DATA[23:0] 080000 080000 100000 100000 200000
RAM_DATA[23:0] 080000 100000 200000
COMPARE_ENABLE 0 1 0 1 0
FLAG 0 0 0 0 0
Mode[7:0] 2A 2A 2C 2C 2E
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 040000 040000 080000 080000 100000
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 5000 5100 5200 5300 5400 5500
CLOCK 51 52 53 54 55 56
RESTART
STATE[7:0] 2F 30 31 32 33
ADDR[23:0] 000000 000000 000000 000000 000000
RTWF 1 0 1 0 1
WRITE_DATA[23:0] 200000 400000 400000 800000 800000
TEST_DATA[23:0] 200000 400000 400000 800000 800000
RAM_DATA[23:0] 400000 800000 0000
COMPARE_ENABLE 1 0 1 0 1
FLAG 0 0 0 0 0
Mode[7:0] 2E 30 30 32 32
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 100000 200000 200000 400000 400000
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 5500 5600 5700 5800 5900 6000
CLOCK 56 57 58 59 60 61
RESTART
STATE[7:0] 3E 3F 3F 3F 3F
ADDR[23:0] 000000 000000 000000 000000 000000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] 000000 AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] 000000 AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0] 000000 AAAAAA
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 3E 3E 3E 3E 3E
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 800000 800000 800000 800000 800000
NUM_PASS[23:0] 000000 000000 000000 000000 000000

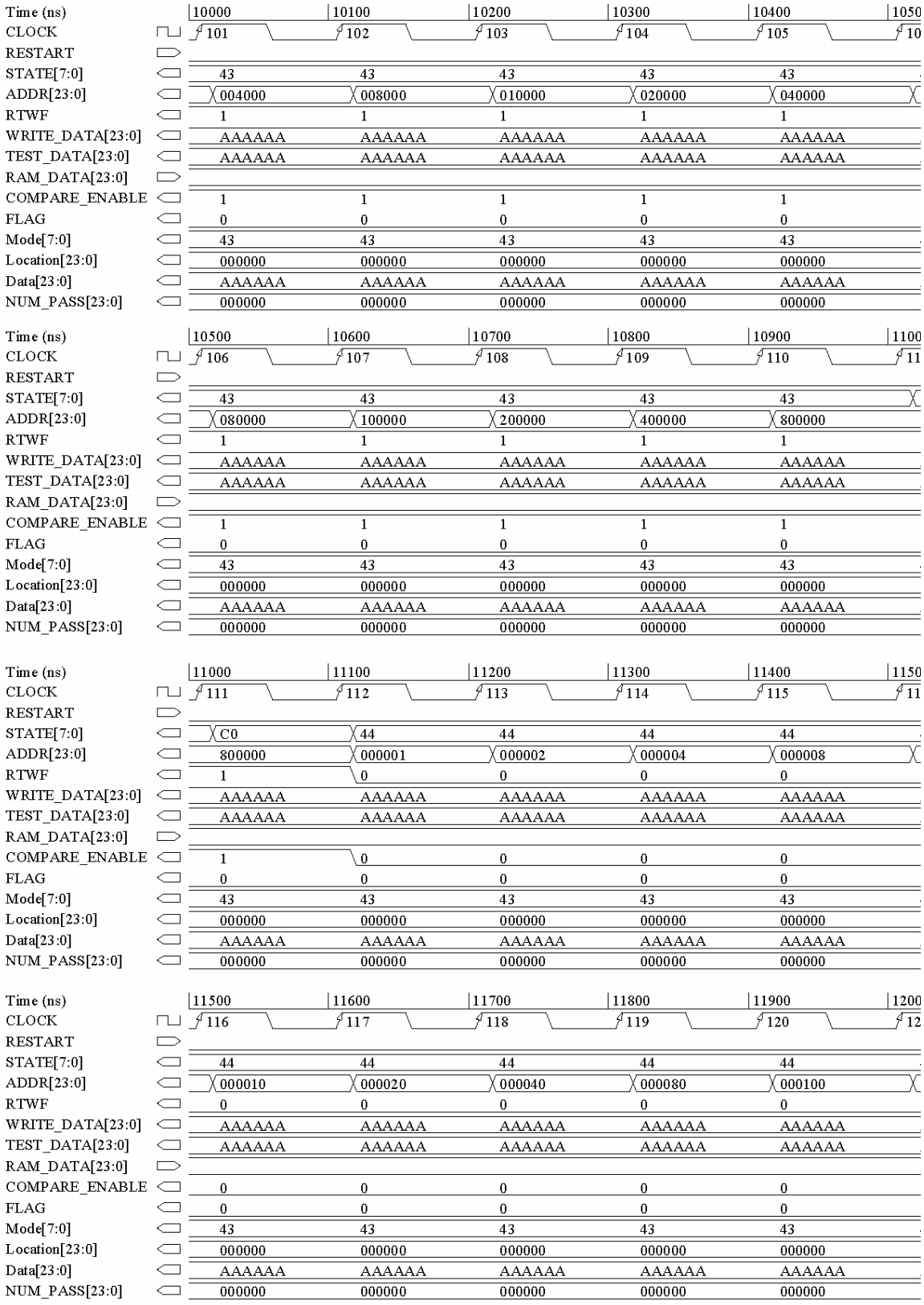

Time (ns) 6000 6100 6200 6300 6400 6500
CLOCK 61 62 63 64 65 66
RESTART
STATE[7:0] 40 40 40 40 40
ADDR[23:0] 000001 000002 000004 000008 000010
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 3E 3E 3E 3E 3E
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 800000 800000 800000 800000 800000
NUM_PASS[23:0] 000000 000000 000000 000000 000000

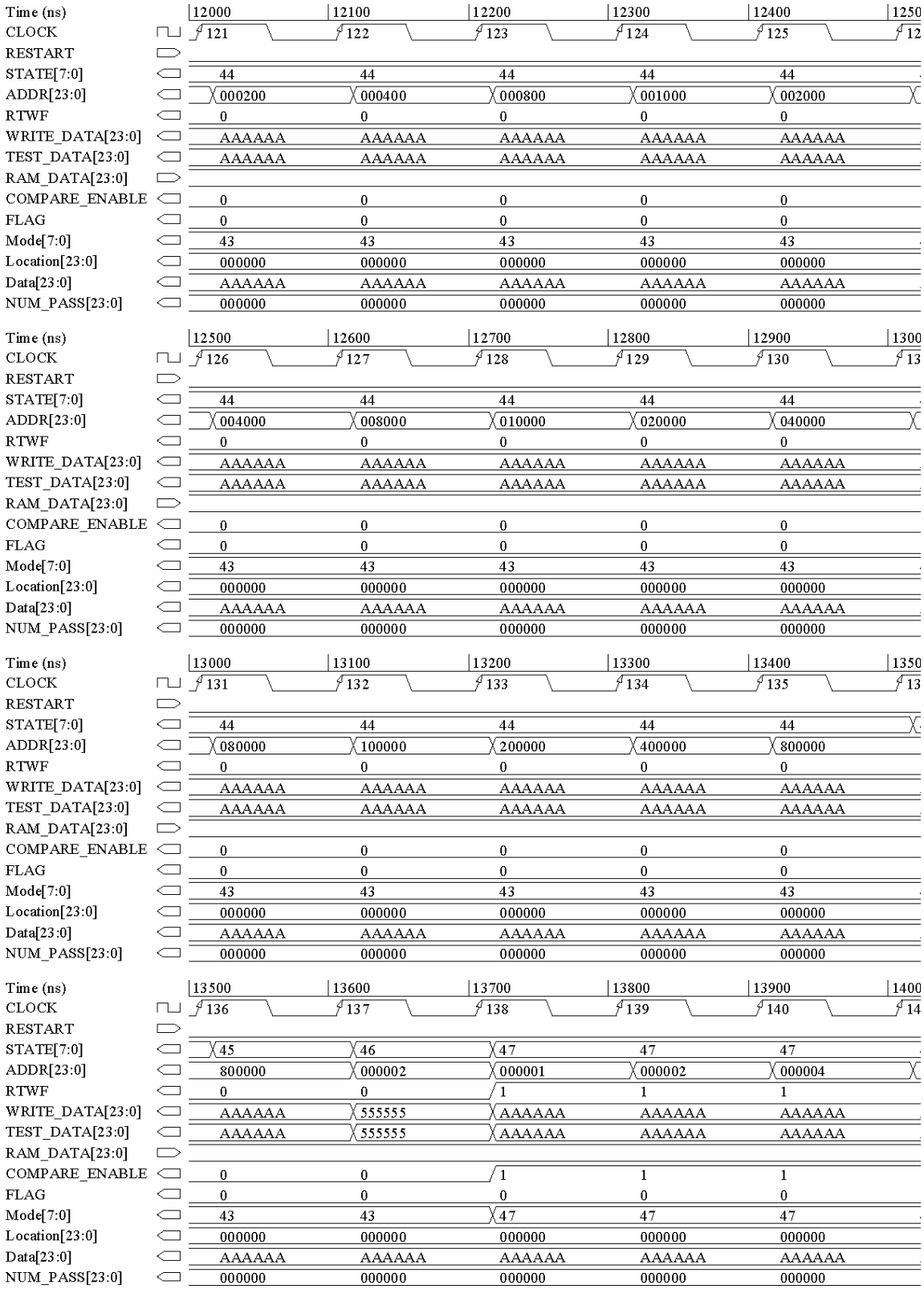

Time (ns) 6500 6600 6700 6800 6900 7000
CLOCK 66 67 68 69 70 71
RESTART
STATE[7:0] 40 40 40 40 40
ADDR[23:0] 000020 000040 000080 000100 000200
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 3E 3E 3E 3E 3E
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 800000 800000 800000 800000 800000
NUM_PASS[23:0] 000000 000000 000000 000000 000000

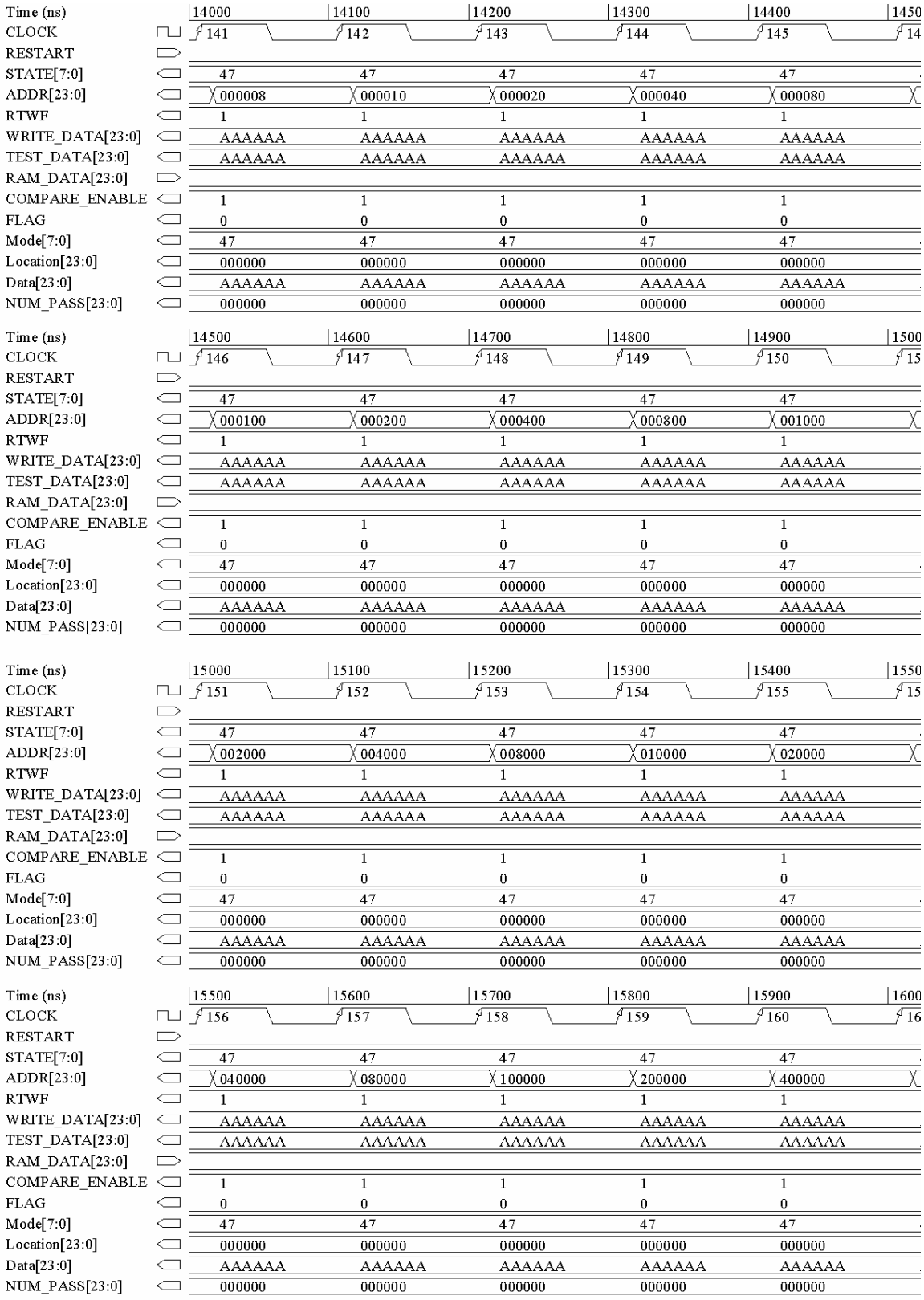

Time (ns) 7000 7100 7200 7300 7400 7500
CLOCK 71 72 73 74 75 76
RESTART
STATE[7:0] 40 40 40 40 40
ADDR[23:0] 000400 000800 001000 002000 004000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 3E 3E 3E 3E 3E
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 800000 800000 800000 800000 800000
NUM_PASS[23:0] 000000 000000 000000 000000 000000

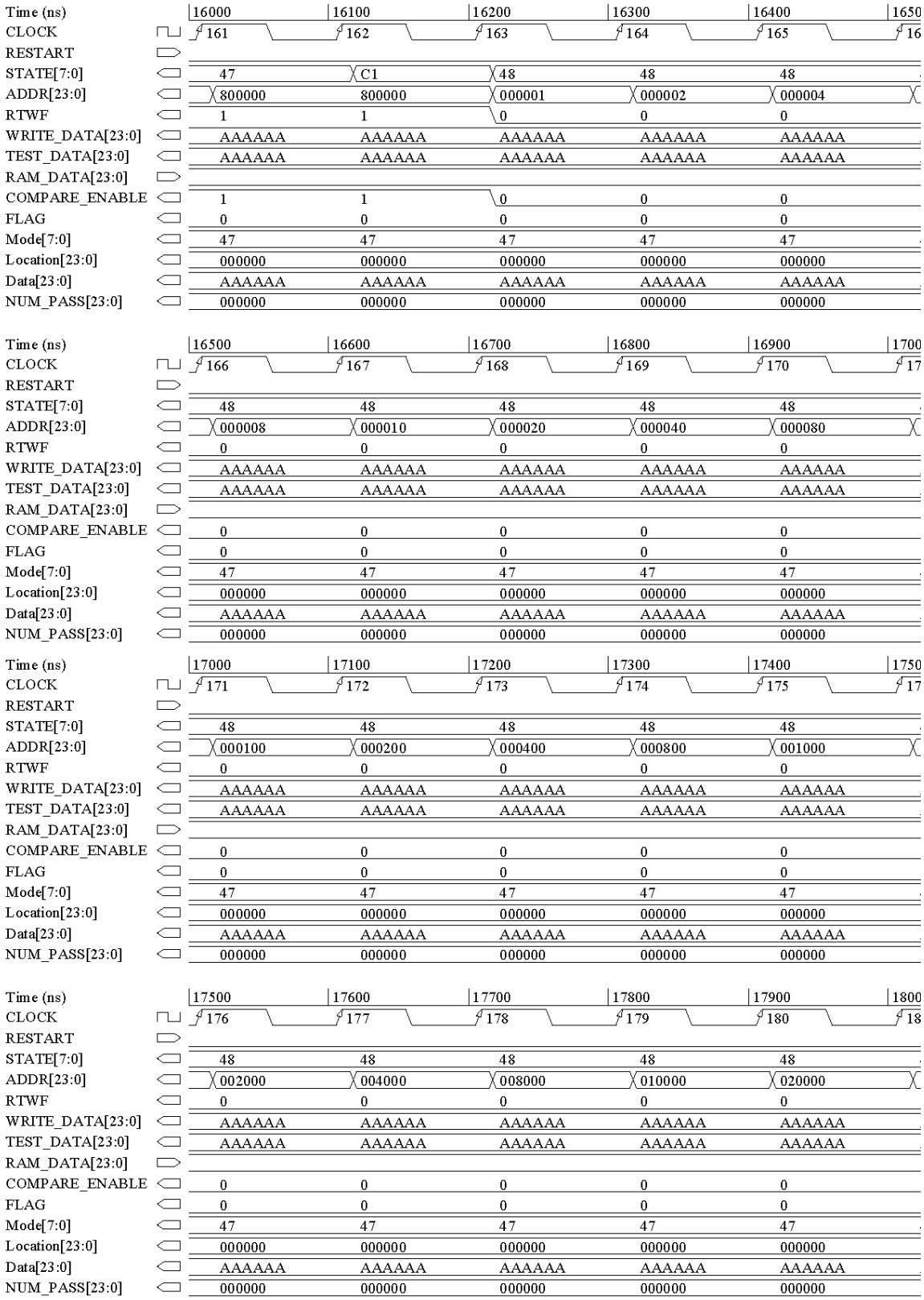

Time (ns) 7500 7600 7700 7800 7900 8000
CLOCK 76 77 78 79 80 81
RESTART
STATE[7:0] 40 40 40 40 40
ADDR[23:0] 008000 010000 020000 040000 080000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 3E 3E 3E 3E 3E
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 800000 800000 800000 800000 800000
NUM_PASS[23:0] 000000 000000 000000 000000 000000

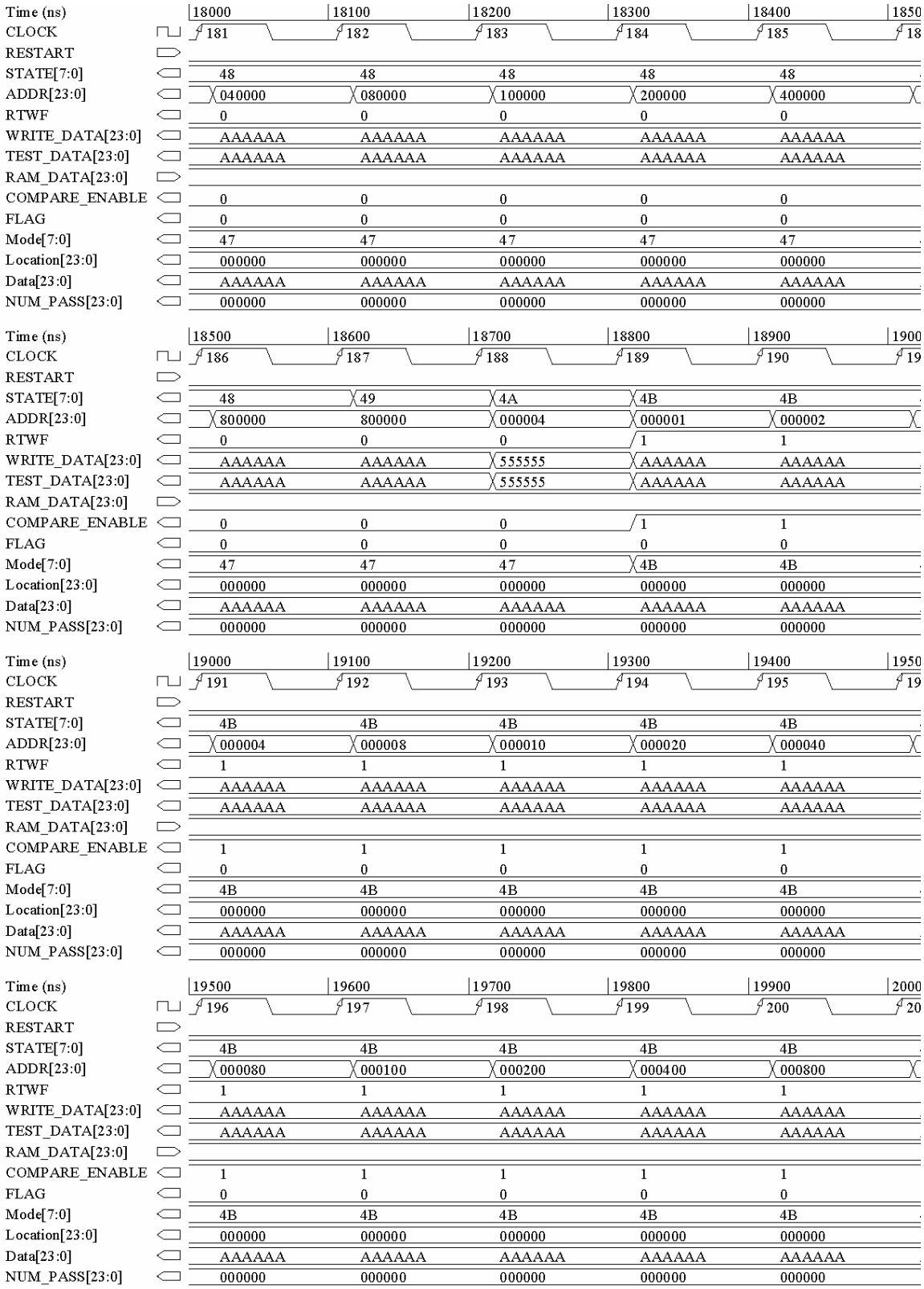

Time (ns) 8000 8100 8200 8300 8400 8500
CLOCK 81 82 83 84 85 86
RESTART
STATE[7:0] 40 40 40 40 41
ADDR[23:0] 100000 200000 400000 800000 800000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 3E 3E 3E 3E 3E
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 800000 800000 800000 800000 800000
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 8500 8600 8700 8800 8900 9000
CLOCK 86 87 88 89 90 91
RESTART
STATE[7:0] 42 43 43 43 43
ADDR[23:0] 000001 000001 000002 000004 000008
RTWF 0 1 1 1 1
WRITE_DATA[23:0] 555555 AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] 555555 AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 3E 43 43 43 43
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] 800000 AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 9000 9100 9200 9300 9400 9500
CLOCK 91 92 93 94 95 96
RESTART
STATE[7:0] 43 43 43 43 43
ADDR[23:0] 000010 000020 000040 000080 000100
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 43 43 43 43 43
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 9500 9600 9700 9800 9900 1000
CLOCK 96 97 98 99 100 10
RESTART
STATE[7:0] 43 43 43 43 43
ADDR[23:0] 000200 000400 000800 001000 002000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 43 43 43 43 43
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

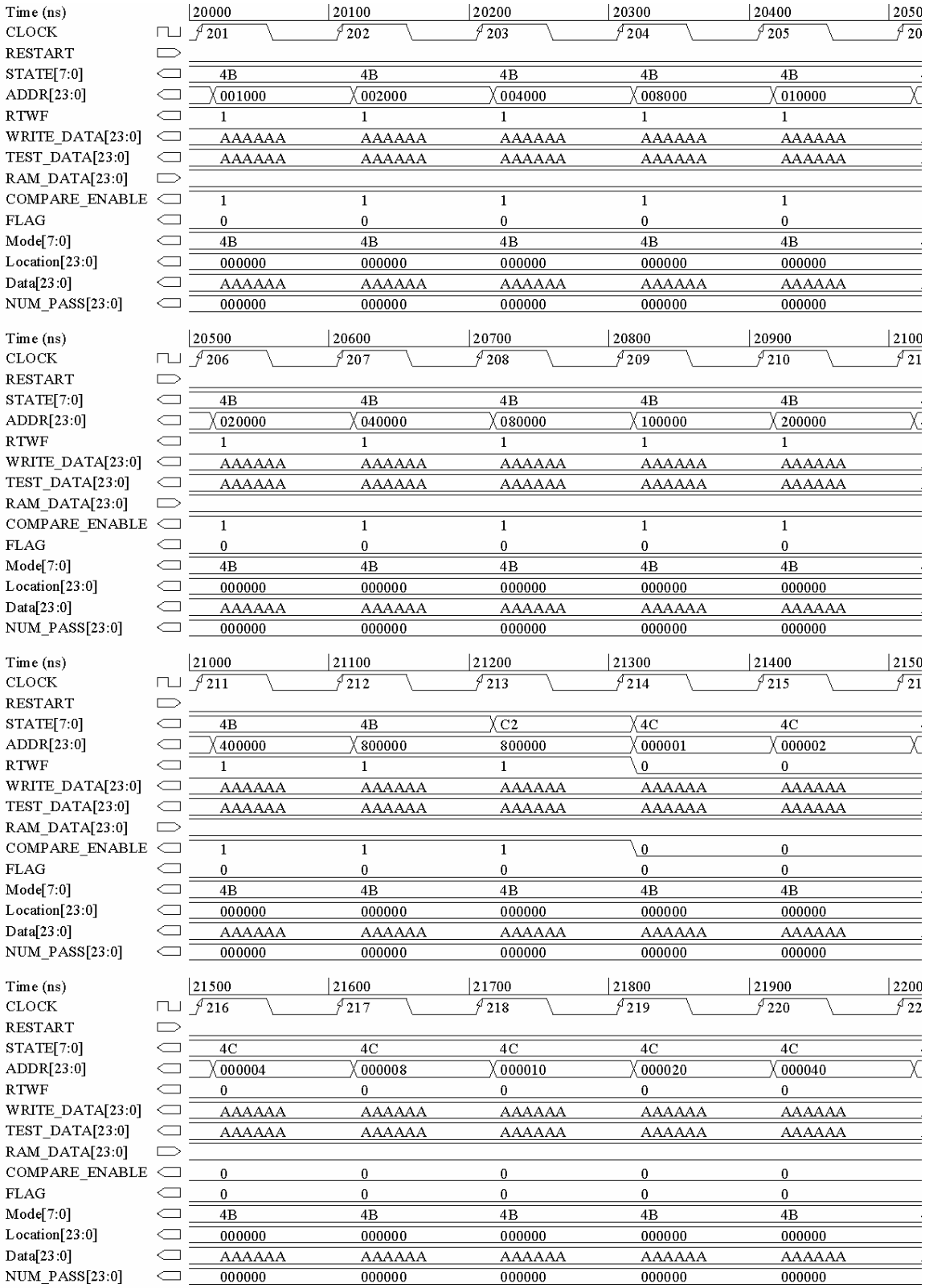

Time (ns) 10000 10100 10200 10300 10400 1050
CLOCK 101 102 103 104 105 10
RESTART
STATE[7:0] 43 43 43 43 43
ADDR[23:0] 004000 008000 010000 020000 040000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 43 43 43 43 43
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 10500 10600 10700 10800 10900 1100
CLOCK 106 107 108 109 110 11
RESTART
STATE[7:0] 43 43 43 43 43
ADDR[23:0] 080000 100000 200000 400000 800000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 43 43 43 43 43
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 11000 11100 11200 11300 11400 1150
CLOCK 111 112 113 114 115 11
RESTART
STATE[7:0] C0 44 44 44 44
ADDR[23:0] 800000 000001 000002 000004 000008
RTWF 1 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 43 43 43 43 43
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 11500 11600 11700 11800 11900 1200
CLOCK 116 117 118 119 120 12
RESTART
STATE[7:0] 44 44 44 44 44
ADDR[23:0] 000010 000020 000040 000080 000100
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 43 43 43 43 43
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

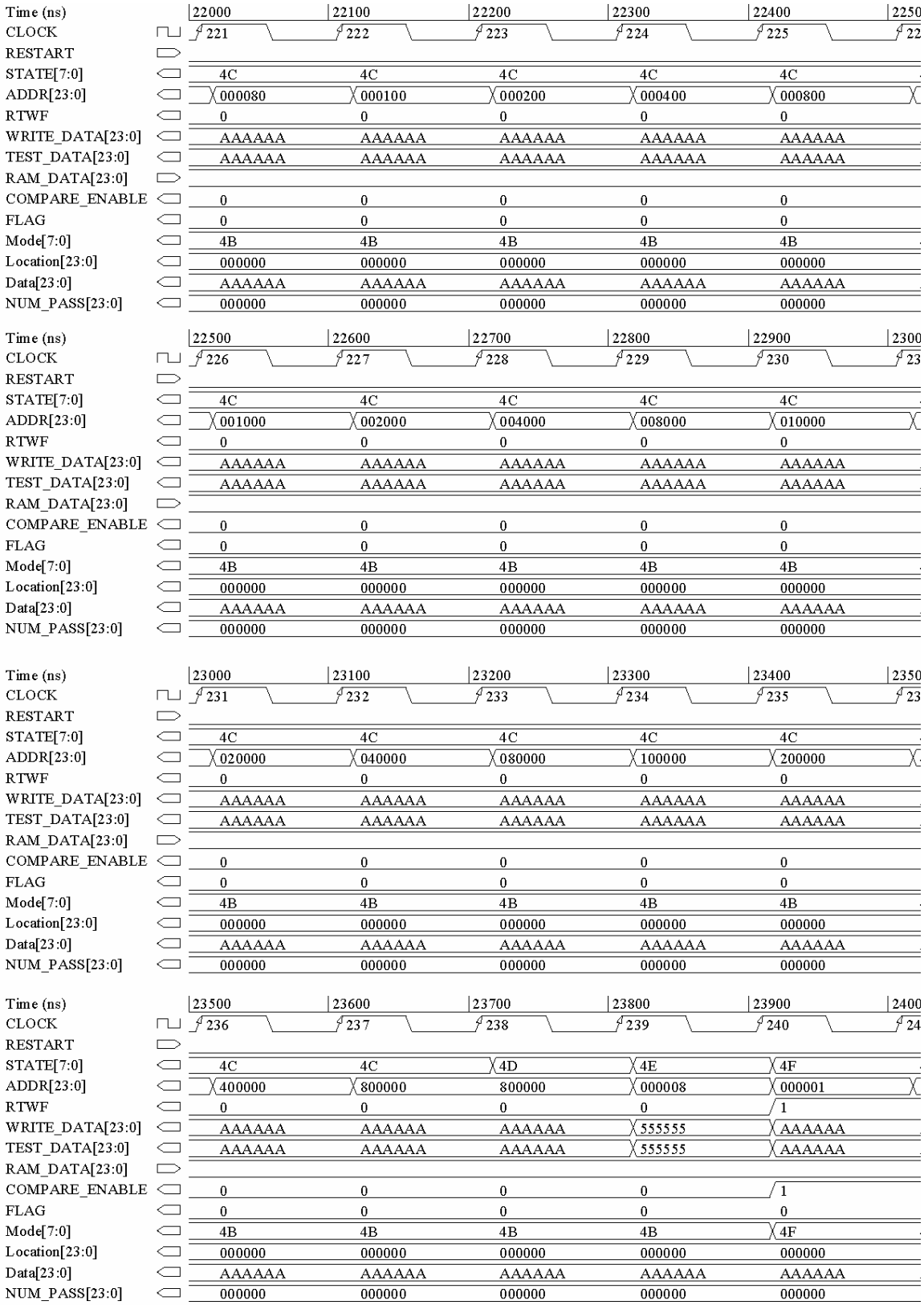

Time (ns) 12000 12100 12200 12300 12400 1250
CLOCK 121 122 123 124 125 12
RESTART
STATE[7:0] 44 44 44 44 44
ADDR[23:0] 000200 000400 000800 001000 002000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 43 43 43 43 43
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 12500 12600 12700 12800 12900 1300
CLOCK 126 127 128 129 130 13
RESTART
STATE[7:0] 44 44 44 44 44
ADDR[23:0] 004000 008000 010000 020000 040000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 43 43 43 43 43
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 13000 13100 13200 13300 13400 1350
CLOCK 131 132 133 134 135 13
RESTART
STATE[7:0] 44 44 44 44 44
ADDR[23:0] 080000 100000 200000 400000 800000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 43 43 43 43 43
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 13500 13600 13700 13800 13900 1400
CLOCK 136 137 138 139 140 14
RESTART
STATE[7:0] 45 46 47 47 47
ADDR[23:0] 800000 000002 000001 000002 000004
RTWF 0 0 1 1 1
WRITE_DATA[23:0] AAAAAA 555555 AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA 555555 AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 43 43 47 47 47
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

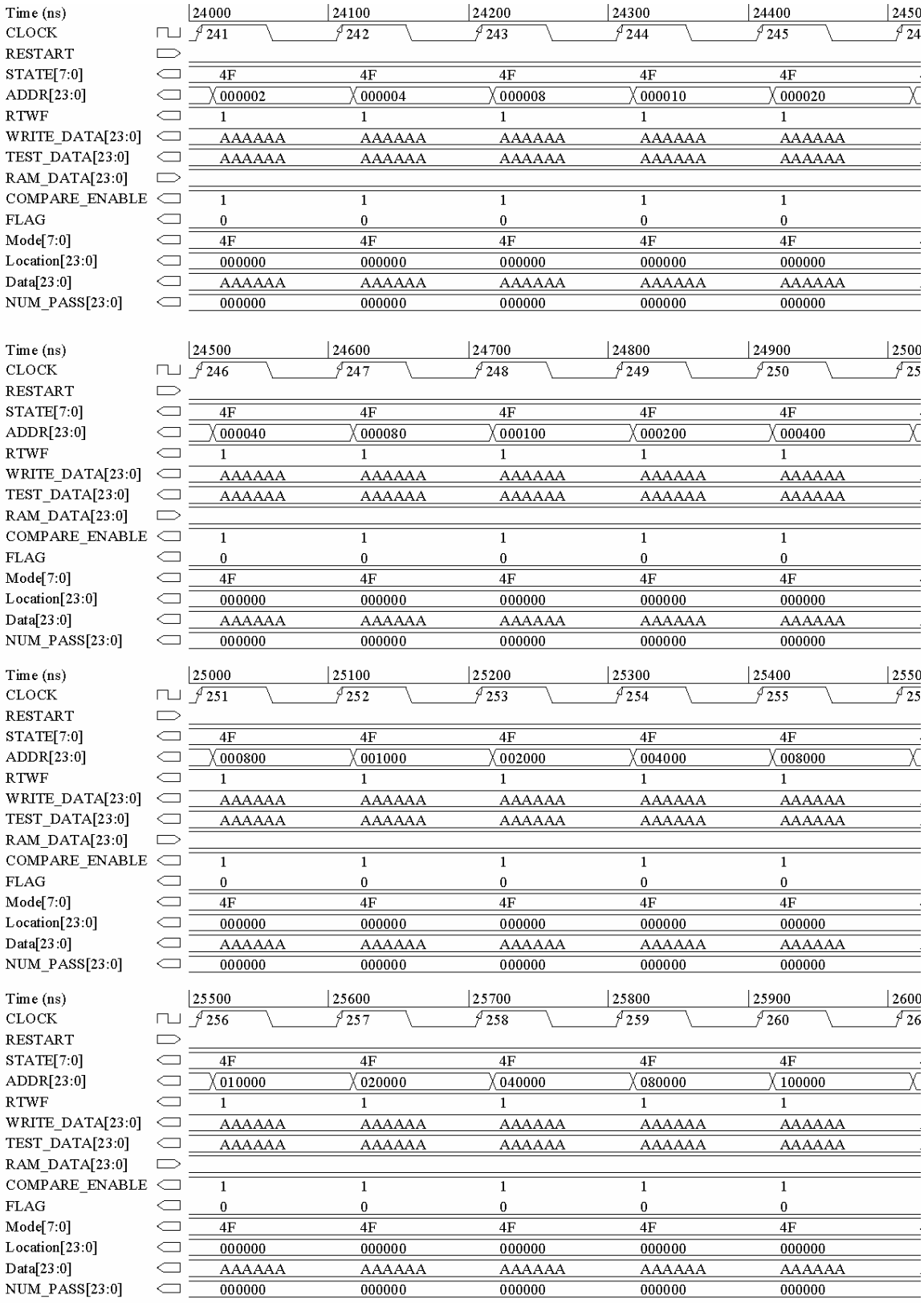

Time (ns) 14000 14100 14200 14300 14400 1450
CLOCK 141 142 143 144 145 14
RESTART
STATE[7:0] 47 47 47 47 47
ADDR[23:0] 000008 000010 000020 000040 000080
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 47 47 47 47 47
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 14500 14600 14700 14800 14900 1500
CLOCK 146 147 148 149 150 15
RESTART
STATE[7:0] 47 47 47 47 47
ADDR[23:0] 000100 000200 000400 000800 001000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 47 47 47 47 47
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 15000 15100 15200 15300 15400 1550
CLOCK 151 152 153 154 155 15
RESTART
STATE[7:0] 47 47 47 47 47
ADDR[23:0] 002000 004000 008000 010000 020000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 47 47 47 47 47
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 15500 15600 15700 15800 15900 1600
CLOCK 156 157 158 159 160 16
RESTART
STATE[7:0] 47 47 47 47 47
ADDR[23:0] 040000 080000 100000 200000 400000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 47 47 47 47 47
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

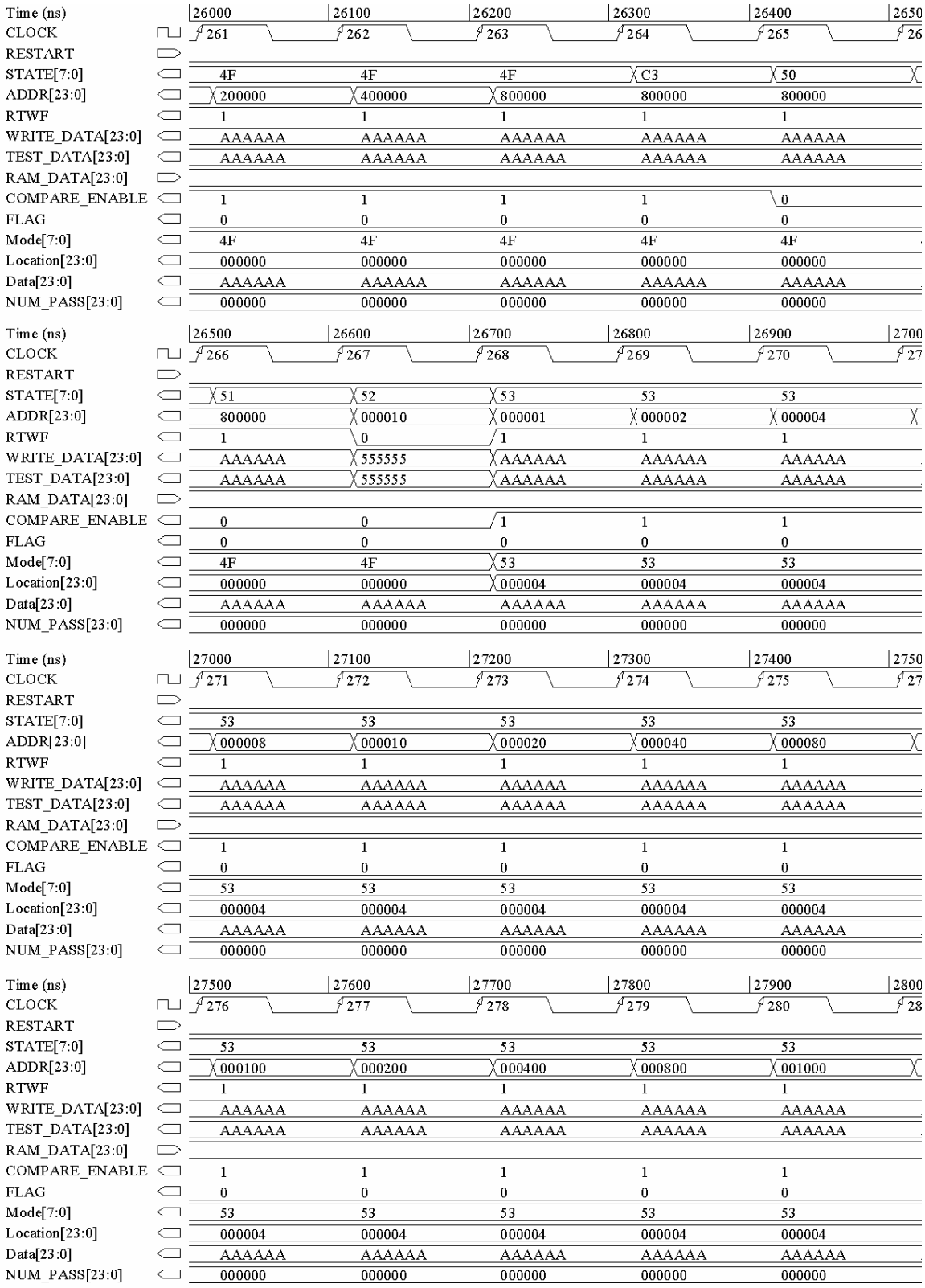

Time (ns) 16000 16100 16200 16300 16400 1650
CLOCK 161 162 163 164 165 16
RESTART
STATE[7:0] 47 C1 48 48 48
ADDR[23:0] 800000 800000 000001 000002 000004
RTWF 1 1 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 47 47 47 47 47
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 16500 16600 16700 16800 16900 1700
CLOCK 166 167 168 169 170 17
RESTART
STATE[7:0] 48 48 48 48 48
ADDR[23:0] 000008 000010 000020 000040 000080
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 47 47 47 47 47
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 17000 17100 17200 17300 17400 1750
CLOCK 171 172 173 174 175 17
RESTART
STATE[7:0] 48 48 48 48 48
ADDR[23:0] 000100 000200 000400 000800 001000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 47 47 47 47 47
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 17500 17600 17700 17800 17900 1800
CLOCK 176 177 178 179 180 18
RESTART
STATE[7:0] 48 48 48 48 48
ADDR[23:0] 002000 004000 008000 010000 020000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 47 47 47 47 47
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

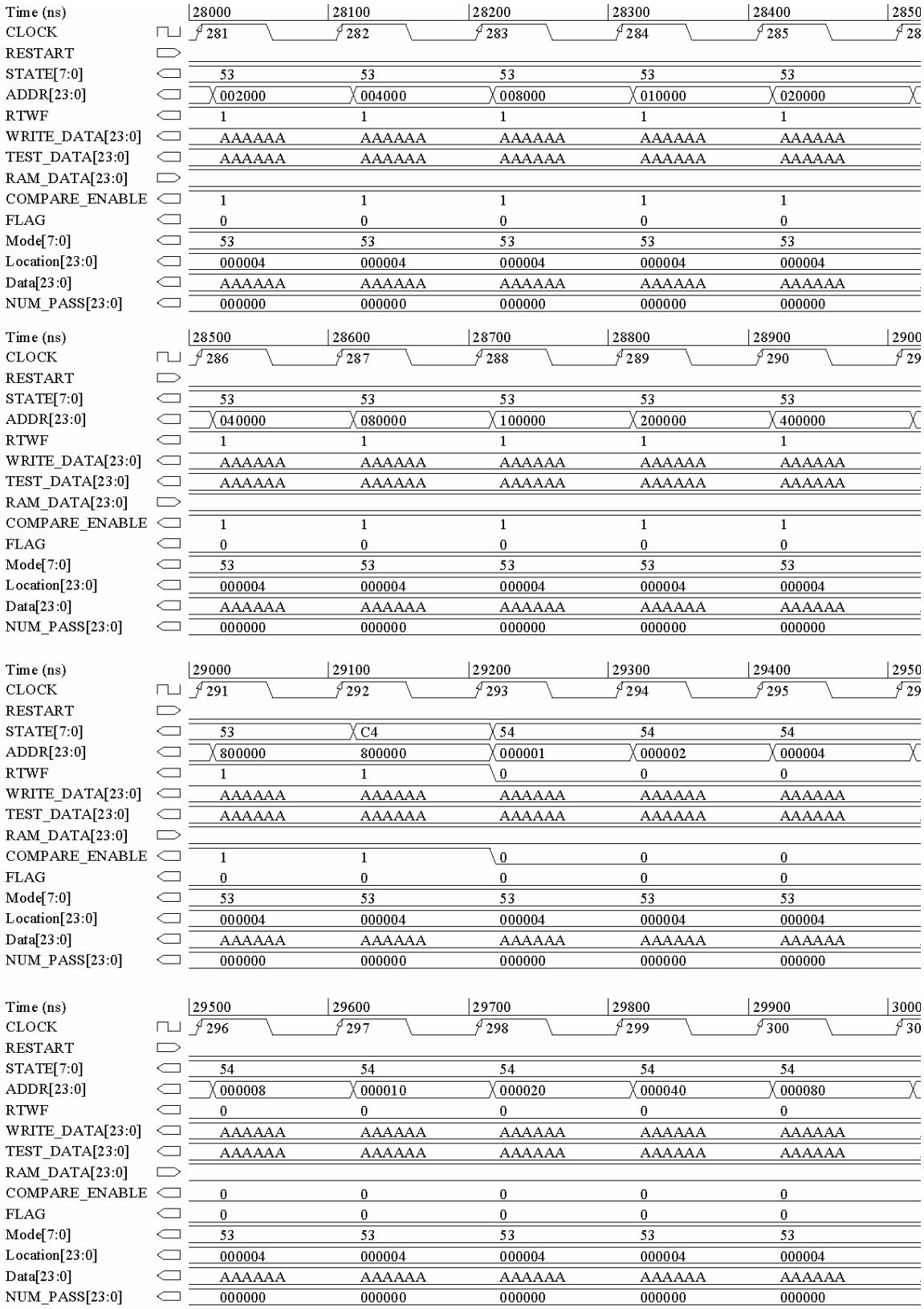

Time (ns) 18000 18100 18200 18300 18400 1850
CLOCK 181 182 183 184 185 18
RESTART
STATE[7:0] 48 48 48 48 48
ADDR[23:0] 040000 080000 100000 200000 400000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 47 47 47 47 47
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 18500 18600 18700 18800 18900 1900
CLOCK 186 187 188 189 190 19
RESTART
STATE[7:0] 48 49 4A 4B 4B
ADDR[23:0] 800000 800000 000004 000001 000002
RTWF 0 0 0 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA 555555 AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA 555555 AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 1 1
FLAG 0 0 0 0 0
Mode[7:0] 47 47 47 4B 4B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 19000 19100 19200 19300 19400 1950
CLOCK 191 192 193 194 195 19
RESTART
STATE[7:0] 4B 4B 4B 4B 4B
ADDR[23:0] 000004 000008 000010 000020 000040
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 4B 4B 4B 4B 4B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 19500 19600 19700 19800 19900 2000
CLOCK 196 197 198 199 200 20
RESTART
STATE[7:0] 4B 4B 4B 4B 4B
ADDR[23:0] 000080 000100 000200 000400 000800
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 4B 4B 4B 4B 4B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

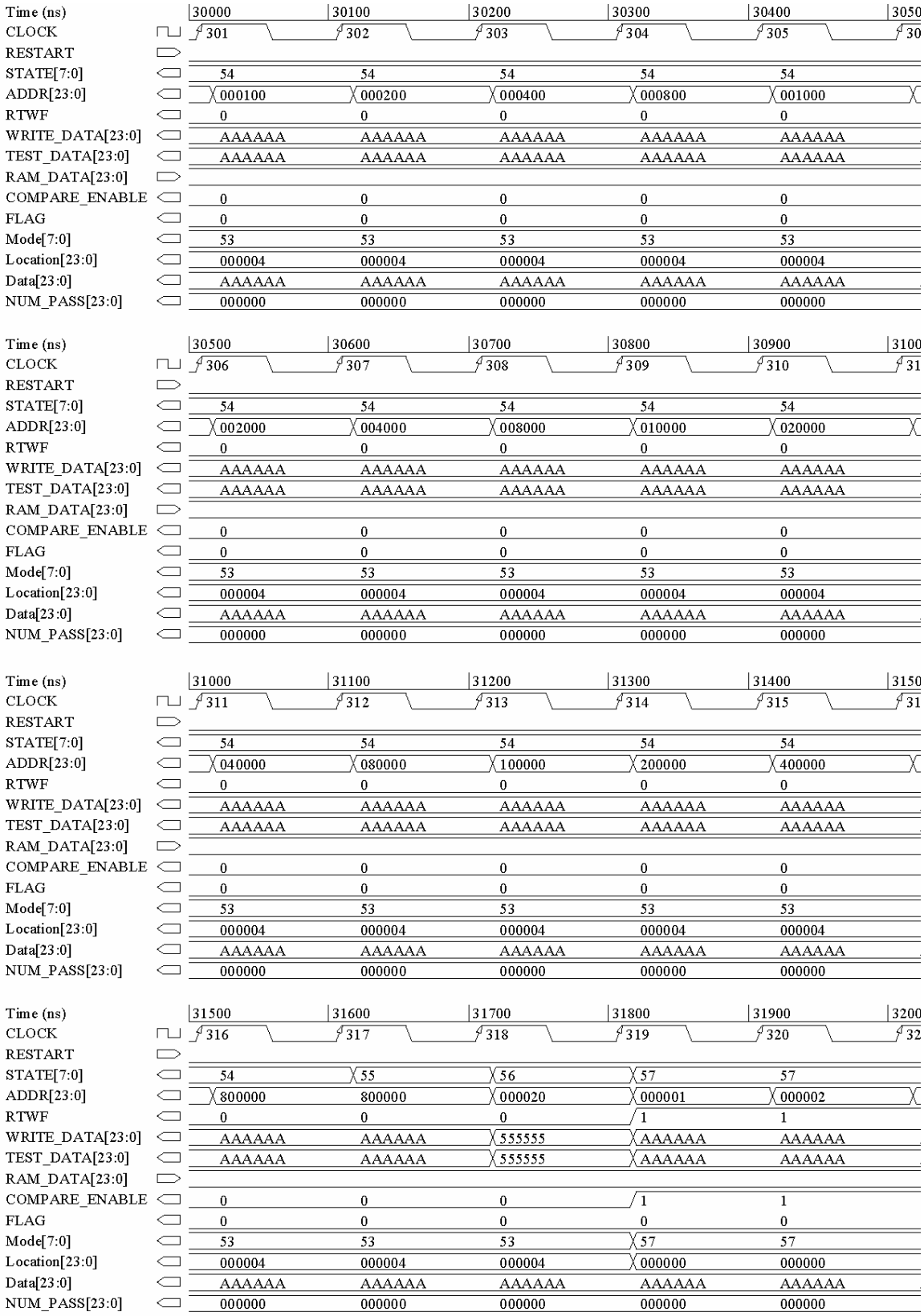

Time (ns) 20000 20100 20200 20300 20400 2050
CLOCK 201 202 203 204 205 20
RESTART
STATE[7:0] 4B 4B 4B 4B 4B
ADDR[23:0] 001000 002000 004000 008000 010000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 4B 4B 4B 4B 4B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 20500 20600 20700 20800 20900 2100
CLOCK 206 207 208 209 210 21
RESTART
STATE[7:0] 4B 4B 4B 4B 4B
ADDR[23:0] 020000 040000 080000 100000 200000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 4B 4B 4B 4B 4B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 21000 21100 21200 21300 21400 2150
CLOCK 211 212 213 214 215 21
RESTART
STATE[7:0] 4B 4B C2 4C 4C
ADDR[23:0] 400000 800000 800000 000001 000002
RTWF 1 1 1 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 0 0
FLAG 0 0 0 0 0
Mode[7:0] 4B 4B 4B 4B 4B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 21500 21600 21700 21800 21900 2200
CLOCK 216 217 218 219 220 22
RESTART
STATE[7:0] 4C 4C 4C 4C 4C
ADDR[23:0] 000004 000008 000010 000020 000040
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 4B 4B 4B 4B 4B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

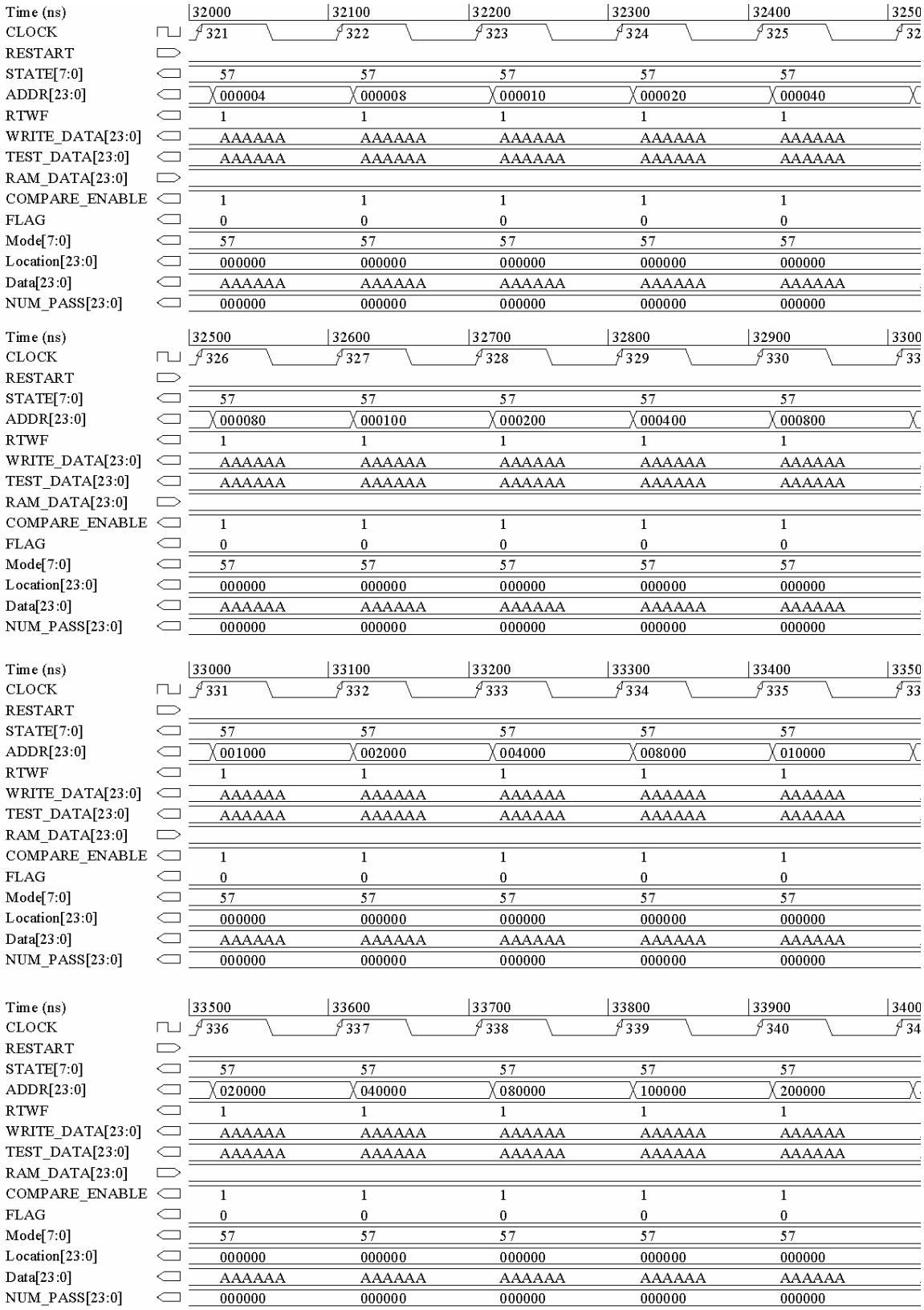

Time (ns) 22000 22100 22200 22300 22400 2250
CLOCK 221 222 223 224 225 22
RESTART
STATE[7:0] 4C 4C 4C 4C 4C
ADDR[23:0] 000080 000100 000200 000400 000800
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 4B 4B 4B 4B 4B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 22500 22600 22700 22800 22900 2300
CLOCK 226 227 228 229 230 23
RESTART
STATE[7:0] 4C 4C 4C 4C 4C
ADDR[23:0] 001000 002000 004000 008000 010000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 4B 4B 4B 4B 4B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 23000 23100 23200 23300 23400 2350
CLOCK 231 232 233 234 235 23
RESTART
STATE[7:0] 4C 4C 4C 4C 4C
ADDR[23:0] 020000 040000 080000 100000 200000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 4B 4B 4B 4B 4B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 23500 23600 23700 23800 23900 2400
CLOCK 236 237 238 239 240 24
RESTART
STATE[7:0] 4C 4C 4D 4E 4F
ADDR[23:0] 400000 800000 800000 000008 000001
RTWF 0 0 0 0 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA 555555 AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA 555555 AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 1
FLAG 0 0 0 0 0
Mode[7:0] 4B 4B 4B 4B 4F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

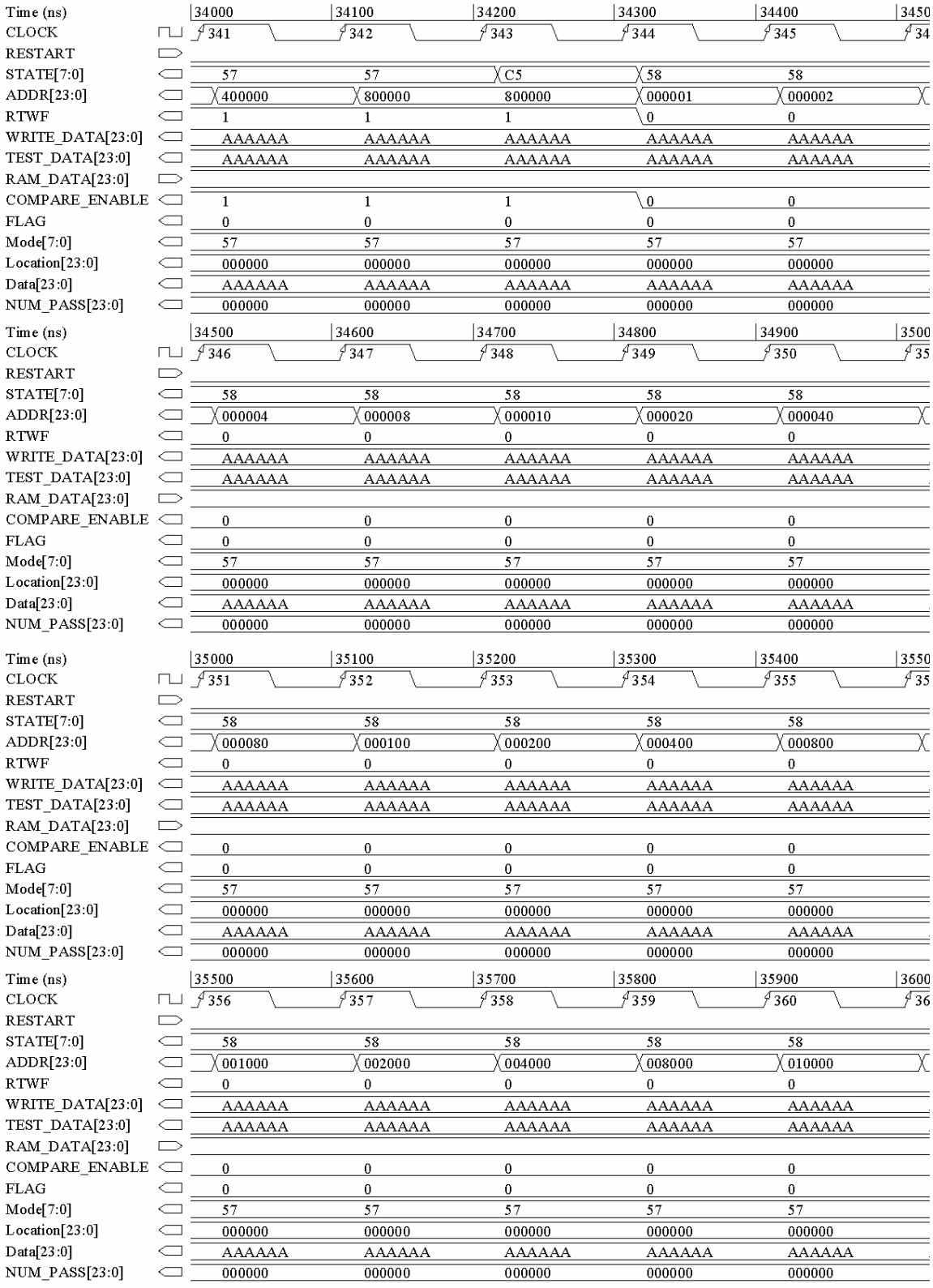

Time (ns) 24000 24100 24200 24300 24400 2450
CLOCK 241 242 243 244 245 24
RESTART
STATE[7:0] 4F 4F 4F 4F 4F
ADDR[23:0] 000002 000004 000008 000010 000020
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 4F 4F 4F 4F 4F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 24500 24600 24700 24800 24900 2500
CLOCK 246 247 248 249 250 25
RESTART
STATE[7:0] 4F 4F 4F 4F 4F
ADDR[23:0] 000040 000080 000100 000200 000400
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 4F 4F 4F 4F 4F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 25000 25100 25200 25300 25400 2550
CLOCK 251 252 253 254 255 25
RESTART
STATE[7:0] 4F 4F 4F 4F 4F
ADDR[23:0] 000800 001000 002000 004000 008000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 4F 4F 4F 4F 4F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 25500 25600 25700 25800 25900 2600
CLOCK 256 257 258 259 260 26
RESTART
STATE[7:0] 4F 4F 4F 4F 4F
ADDR[23:0] 010000 020000 040000 080000 100000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 4F 4F 4F 4F 4F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

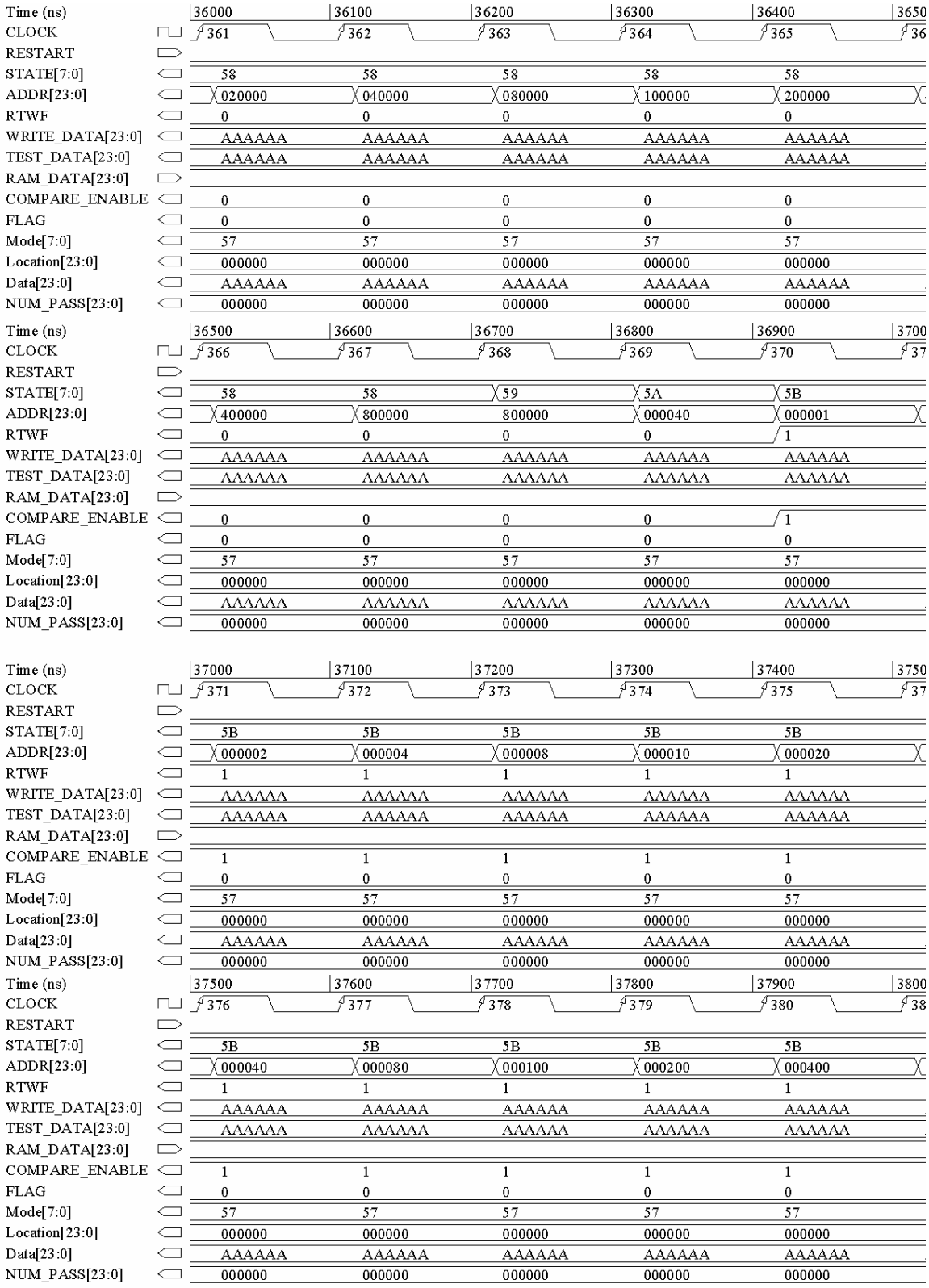

Time (ns) 26000 26100 26200 26300 26400 2650
CLOCK 261 262 263 264 265 26
RESTART
STATE[7:0] 4F 4F 4F C3 50
ADDR[23:0] 200000 400000 800000 800000 800000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 0
FLAG 0 0 0 0 0
Mode[7:0] 4F 4F 4F 4F 4F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 26500 26600 26700 26800 26900 2700
CLOCK 266 267 268 269 270 27
RESTART
STATE[7:0] 51 52 53 53 53
ADDR[23:0] 800000 000010 000001 000002 000004
RTWF 1 0 1 1 1
WRITE_DATA[23:0] AAAAAA 555555 AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA 555555 AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 4F 4F 53 53 53
Location[23:0] 000000 000000 000004 000004 000004
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 27000 27100 27200 27300 27400 2750
CLOCK 271 272 273 274 275 27
RESTART
STATE[7:0] 53 53 53 53 53
ADDR[23:0] 000008 000010 000020 000040 000080
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 53 53 53 53 53
Location[23:0] 000004 000004 000004 000004 000004
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 27500 27600 27700 27800 27900 2800
CLOCK 276 277 278 279 280 28
RESTART
STATE[7:0] 53 53 53 53 53
ADDR[23:0] 000100 000200 000400 000800 001000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 53 53 53 53 53
Location[23:0] 000004 000004 000004 000004 000004
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

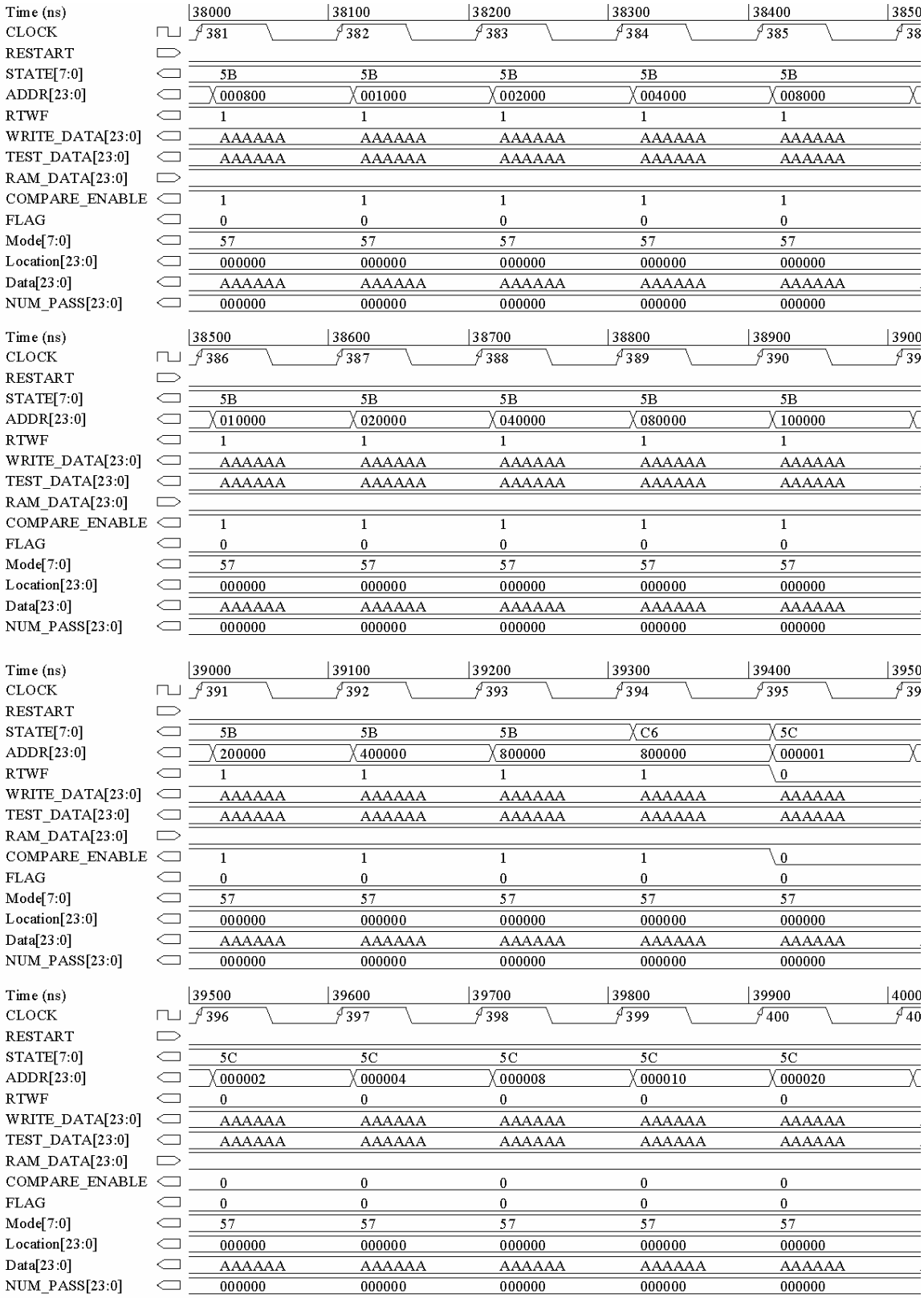

Time (ns) 28000 28100 28200 28300 28400 2850
CLOCK 281 282 283 284 285 28
RESTART
STATE[7:0] 53 53 53 53 53
ADDR[23:0] 002000 004000 008000 010000 020000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 53 53 53 53 53
Location[23:0] 000004 000004 000004 000004 000004
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 28500 28600 28700 28800 28900 2900
CLOCK 286 287 288 289 290 29
RESTART
STATE[7:0] 53 53 53 53 53
ADDR[23:0] 040000 080000 100000 200000 400000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 53 53 53 53 53
Location[23:0] 000004 000004 000004 000004 000004
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 29000 29100 29200 29300 29400 2950
CLOCK 291 292 293 294 295 29
RESTART
STATE[7:0] 53 C4 54 54 54
ADDR[23:0] 800000 800000 000001 000002 000004
RTWF 1 1 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 53 53 53 53 53
Location[23:0] 000004 000004 000004 000004 000004
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 29500 29600 29700 29800 29900 3000
CLOCK 296 297 298 299 300 30
RESTART
STATE[7:0] 54 54 54 54 54
ADDR[23:0] 000008 000010 000020 000040 000080
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 53 53 53 53 53
Location[23:0] 000004 000004 000004 000004 000004
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

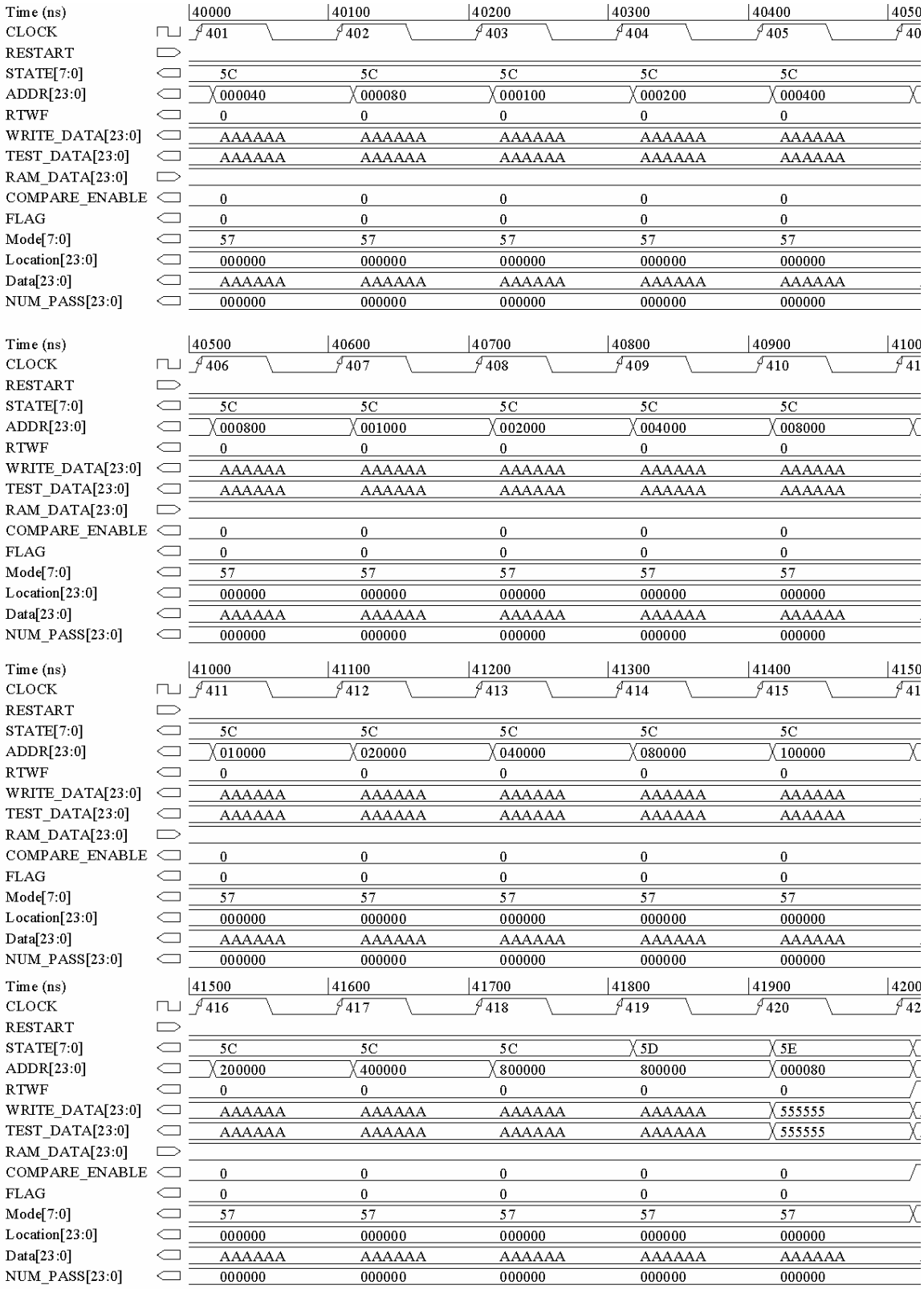

Time (ns) 30000 30100 30200 30300 30400 3050
CLOCK 301 302 303 304 305 30
RESTART
STATE[7:0] 54 54 54 54 54
ADDR[23:0] 000100 000200 000400 000800 001000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 53 53 53 53 53
Location[23:0] 000004 000004 000004 000004 000004
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 30500 30600 30700 30800 30900 3100
CLOCK 306 307 308 309 310 31
RESTART
STATE[7:0] 54 54 54 54 54
ADDR[23:0] 002000 004000 008000 010000 020000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 53 53 53 53 53
Location[23:0] 000004 000004 000004 000004 000004
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 31000 31100 31200 31300 31400 3150
CLOCK 311 312 313 314 315 31
RESTART
STATE[7:0] 54 54 54 54 54
ADDR[23:0] 040000 080000 100000 200000 400000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 53 53 53 53 53
Location[23:0] 000004 000004 000004 000004 000004
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 31500 31600 31700 31800 31900 3200
CLOCK 316 317 318 319 320 32
RESTART
STATE[7:0] 54 55 56 57 57
ADDR[23:0] 800000 800000 000020 000001 000002
RTWF 0 0 0 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA 555555 AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA 555555 AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 1 1
FLAG 0 0 0 0 0
Mode[7:0] 53 53 53 57 57
Location[23:0] 000004 000004 000004 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

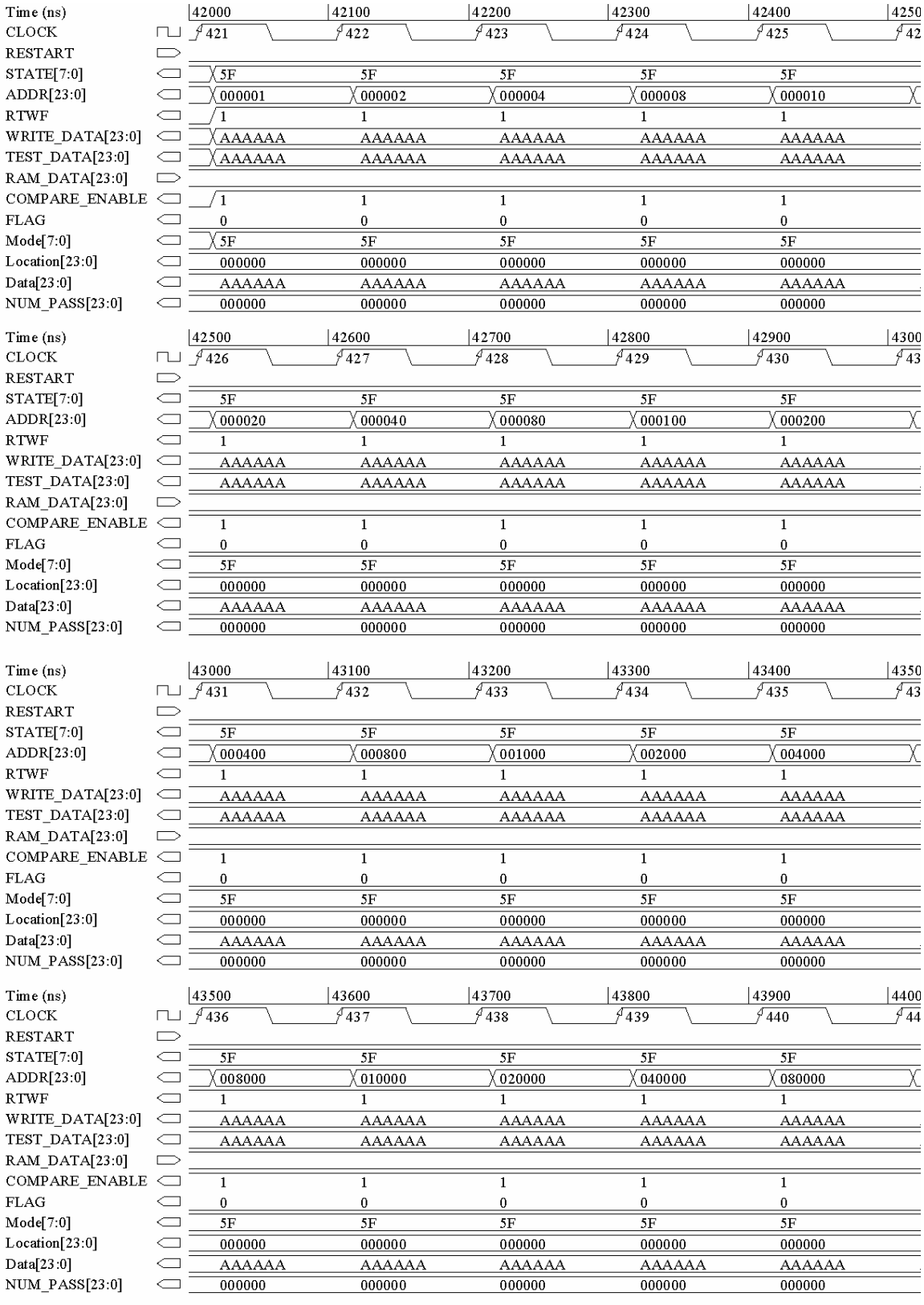

Time (ns) 32000 32100 32200 32300 32400 3250
CLOCK 321 322 323 324 325 32
RESTART
STATE[7:0] 57 57 57 57 57
ADDR[23:0] 000004 000008 000010 000020 000040
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 32500 32600 32700 32800 32900 3300
CLOCK 326 327 328 329 330 33
RESTART
STATE[7:0] 57 57 57 57 57
ADDR[23:0] 000080 000100 000200 000400 000800
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 33000 33100 33200 33300 33400 3350
CLOCK 331 332 333 334 335 33
RESTART
STATE[7:0] 57 57 57 57 57
ADDR[23:0] 001000 002000 004000 008000 010000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 33500 33600 33700 33800 33900 3400
CLOCK 336 337 338 339 340 34
RESTART
STATE[7:0] 57 57 57 57 57
ADDR[23:0] 020000 040000 080000 100000 200000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

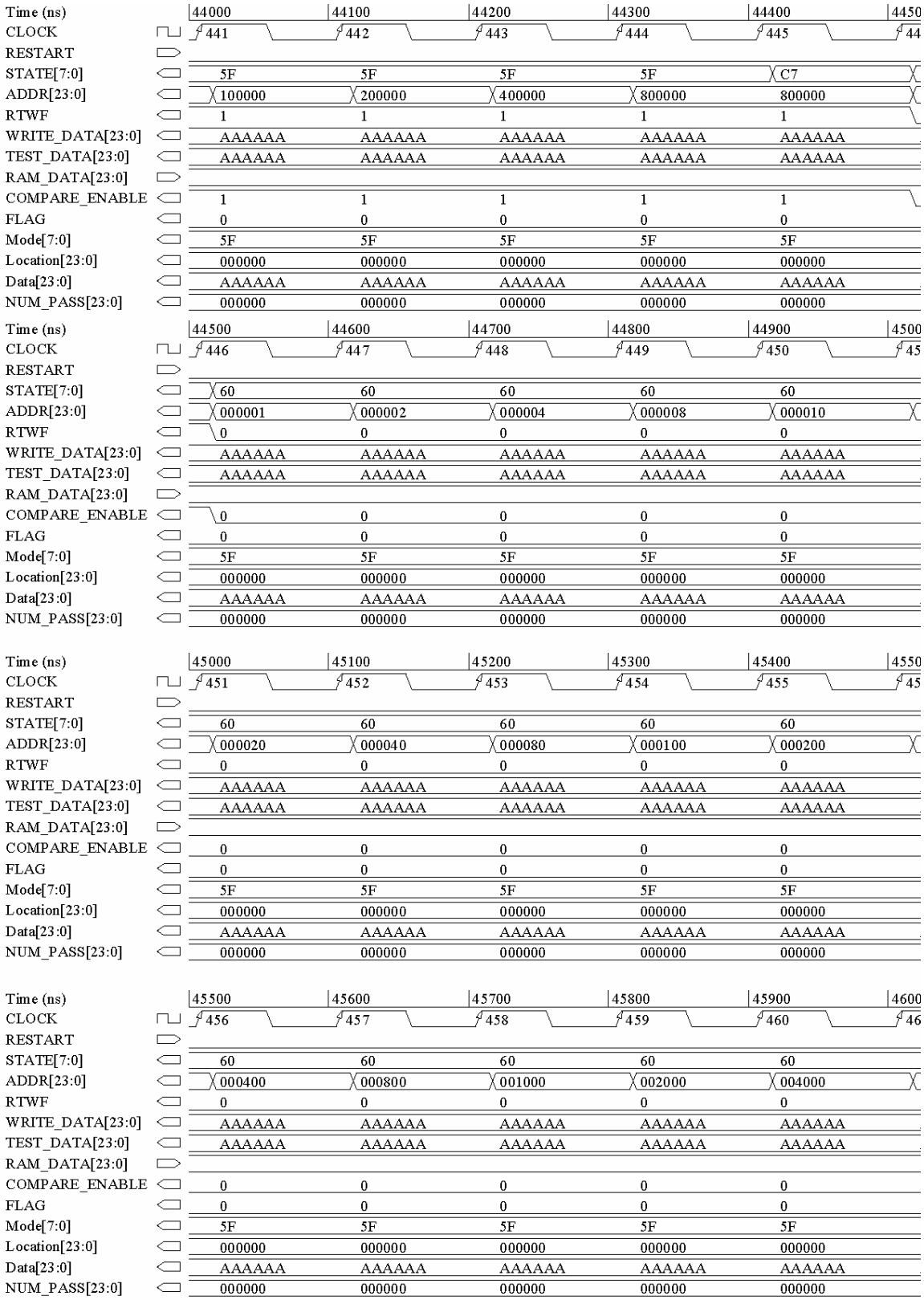

Time (ns) 34000 34100 34200 34300 34400 3450
CLOCK 341 342 343 344 345 34
RESTART
STATE[7:0] 57 57 C5 58 58
ADDR[23:0] 400000 800000 800000 000001 000002
RTWF 1 1 1 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 0 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 34500 34600 34700 34800 34900 3500
CLOCK 346 347 348 349 350 35
RESTART
STATE[7:0] 58 58 58 58 58
ADDR[23:0] 000004 000008 000010 000020 000040
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 35000 35100 35200 35300 35400 3550
CLOCK 351 352 353 354 355 35
RESTART
STATE[7:0] 58 58 58 58 58
ADDR[23:0] 000080 000100 000200 000400 000800
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 35500 35600 35700 35800 35900 3600
CLOCK 356 357 358 359 360 36
RESTART
STATE[7:0] 58 58 58 58 58
ADDR[23:0] 001000 002000 004000 008000 010000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

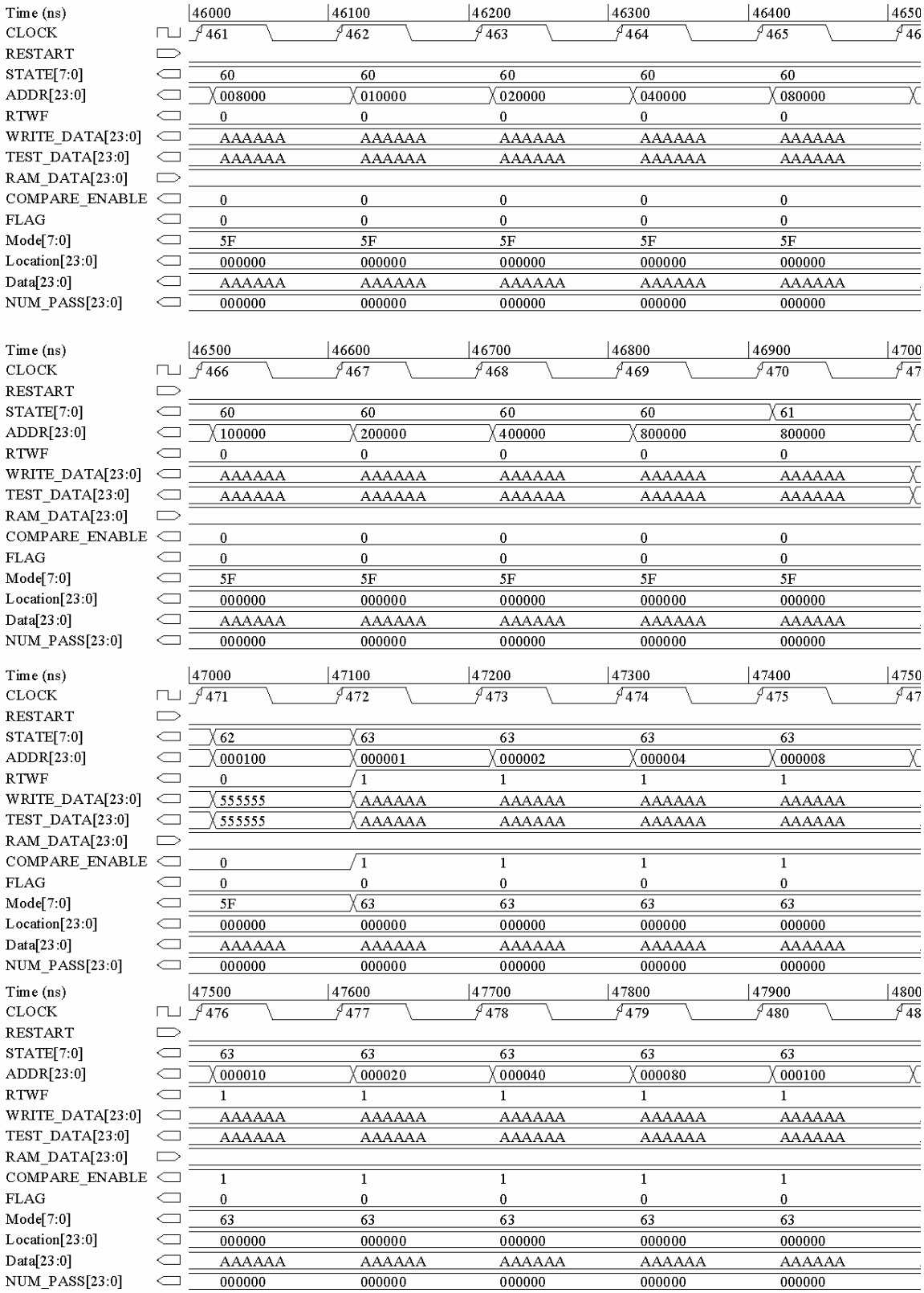

Time (ns) 36000 36100 36200 36300 36400 3650
CLOCK 361 362 363 364 365 36
RESTART
STATE[7:0] 58 58 58 58 58
ADDR[23:0] 020000 040000 080000 100000 200000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 36500 36600 36700 36800 36900 3700
CLOCK 366 367 368 369 370 37
RESTART
STATE[7:0] 58 58 59 5A 5B
ADDR[23:0] 400000 800000 800000 000040 000001
RTWF 0 0 0 0 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 1
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 37000 37100 37200 37300 37400 3750
CLOCK 371 372 373 374 375 37
RESTART
STATE[7:0] 5B 5B 5B 5B 5B
ADDR[23:0] 000002 000004 000008 000010 000020
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 37500 37600 37700 37800 37900 3800
CLOCK 376 377 378 379 380 38
RESTART
STATE[7:0] 5B 5B 5B 5B 5B
ADDR[23:0] 000040 000080 000100 000200 000400
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

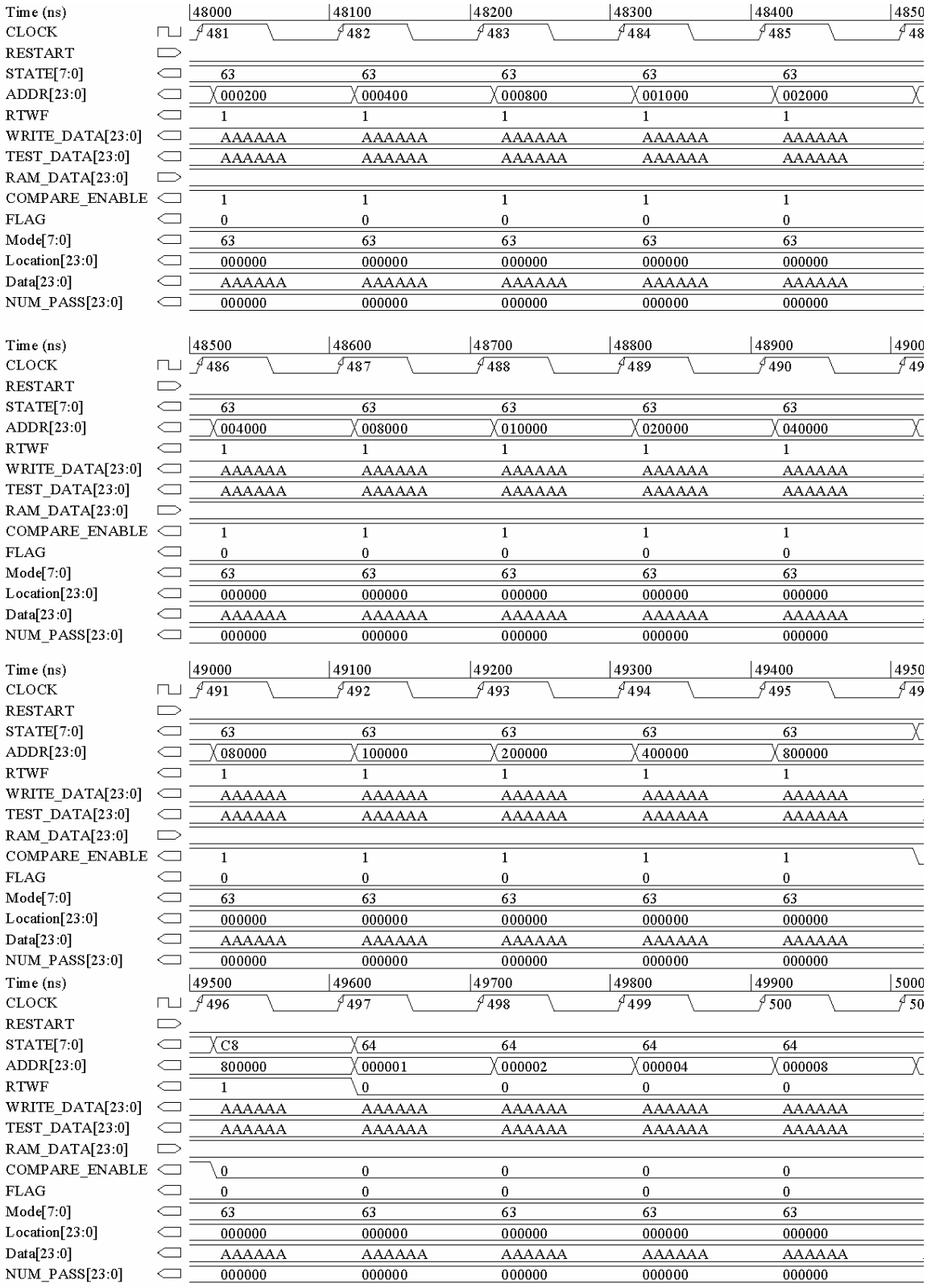

Time (ns) 38000 38100 38200 38300 38400 3850
CLOCK 381 382 383 384 385 38
RESTART
STATE[7:0] 5B 5B 5B 5B 5B
ADDR[23:0] 000800 001000 002000 004000 008000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 38500 38600 38700 38800 38900 3900
CLOCK 386 387 388 389 390 39
RESTART
STATE[7:0] 5B 5B 5B 5B 5B
ADDR[23:0] 010000 020000 040000 080000 100000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 39000 39100 39200 39300 39400 3950
CLOCK 391 392 393 394 395 39
RESTART
STATE[7:0] 5B 5B 5B C6 5C
ADDR[23:0] 200000 400000 800000 800000 000001
RTWF 1 1 1 1 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 39500 39600 39700 39800 39900 4000
CLOCK 396 397 398 399 400 40
RESTART
STATE[7:0] 5C 5C 5C 5C 5C
ADDR[23:0] 000002 000004 000008 000010 000020
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

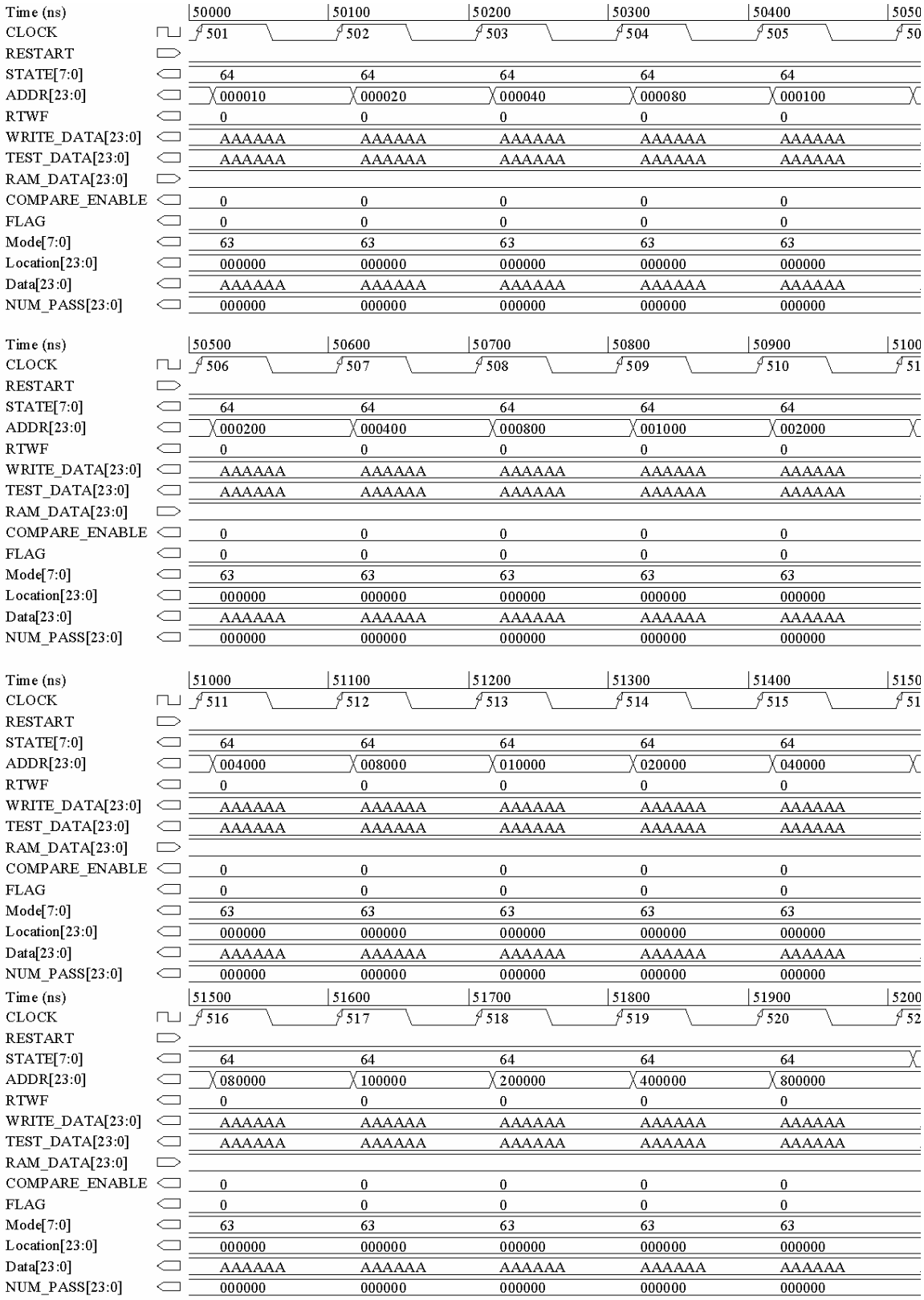

Time (ns) 40000 40100 40200 40300 40400 4050
CLOCK 401 402 403 404 405 40
RESTART
STATE[7:0] 5C 5C 5C 5C 5C
ADDR[23:0] 000040 000080 000100 000200 000400
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 40500 40600 40700 40800 40900 4100
CLOCK 406 407 408 409 410 41
RESTART
STATE[7:0] 5C 5C 5C 5C 5C
ADDR[23:0] 000800 001000 002000 004000 008000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 41000 41100 41200 41300 41400 4150
CLOCK 411 412 413 414 415 41
RESTART
STATE[7:0] 5C 5C 5C 5C 5C
ADDR[23:0] 010000 020000 040000 080000 100000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 41500 41600 41700 41800 41900 4200
CLOCK 416 417 418 419 420 42
RESTART
STATE[7:0] 5C 5C 5C 5D 5E
ADDR[23:0] 200000 400000 800000 800000 000080
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA 555555
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA 555555
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 57 57 57 57 57
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

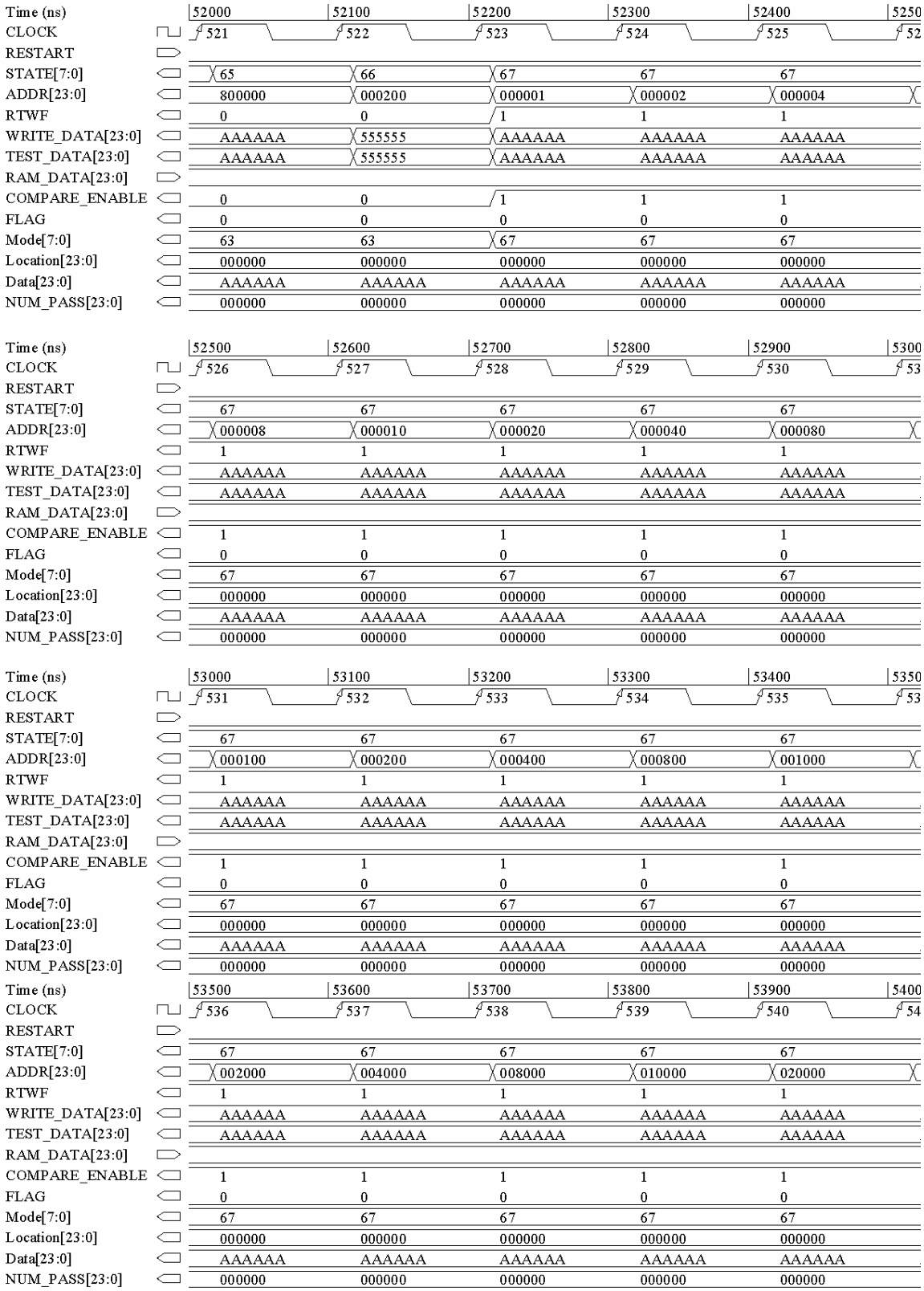

Time (ns) 42000 42100 42200 42300 42400 4250
CLOCK 421 422 423 424 425 42
RESTART
STATE[7:0] 5F 5F 5F 5F 5F
ADDR[23:0] 000001 000002 000004 000008 000010
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 5F 5F 5F 5F 5F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 42500 42600 42700 42800 42900 4300
CLOCK 426 427 428 429 430 43
RESTART
STATE[7:0] 5F 5F 5F 5F 5F
ADDR[23:0] 000020 000040 000080 000100 000200
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 5F 5F 5F 5F 5F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 43000 43100 43200 43300 43400 4350
CLOCK 431 432 433 434 435 43
RESTART
STATE[7:0] 5F 5F 5F 5F 5F
ADDR[23:0] 000400 000800 001000 002000 004000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 5F 5F 5F 5F 5F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 43500 43600 43700 43800 43900 4400
CLOCK 436 437 438 439 440 44
RESTART
STATE[7:0] 5F 5F 5F 5F 5F
ADDR[23:0] 008000 010000 020000 040000 080000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 5F 5F 5F 5F 5F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

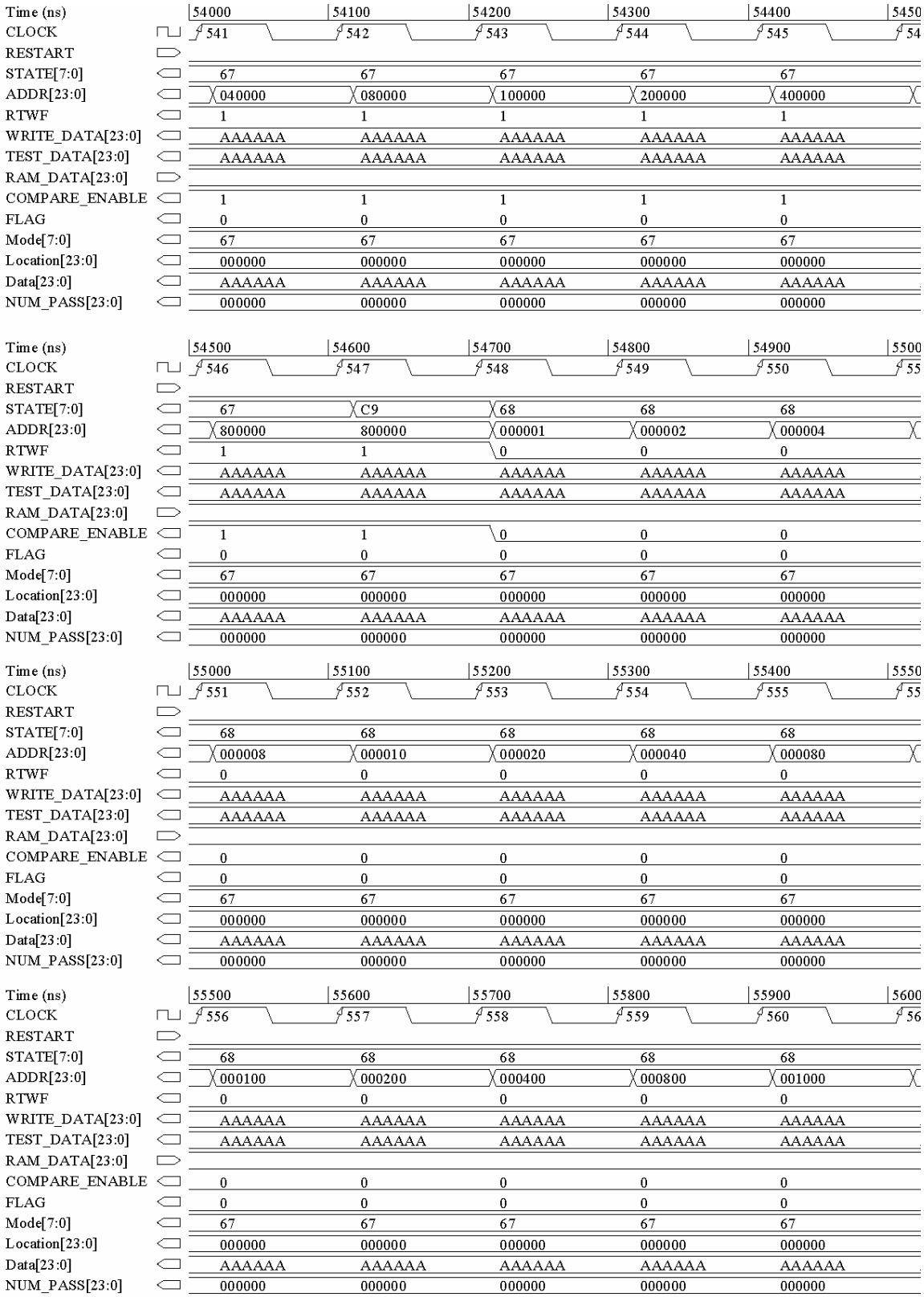

Time (ns)
44000
44100
44200
44300
44400
4450
CLOCK
441
442
443
444
445
44
RESTART
STATE[7:0]
5F
5F
5F
5F
C7
ADDR[23:0]
100000
200000
400000
800000
800000
RTWF
1
1
1
1
1
WRITE_DATA[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
TEST_DATA[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE
1
1
1
1
1
FLAG
0
0
0
0
0
Mode[7:0]
5F
5F
5F
5F
5F
Location[23:0]
000000
000000
000000
000000
000000
Data[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
NUM_PASS[23:0]
000000
000000
000000
000000
000000
Time (ns)
44500
44600
44700
44800
44900
4500
CLOCK
446
447
448
449
450
45
RESTART
STATE[7:0]
60
60
60
60
60
ADDR[23:0]
000001
000002
000004
000008
000010
RTWF
0
0
0
0
0
WRITE_DATA[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
TEST_DATA[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE
0
0
0
0
0
FLAG
0
0
0
0
0
Mode[7:0]
5F
5F
5F
5F
5F
Location[23:0]
000000
000000
000000
000000
000000
Data[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
NUM_PASS[23:0]
000000
000000
000000
000000
000000
Time (ns)
45000
45100
45200
45300
45400
4550
CLOCK
451
452
453
454
455
45
RESTART
STATE[7:0]
60
60
60
60
60
ADDR[23:0]
000020
000040
000080
000100
000200
RTWF
0
0
0
0
0
WRITE_DATA[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
TEST_DATA[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE
0
0
0
0
0
FLAG
0
0
0
0
0
Mode[7:0]
5F
5F
5F
5F
5F
Location[23:0]
000000
000000
000000
000000
000000
Data[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
NUM_PASS[23:0]
000000
000000
000000
000000
000000
Time (ns)
45500
45600
45700
45800
45900
4600
CLOCK
456
457
458
459
460
46
RESTART
STATE[7:0]
60
60
60
60
60
ADDR[23:0]
000400
000800
001000
002000
004000
RTWF
0
0
0
0
0
WRITE_DATA[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
TEST_DATA[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE
0
0
0
0
0
FLAG
0
0
0
0
0
Mode[7:0]
5F
5F
5F
5F
5F
Location[23:0]
000000
000000
000000
000000
000000
Data[23:0]
AAAAAA
AAAAAA
AAAAAA
AAAAAA
AAAAAA
NUM_PASS[23:0]
000000
000000
000000
000000
000000

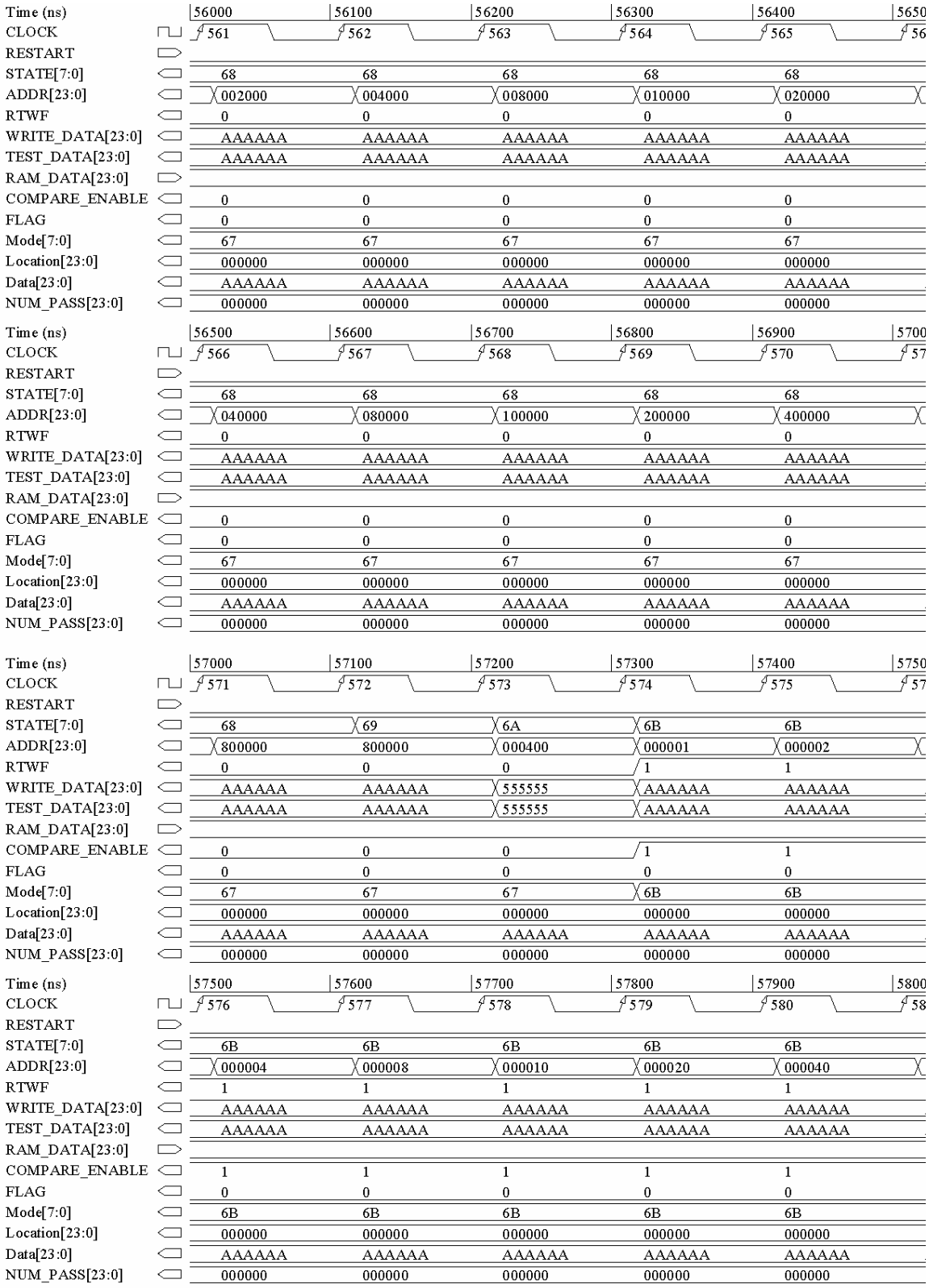

Time (ns) 46000 46100 46200 46300 46400 4650
CLOCK 461 462 463 464 465 46
RESTART
STATE[7:0] 60 60 60 60 60
ADDR[23:0] 008000 010000 020000 040000 080000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 5F 5F 5F 5F 5F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 46500 46600 46700 46800 46900 4700
CLOCK 466 467 468 469 470 47
RESTART
STATE[7:0] 60 60 60 60 61
ADDR[23:0] 100000 200000 400000 800000 800000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 5F 5F 5F 5F 5F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 47000 47100 47200 47300 47400 4750
CLOCK 471 472 473 474 475 47
RESTART
STATE[7:0] 62 63 63 63 63
ADDR[23:0] 000100 000001 000002 000004 000008
RTWF 0 1 1 1 1
WRITE_DATA[23:0] 555555 AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] 555555 AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 5F 63 63 63 63
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 47500 47600 47700 47800 47900 4800
CLOCK 476 477 478 479 480 48
RESTART
STATE[7:0] 63 63 63 63 63
ADDR[23:0] 000010 000020 000040 000080 000100
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 63 63 63 63 63
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

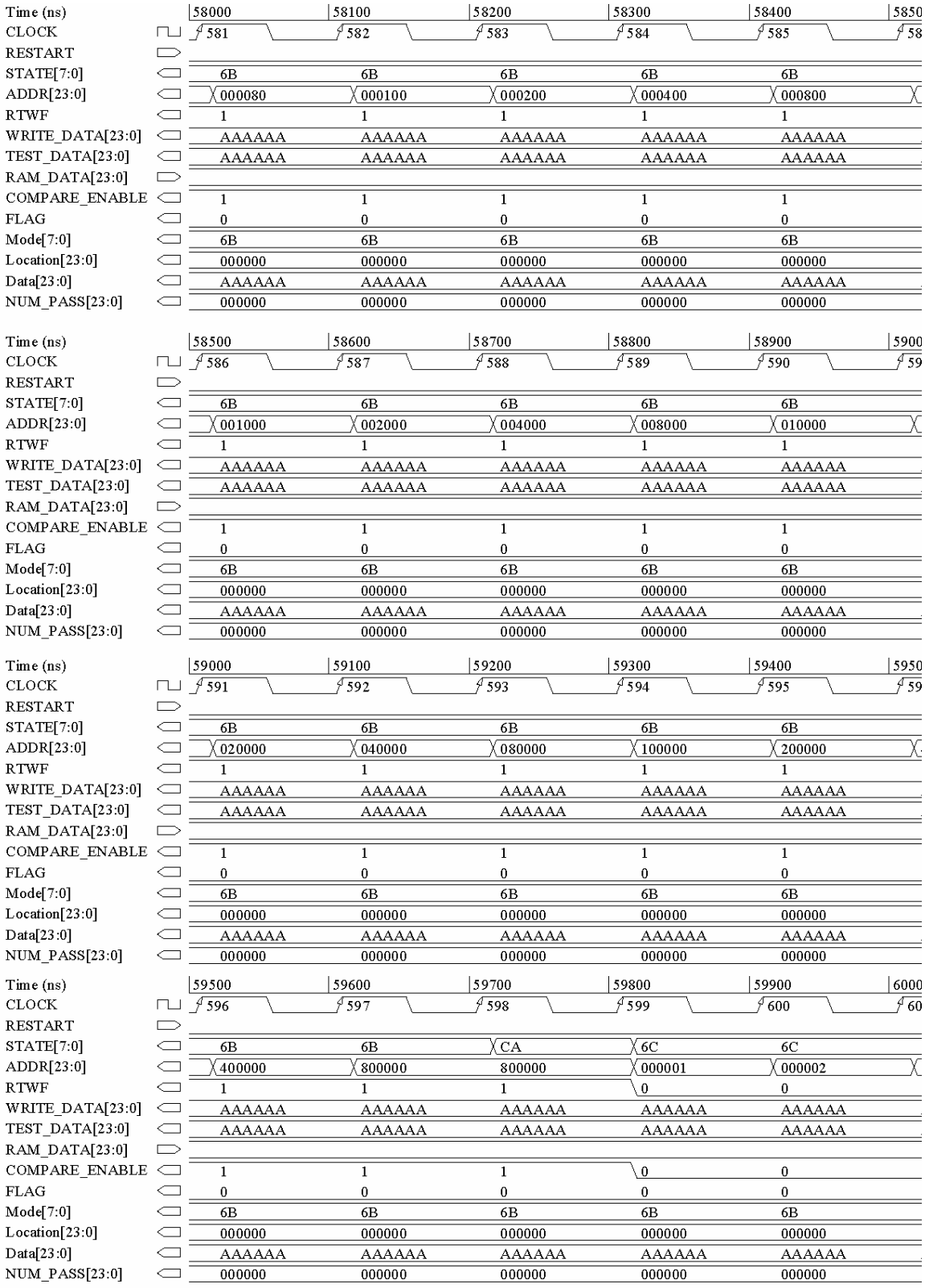

Time (ns) 48000 48100 48200 48300 48400 4850
CLOCK 481 482 483 484 485 48
RESTART
STATE[7:0] 63 63 63 63 63
ADDR[23:0] 000200 000400 000800 001000 002000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 63 63 63 63 63
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 48500 48600 48700 48800 48900 4900
CLOCK 486 487 488 489 490 49
RESTART
STATE[7:0] 63 63 63 63 63
ADDR[23:0] 004000 008000 010000 020000 040000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 63 63 63 63 63
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 49000 49100 49200 49300 49400 4950
CLOCK 491 492 493 494 495 49
RESTART
STATE[7:0] 63 63 63 63 63
ADDR[23:0] 080000 100000 200000 400000 800000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 63 63 63 63 63
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 49500 49600 49700 49800 49900 5000
CLOCK 496 497 498 499 500 50
RESTART
STATE[7:0] C8 64 64 64 64
ADDR[23:0] 800000 000001 000002 000004 000008
RTWF 1 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 63 63 63 63 63
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

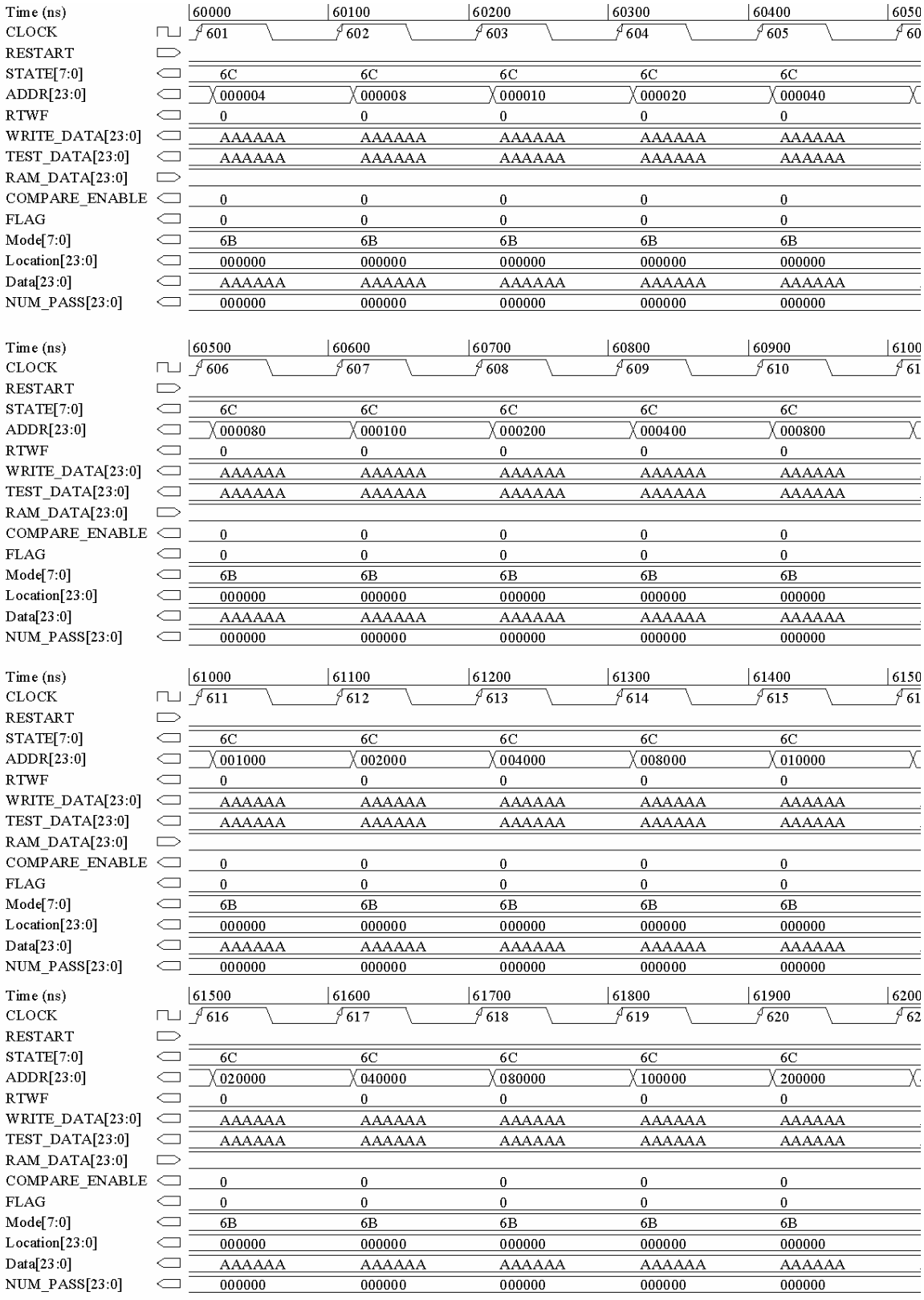

Time (ns) 50000 50100 50200 50300 50400 5050
CLOCK 501 502 503 504 505 50
RESTART
STATE[7:0] 64 64 64 64 64
ADDR[23:0] 000010 000020 000040 000080 000100
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 63 63 63 63 63
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 50500 50600 50700 50800 50900 5100
CLOCK 506 507 508 509 510 51
RESTART
STATE[7:0] 64 64 64 64 64
ADDR[23:0] 000200 000400 000800 001000 002000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 63 63 63 63 63
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 51000 51100 51200 51300 51400 5150
CLOCK 511 512 513 514 515 51
RESTART
STATE[7:0] 64 64 64 64 64
ADDR[23:0] 004000 008000 010000 020000 040000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 63 63 63 63 63
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 51500 51600 51700 51800 51900 5200
CLOCK 516 517 518 519 520 52
RESTART
STATE[7:0] 64 64 64 64 64
ADDR[23:0] 080000 100000 200000 400000 800000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 63 63 63 63 63
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

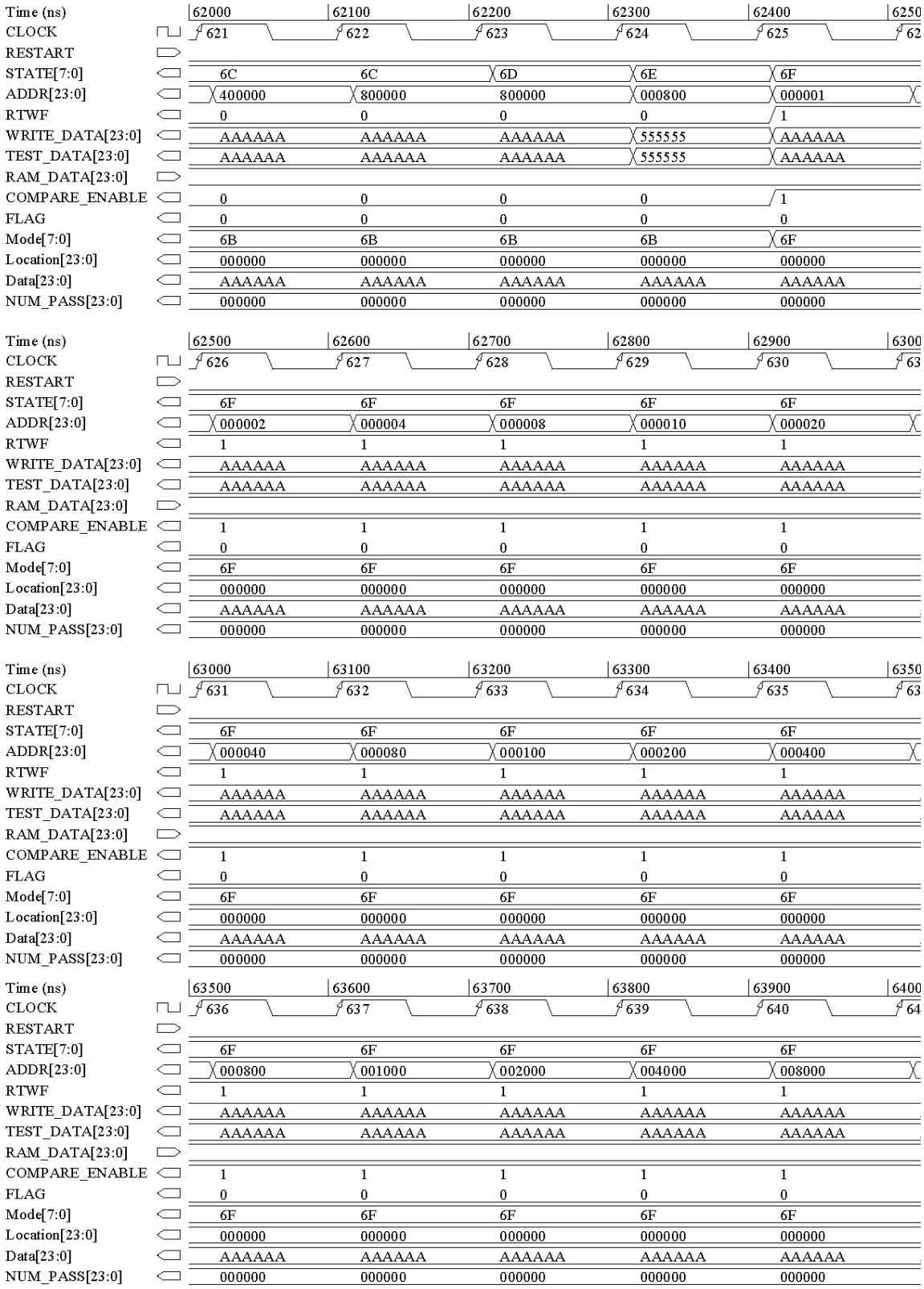

Time (ns) 52000 52100 52200 52300 52400 5250
CLOCK 521 522 523 524 525 52
RESTART
STATE[7:0] 65 66 67 67 67
ADDR[23:0] 800000 000200 000001 000002 000004
RTWF 0 0 1 1 1
WRITE_DATA[23:0] AAAAAA 555555 AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA 555555 AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 63 63 67 67 67
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 52500 52600 52700 52800 52900 5300
CLOCK 526 527 528 529 530 53
RESTART
STATE[7:0] 67 67 67 67 67
ADDR[23:0] 000008 000010 000020 000040 000080
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 67 67 67 67 67
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 53000 53100 53200 53300 53400 5350
CLOCK 531 532 533 534 535 53
RESTART
STATE[7:0] 67 67 67 67 67
ADDR[23:0] 000100 000200 000400 000800 001000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 67 67 67 67 67
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 53500 53600 53700 53800 53900 5400
CLOCK 536 537 538 539 540 54
RESTART
STATE[7:0] 67 67 67 67 67
ADDR[23:0] 002000 004000 008000 010000 020000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 67 67 67 67 67
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

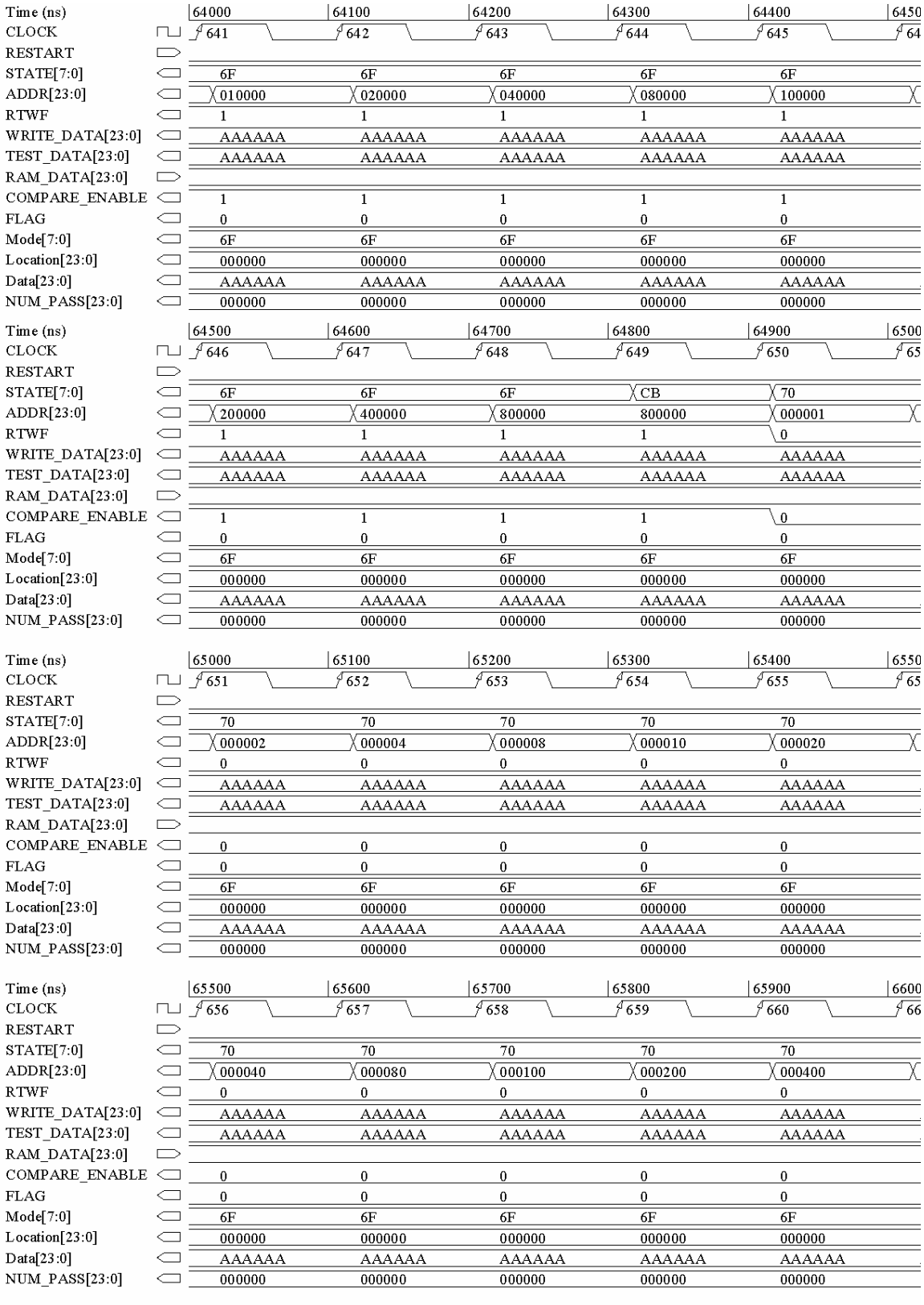

Time (ns) 54000 54100 54200 54300 54400 5450
CLOCK 541 542 543 544 545 54
RESTART
STATE[7:0] 67 67 67 67 67
ADDR[23:0] 040000 080000 100000 200000 400000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 67 67 67 67 67
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 54500 54600 54700 54800 54900 5500
CLOCK 546 547 548 549 550 55
RESTART
STATE[7:0] 67 C9 68 68 68
ADDR[23:0] 800000 800000 000001 000002 000004
RTWF 1 1 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 67 67 67 67 67
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 55000 55100 55200 55300 55400 5550
CLOCK 551 552 553 554 555 55
RESTART
STATE[7:0] 68 68 68 68 68
ADDR[23:0] 000008 000010 000020 000040 000080
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 67 67 67 67 67
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 55500 55600 55700 55800 55900 5600
CLOCK 556 557 558 559 560 56
RESTART
STATE[7:0] 68 68 68 68 68
ADDR[23:0] 000100 000200 000400 000800 001000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 67 67 67 67 67
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

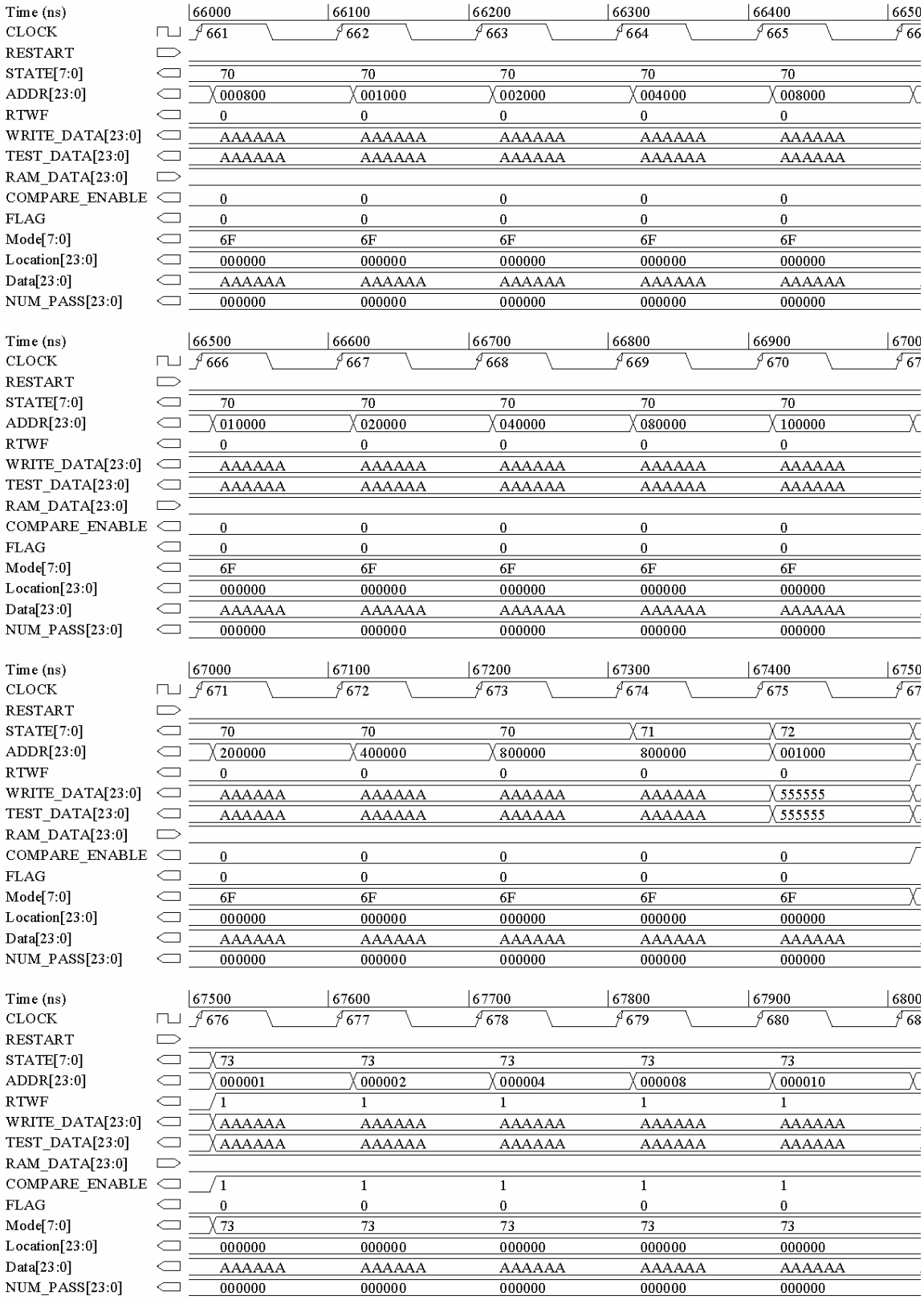

Time (ns) 56000 56100 56200 56300 56400 5650
CLOCK 561 562 563 564 565 56
RESTART
STATE[7:0] 68 68 68 68 68
ADDR[23:0] 002000 004000 008000 010000 020000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 67 67 67 67 67
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 56500 56600 56700 56800 56900 5700
CLOCK 566 567 568 569 570 57
RESTART
STATE[7:0] 68 68 68 68 68
ADDR[23:0] 040000 080000 100000 200000 400000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 67 67 67 67 67
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 57000 57100 57200 57300 57400 5750
CLOCK 571 572 573 574 575 57
RESTART
STATE[7:0] 68 69 6A 6B 6B
ADDR[23:0] 800000 800000 000400 000001 000002
RTWF 0 0 0 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA 555555 AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA 555555 AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 1 1
FLAG 0 0 0 0 0
Mode[7:0] 67 67 67 6B 6B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 57500 57600 57700 57800 57900 5800
CLOCK 576 577 578 579 580 58
RESTART
STATE[7:0] 6B 6B 6B 6B 6B
ADDR[23:0] 000004 000008 000010 000020 000040
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 6B 6B 6B 6B 6B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

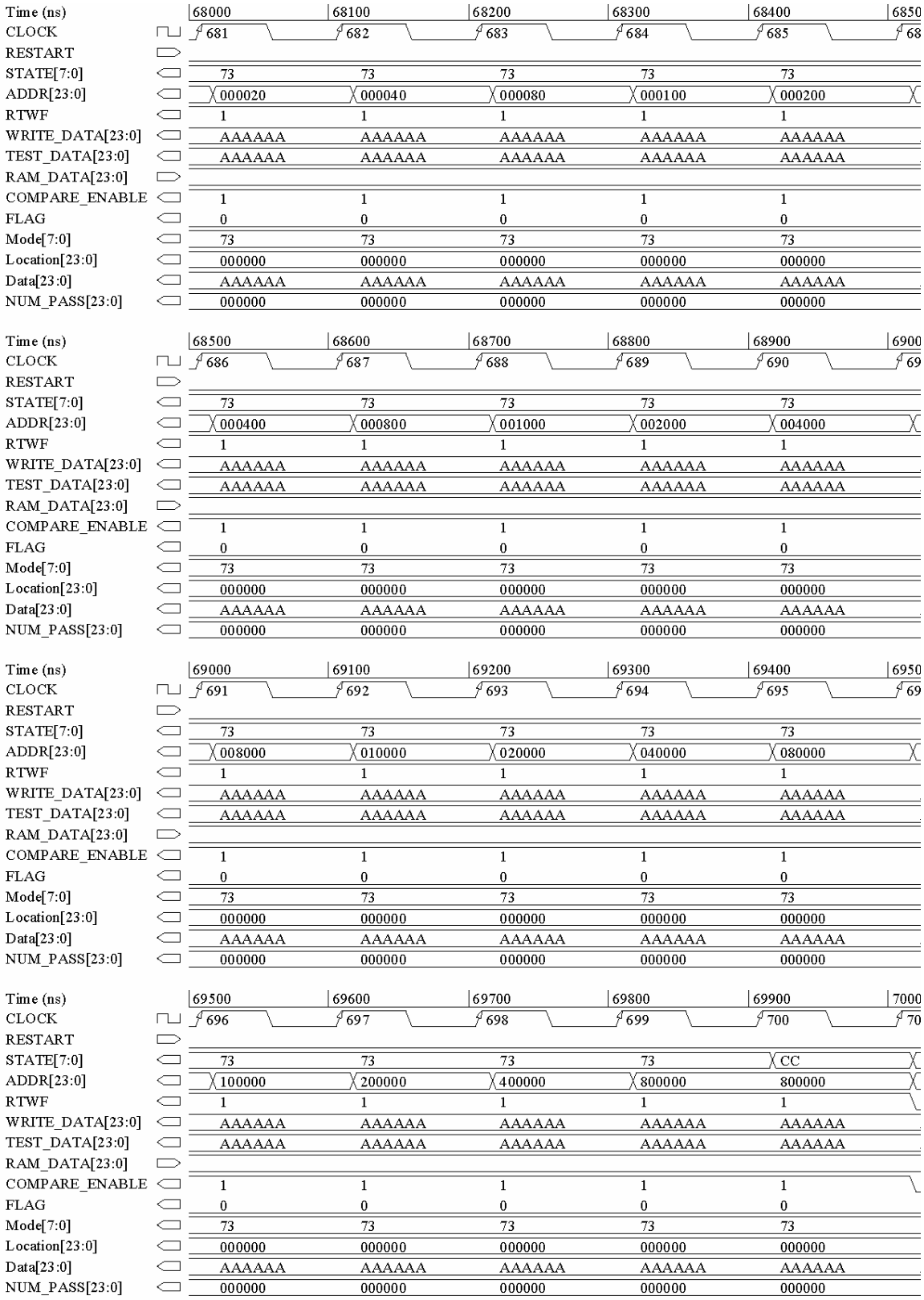

Time (ns) 58000 58100 58200 58300 58400 5850
CLOCK 581 582 583 584 585 58
RESTART
STATE[7:0] 6B 6B 6B 6B 6B
ADDR[23:0] 000080 000100 000200 000400 000800
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 6B 6B 6B 6B 6B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 58500 58600 58700 58800 58900 5900
CLOCK 586 587 588 589 590 59
RESTART
STATE[7:0] 6B 6B 6B 6B 6B
ADDR[23:0] 001000 002000 004000 008000 010000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 6B 6B 6B 6B 6B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 59000 59100 59200 59300 59400 5950
CLOCK 591 592 593 594 595 59
RESTART
STATE[7:0] 6B 6B 6B 6B 6B
ADDR[23:0] 020000 040000 080000 100000 200000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 6B 6B 6B 6B 6B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 59500 59600 59700 59800 59900 6000
CLOCK 596 597 598 599 600 60
RESTART
STATE[7:0] 6B 6B CA 6C 6C
ADDR[23:0] 400000 800000 800000 000001 000002
RTWF 1 1 1 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 0 0
FLAG 0 0 0 0 0
Mode[7:0] 6B 6B 6B 6B 6B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

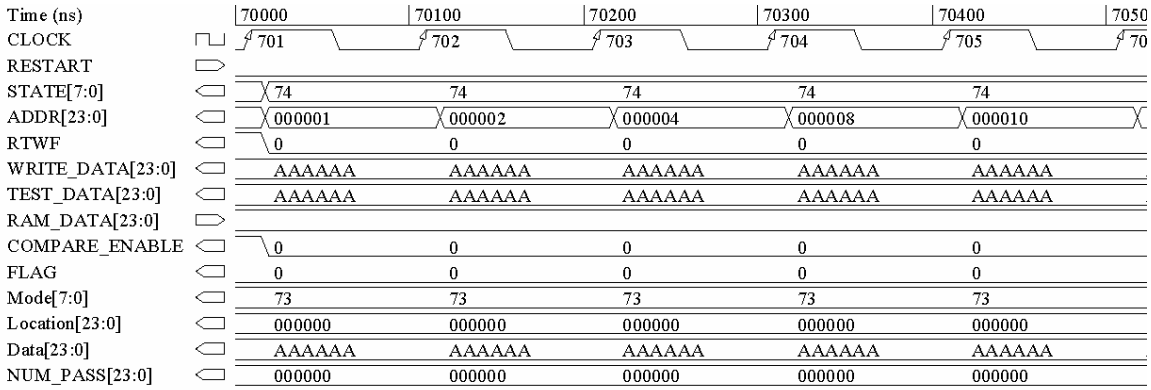

Time (ns) 60000 60100 60200 60300 60400 6050
CLOCK 601 602 603 604 605 60
RESTART
STATE[7:0] 6C 6C 6C 6C 6C
ADDR[23:0] 000004 000008 000010 000020 000040
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 6B 6B 6B 6B 6B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 60500 60600 60700 60800 60900 6100
CLOCK 606 607 608 609 610 61
RESTART
STATE[7:0] 6C 6C 6C 6C 6C
ADDR[23:0] 000080 000100 000200 000400 000800
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 6B 6B 6B 6B 6B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 61000 61100 61200 61300 61400 6150
CLOCK 611 612 613 614 615 61
RESTART
STATE[7:0] 6C 6C 6C 6C 6C
ADDR[23:0] 001000 002000 004000 008000 010000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 6B 6B 6B 6B 6B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 61500 61600 61700 61800 61900 6200
CLOCK 616 617 618 619 620 62
RESTART
STATE[7:0] 6C 6C 6C 6C 6C
ADDR[23:0] 020000 040000 080000 100000 200000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 6B 6B 6B 6B 6B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

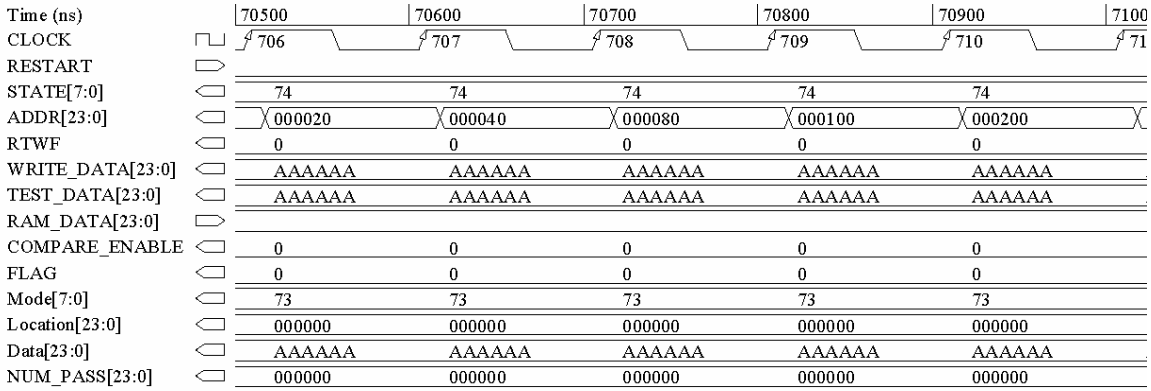

Time (ns) 62000 62100 62200 62300 62400 6250
CLOCK 621 622 623 624 625 62
RESTART
STATE[7:0] 6C 6C 6D 6E 6F
ADDR[23:0] 400000 800000 800000 000800 000001
RTWF 0 0 0 0 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA 555555 AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA 555555 AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 1
FLAG 0 0 0 0 0
Mode[7:0] 6B 6B 6B 6B 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 62500 62600 62700 62800 62900 6300
CLOCK 626 627 628 629 630 63
RESTART
STATE[7:0] 6F 6F 6F 6F 6F
ADDR[23:0] 000002 000004 000008 000010 000020
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 6F 6F 6F 6F 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 63000 63100 63200 63300 63400 6350
CLOCK 631 632 633 634 635 63
RESTART
STATE[7:0] 6F 6F 6F 6F 6F
ADDR[23:0] 000040 000080 000100 000200 000400
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 6F 6F 6F 6F 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 63500 63600 63700 63800 63900 6400
CLOCK 636 637 638 639 640 64
RESTART
STATE[7:0] 6F 6F 6F 6F 6F
ADDR[23:0] 000800 001000 002000 004000 008000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 6F 6F 6F 6F 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

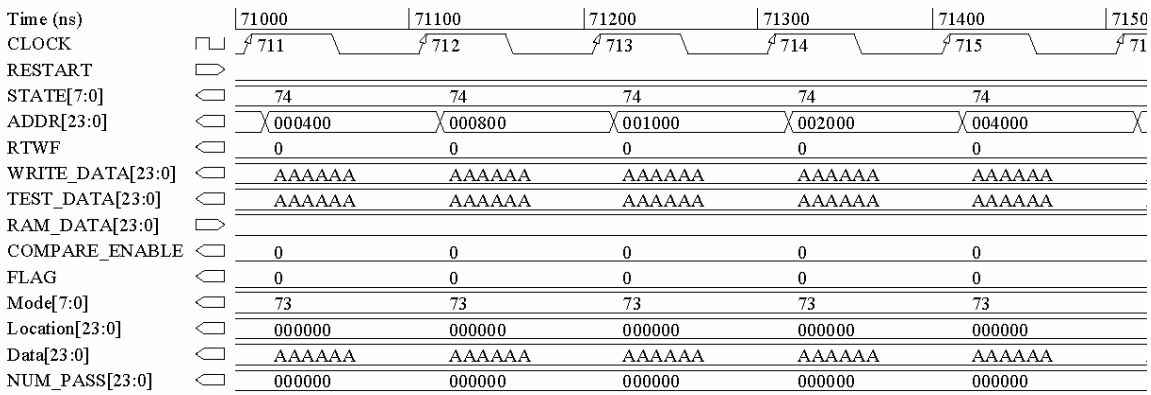

Time (ns) 64000 64100 64200 64300 64400 6450
CLOCK 641 642 643 644 645 64
RESTART
STATE[7:0] 6F 6F 6F 6F 6F
ADDR[23:0] 010000 020000 040000 080000 100000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 6F 6F 6F 6F 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 64500 64600 64700 64800 64900 6500
CLOCK 646 647 648 649 650 65
RESTART
STATE[7:0] 6F 6F 6F CB 70
ADDR[23:0] 200000 400000 800000 800000 000001
RTWF 1 1 1 1 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 0
FLAG 0 0 0 0 0
Mode[7:0] 6F 6F 6F 6F 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 65000 65100 65200 65300 65400 6550
CLOCK 651 652 653 654 655 65
RESTART
STATE[7:0] 70 70 70 70 70
ADDR[23:0] 000002 000004 000008 000010 000020
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 6F 6F 6F 6F 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 65500 65600 65700 65800 65900 6600
CLOCK 656 657 658 659 660 66
RESTART
STATE[7:0] 70 70 70 70 70
ADDR[23:0] 000040 000080 000100 000200 000400
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 6F 6F 6F 6F 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

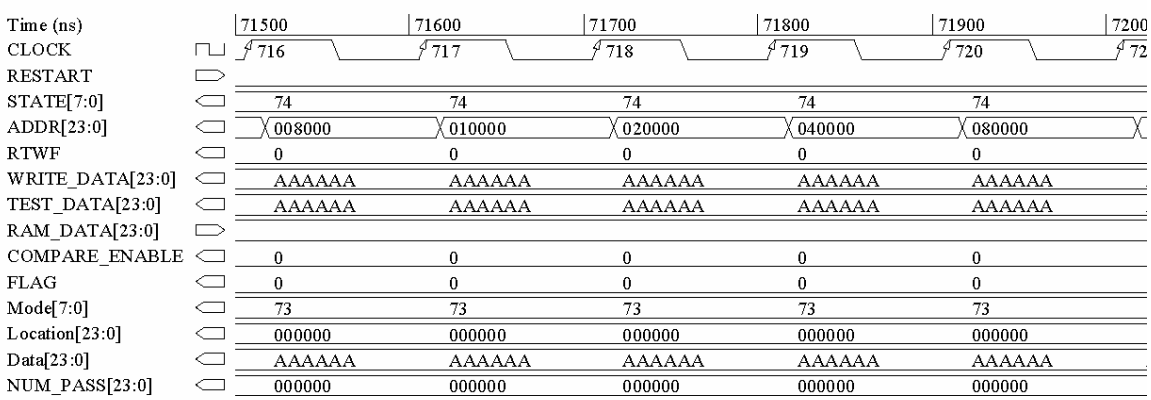

Time (ns) 66000 66100 66200 66300 66400 6650
CLOCK 661 662 663 664 665 66
RESTART
STATE[7:0] 70 70 70 70 70
ADDR[23:0] 000800 001000 002000 004000 008000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 6F 6F 6F 6F 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 66500 66600 66700 66800 66900 6700
CLOCK 666 667 668 669 670 67
RESTART
STATE[7:0] 70 70 70 70 70
ADDR[23:0] 010000 020000 040000 080000 100000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 6F 6F 6F 6F 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 67000 67100 67200 67300 67400 6750
CLOCK 671 672 673 674 675 67
RESTART
STATE[7:0] 70 70 70 71 72
ADDR[23:0] 200000 400000 800000 800000 001000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA 555555
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA 555555
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 6F 6F 6F 6F 6F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 67500 67600 67700 67800 67900 6800
CLOCK 676 677 678 679 680 68
RESTART
STATE[7:0] 73 73 73 73 73
ADDR[23:0] 000001 000002 000004 000008 000010
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 73 73 73 73 73
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

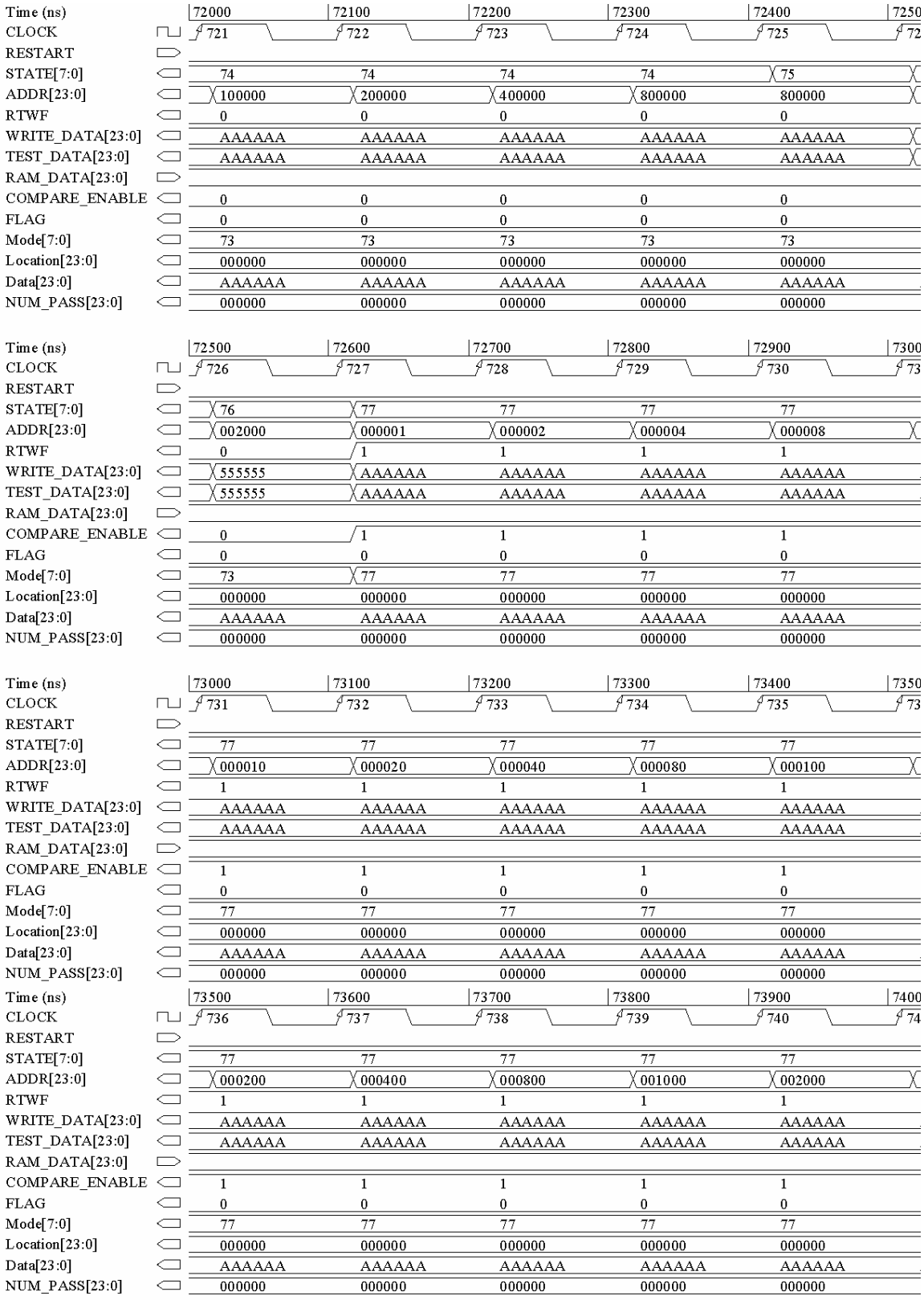

Time (ns) 68000 68100 68200 68300 68400 6850
CLOCK 681 682 683 684 685 68
RESTART
STATE[7:0] 73 73 73 73 73
ADDR[23:0] 000020 000040 000080 000100 000200
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 73 73 73 73 73
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 68500 68600 68700 68800 68900 6900
CLOCK 686 687 688 689 690 69
RESTART
STATE[7:0] 73 73 73 73 73
ADDR[23:0] 000400 000800 001000 002000 004000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 73 73 73 73 73
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 69000 69100 69200 69300 69400 6950
CLOCK 691 692 693 694 695 69
RESTART
STATE[7:0] 73 73 73 73 73
ADDR[23:0] 008000 010000 020000 040000 080000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 73 73 73 73 73
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 69500 69600 69700 69800 69900 7000
CLOCK 696 697 698 699 700 70
RESTART
STATE[7:0] 73 73 73 73 CC
ADDR[23:0] 100000 200000 400000 800000 800000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 73 73 73 73 73
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

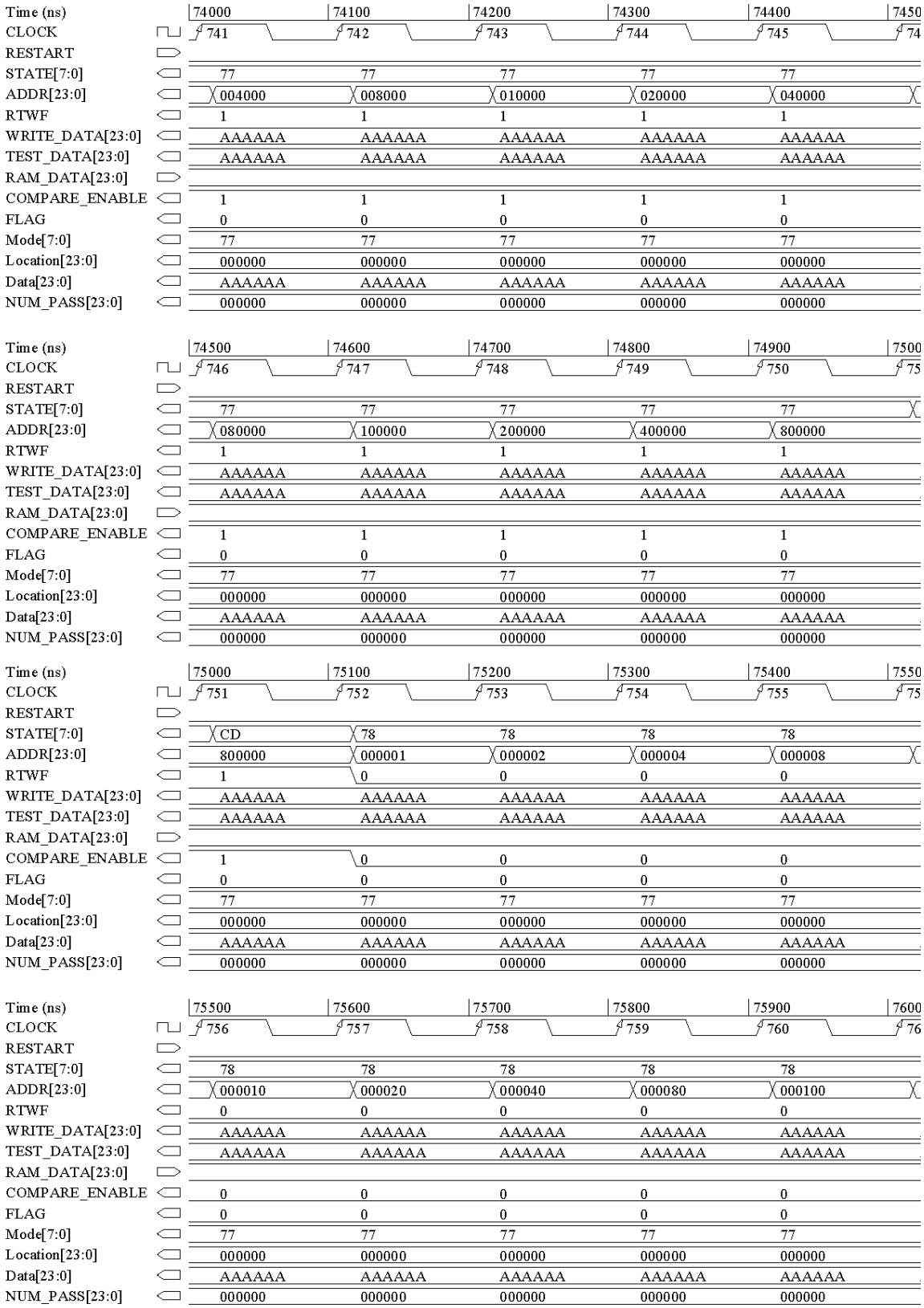

Time (ns) 70000 70100 70200 70300 70400 7050
CLOCK 701 702 703 704 705 70
RESTART
STATE[7:0] 74 74 74 74 74
ADDR[23:0] 000001 000002 000004 000008 000010
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 73 73 73 73 73
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

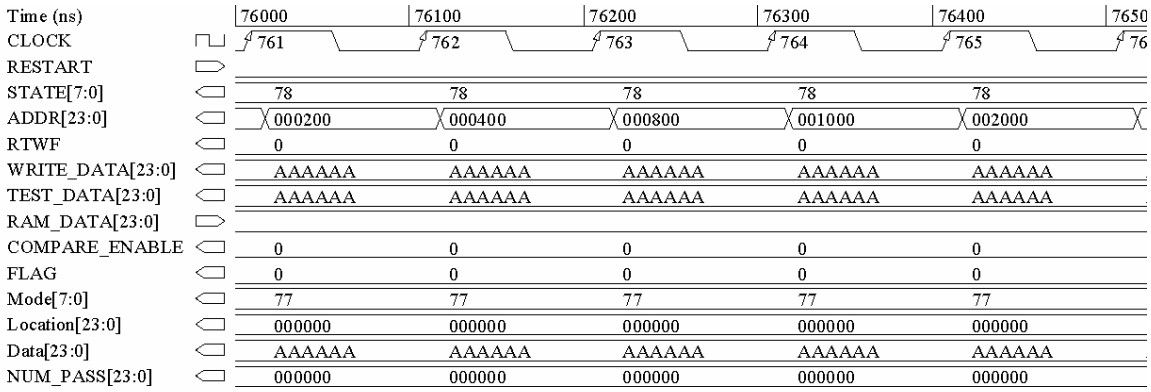

Time (ns) 70500 70600 70700 70800 70900 7100
CLOCK 706 707 708 709 710 71
RESTART
STATE[7:0] 74 74 74 74 74
ADDR[23:0] 000020 000040 000080 000100 000200
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 73 73 73 73 73
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

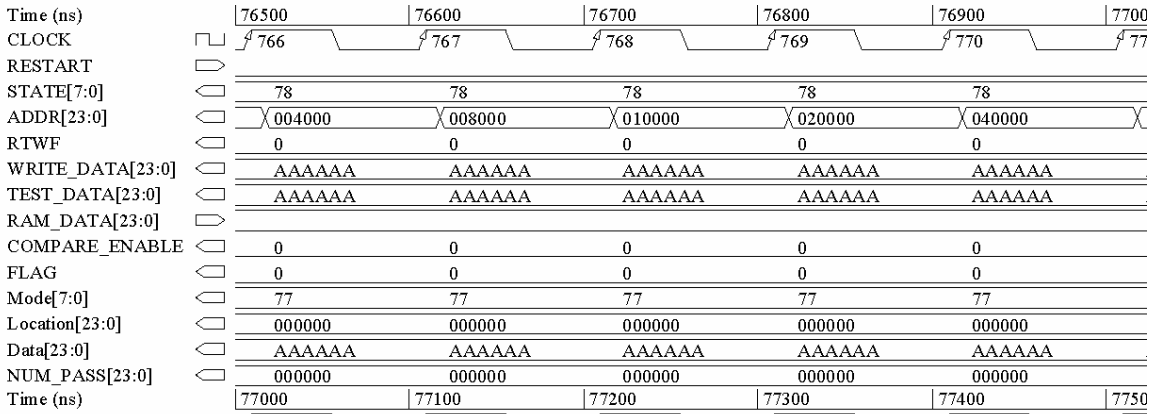

Time (ns) 71000 71100 71200 71300 71400 7150
CLOCK 711 712 713 714 715 71
RESTART
STATE[7:0] 74 74 74 74 74
ADDR[23:0] 000400 000800 001000 002000 004000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 73 73 73 73 73
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

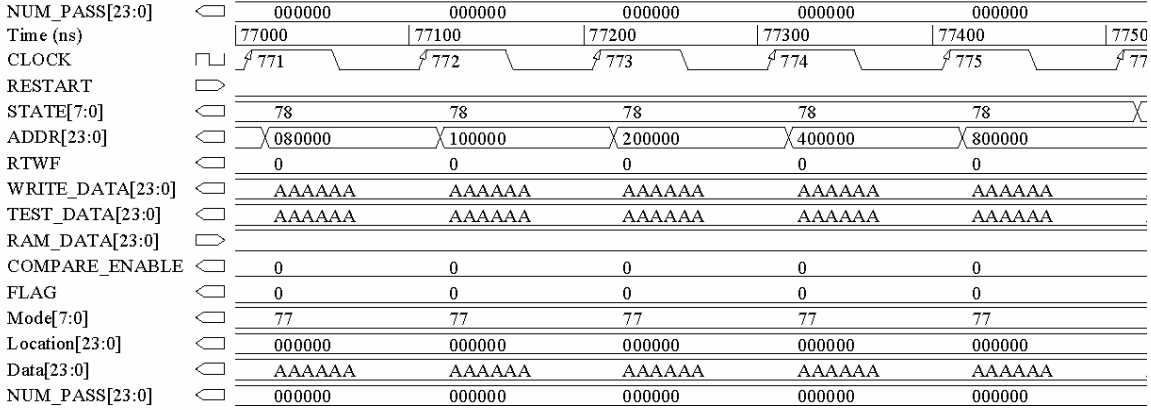

Time (ns) 71500 71600 71700 71800 71900 7200
CLOCK 716 717 718 719 720 72
RESTART
STATE[7:0] 74 74 74 74 74
ADDR[23:0] 008000 010000 020000 040000 080000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 73 73 73 73 73
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

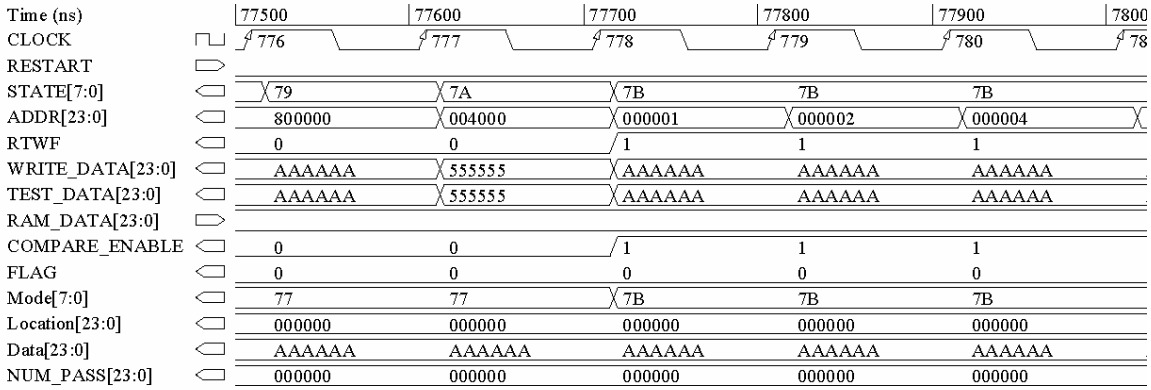

Time (ns) 72000 72100 72200 72300 72400 7250
CLOCK 721 722 723 724 725 72
RESTART
STATE[7:0] 74 74 74 74 75
ADDR[23:0] 100000 200000 400000 800000 800000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 73 73 73 73 73
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 72500 72600 72700 72800 72900 7300
CLOCK 726 727 728 729 730 73
RESTART
STATE[7:0] 76 77 77 77 77
ADDR[23:0] 002000 000001 000002 000004 000008
RTWF 0 1 1 1 1
WRITE_DATA[23:0] 555555 AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] 555555 AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 73 77 77 77 77
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 73000 73100 73200 73300 73400 7350
CLOCK 731 732 733 734 735 73
RESTART
STATE[7:0] 77 77 77 77 77
ADDR[23:0] 000010 000020 000040 000080 000100
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 77 77 77 77 77
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 73500 73600 73700 73800 73900 7400
CLOCK 736 737 738 739 740 74
RESTART
STATE[7:0] 77 77 77 77 77
ADDR[23:0] 000200 000400 000800 001000 002000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 77 77 77 77 77
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

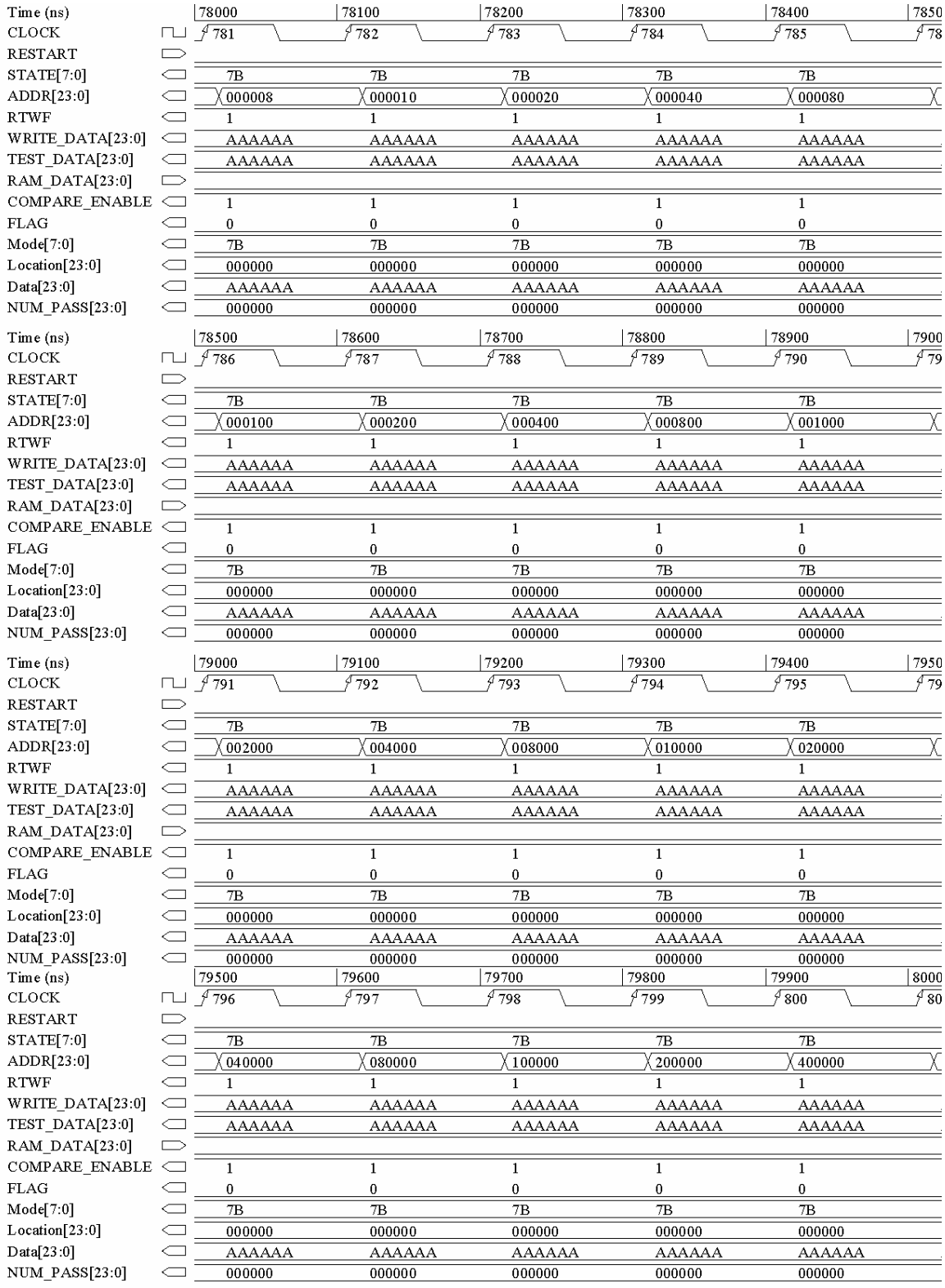

Time (ns) 74000 74100 74200 74300 74400 7450
CLOCK 741 742 743 744 745 74
RESTART
STATE[7:0] 77 77 77 77 77
ADDR[23:0] 004000 008000 010000 020000 040000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 77 77 77 77 77
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 74500 74600 74700 74800 74900 7500
CLOCK 746 747 748 749 750 75
RESTART
STATE[7:0] 77 77 77 77 77
ADDR[23:0] 080000 100000 200000 400000 800000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 77 77 77 77 77
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 75000 75100 75200 75300 75400 7550
CLOCK 751 752 753 754 755 75
RESTART
STATE[7:0] CD 78 78 78 78
ADDR[23:0] 800000 000001 000002 000004 000008
RTWF 1 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 77 77 77 77 77
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 75500 75600 75700 75800 75900 7600
CLOCK 756 757 758 759 760 76
RESTART
STATE[7:0] 78 78 78 78 78
ADDR[23:0] 000010 000020 000040 000080 000100
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 77 77 77 77 77
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

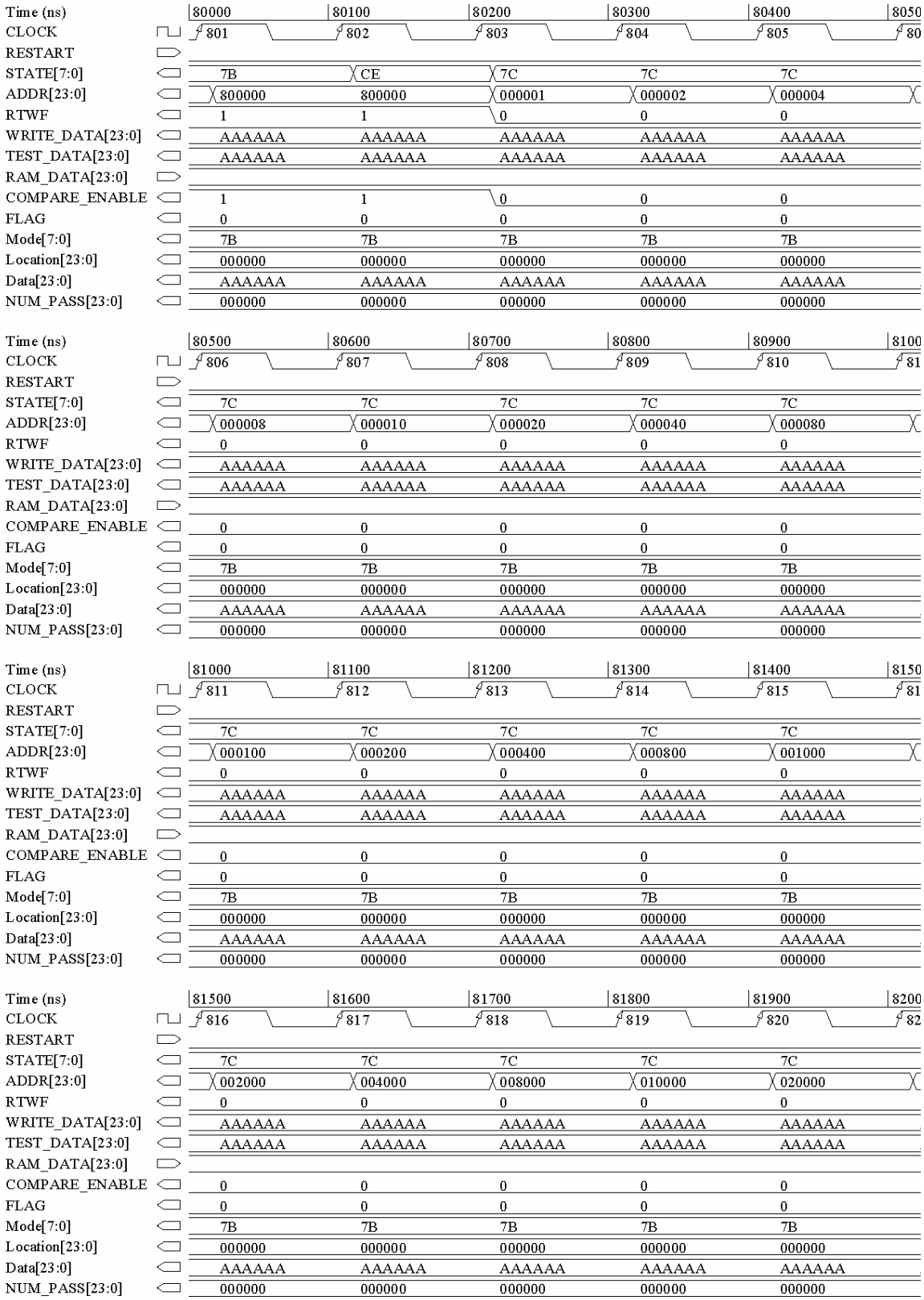

Time (ns) 76000 76100 76200 76300 76400 7650
CLOCK 761 762 763 764 765 76
RESTART
STATE[7:0] 78 78 78 78 78
ADDR[23:0] 000200 000400 000800 001000 002000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 77 77 77 77 77
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

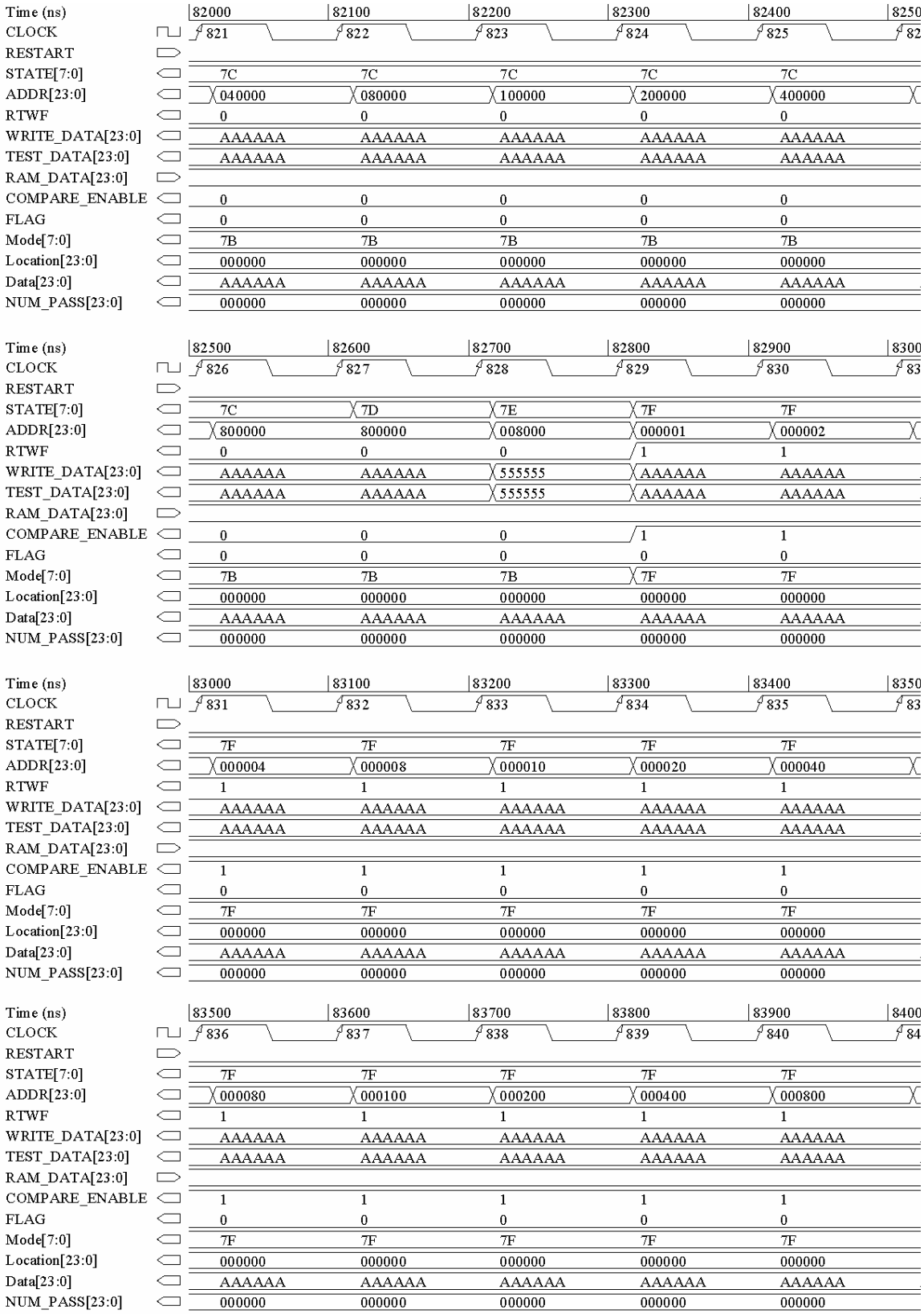

Time (ns) 76500 76600 76700 76800 76900 7700
CLOCK 766 767 768 769 770 77
RESTART
STATE[7:0] 78 78 78 78 78
ADDR[23:0] 004000 008000 010000 020000 040000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 77 77 77 77 77
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

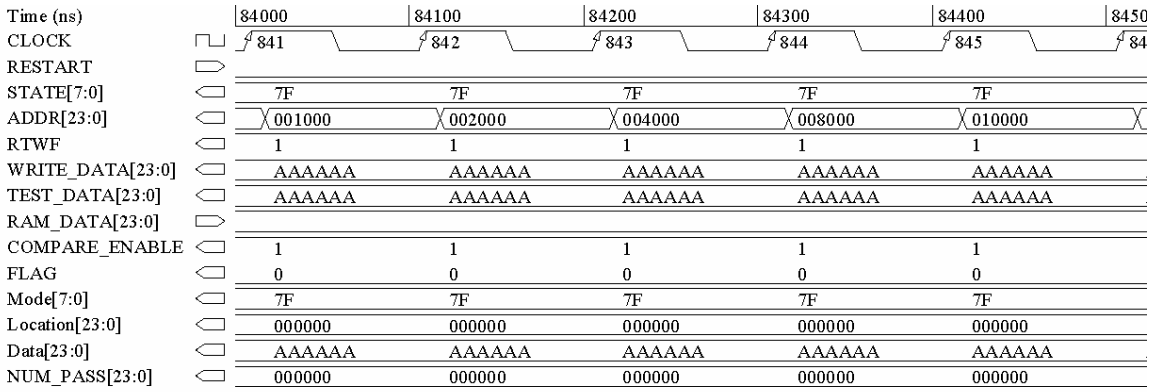

Time (ns) 77000 77100 77200 77300 77400 7750
CLOCK 771 772 773 774 775 77
RESTART
STATE[7:0] 78 78 78 78 78
ADDR[23:0] 080000 100000 200000 400000 800000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 77 77 77 77 77
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

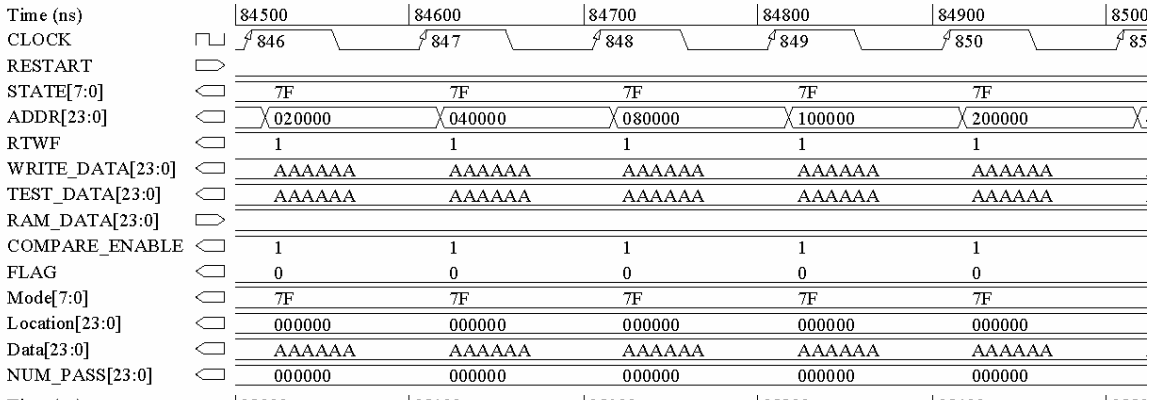

Time (ns) 77500 77600 77700 77800 77900 7800
CLOCK 776 777 778 779 780 78
RESTART
STATE[7:0] 79 7A 7B 7B 7B
ADDR[23:0] 800000 004000 000001 000002 000004
RTWF 0 0 1 1 1
WRITE_DATA[23:0] AAAAAA 555555 AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA 555555 AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 77 77 7B 7B 7B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

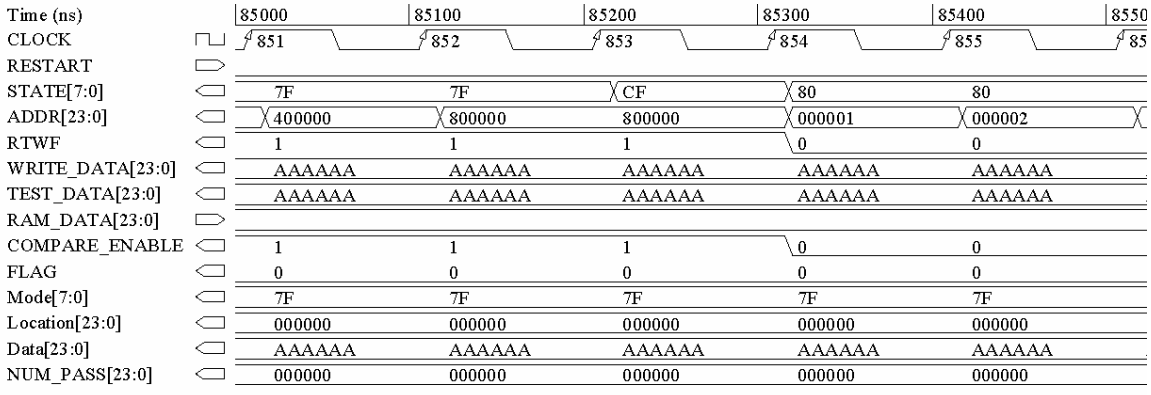

Time (ns) 78000 78100 78200 78300 78400 7850
CLOCK 781 782 783 784 785 78
RESTART
STATE[7:0] 7B 7B 7B 7B 7B
ADDR[23:0] 000008 000010 000020 000040 000080
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 7B 7B 7B 7B 7B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 78500 78600 78700 78800 78900 7900
CLOCK 786 787 788 789 790 79
RESTART
STATE[7:0] 7B 7B 7B 7B 7B
ADDR[23:0] 000100 000200 000400 000800 001000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 7B 7B 7B 7B 7B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 79000 79100 79200 79300 79400 7950
CLOCK 791 792 793 794 795 79
RESTART
STATE[7:0] 7B 7B 7B 7B 7B
ADDR[23:0] 002000 004000 008000 010000 020000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 7B 7B 7B 7B 7B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 79500 79600 79700 79800 79900 8000
CLOCK 796 797 798 799 800 80
RESTART
STATE[7:0] 7B 7B 7B 7B 7B
ADDR[23:0] 040000 080000 100000 200000 400000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 7B 7B 7B 7B 7B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

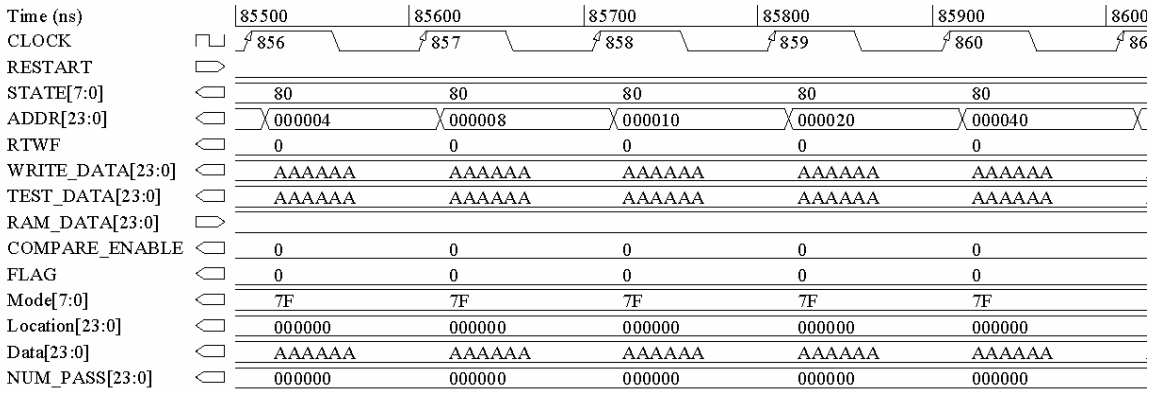

Time (ns) 80000 80100 80200 80300 80400 8050
CLOCK 801 802 803 804 805 80
RESTART
STATE[7:0] 7B CE 7C 7C 7C
ADDR[23:0] 800000 800000 000001 000002 000004
RTWF 1 1 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 7B 7B 7B 7B 7B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 80500 80600 80700 80800 80900 8100
CLOCK 806 807 808 809 810 81
RESTART
STATE[7:0] 7C 7C 7C 7C 7C
ADDR[23:0] 000008 000010 000020 000040 000080
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 7B 7B 7B 7B 7B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 81000 81100 81200 81300 81400 8150
CLOCK 811 812 813 814 815 81
RESTART
STATE[7:0] 7C 7C 7C 7C 7C
ADDR[23:0] 000100 000200 000400 000800 001000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 7B 7B 7B 7B 7B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 81500 81600 81700 81800 81900 8200
CLOCK 816 817 818 819 820 82
RESTART
STATE[7:0] 7C 7C 7C 7C 7C
ADDR[23:0] 002000 004000 008000 010000 020000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 7B 7B 7B 7B 7B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

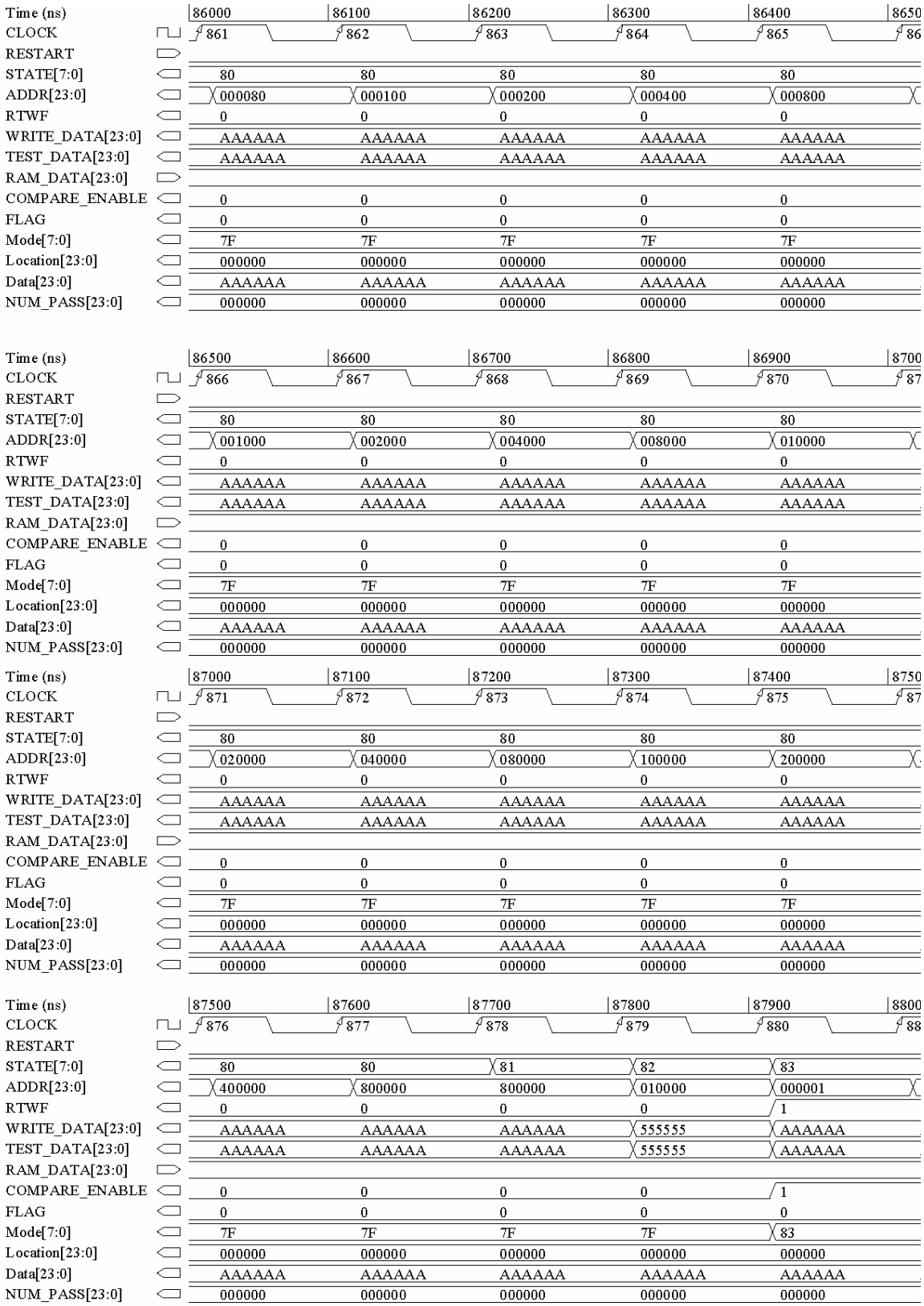

Time (ns) 82000 82100 82200 82300 82400 8250
CLOCK 821 822 823 824 825 82
RESTART
STATE[7:0] 7C 7C 7C 7C 7C
ADDR[23:0] 040000 080000 100000 200000 400000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 7B 7B 7B 7B 7B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 82500 82600 82700 82800 82900 8300
CLOCK 826 827 828 829 830 83
RESTART
STATE[7:0] 7C 7D 7E 7F 7F
ADDR[23:0] 800000 800000 008000 000001 000002
RTWF 0 0 0 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA 555555 AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA 555555 AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 1 1
FLAG 0 0 0 0 0
Mode[7:0] 7B 7B 7B 7F 7F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 83000 83100 83200 83300 83400 8350
CLOCK 831 832 833 834 835 83
RESTART
STATE[7:0] 7F 7F 7F 7F 7F
ADDR[23:0] 000004 000008 000010 000020 000040
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 7F 7F 7F 7F 7F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 83500 83600 83700 83800 83900 8400
CLOCK 836 837 838 839 840 84
RESTART
STATE[7:0] 7F 7F 7F 7F 7F
ADDR[23:0] 000080 000100 000200 000400 000800
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 7F 7F 7F 7F 7F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

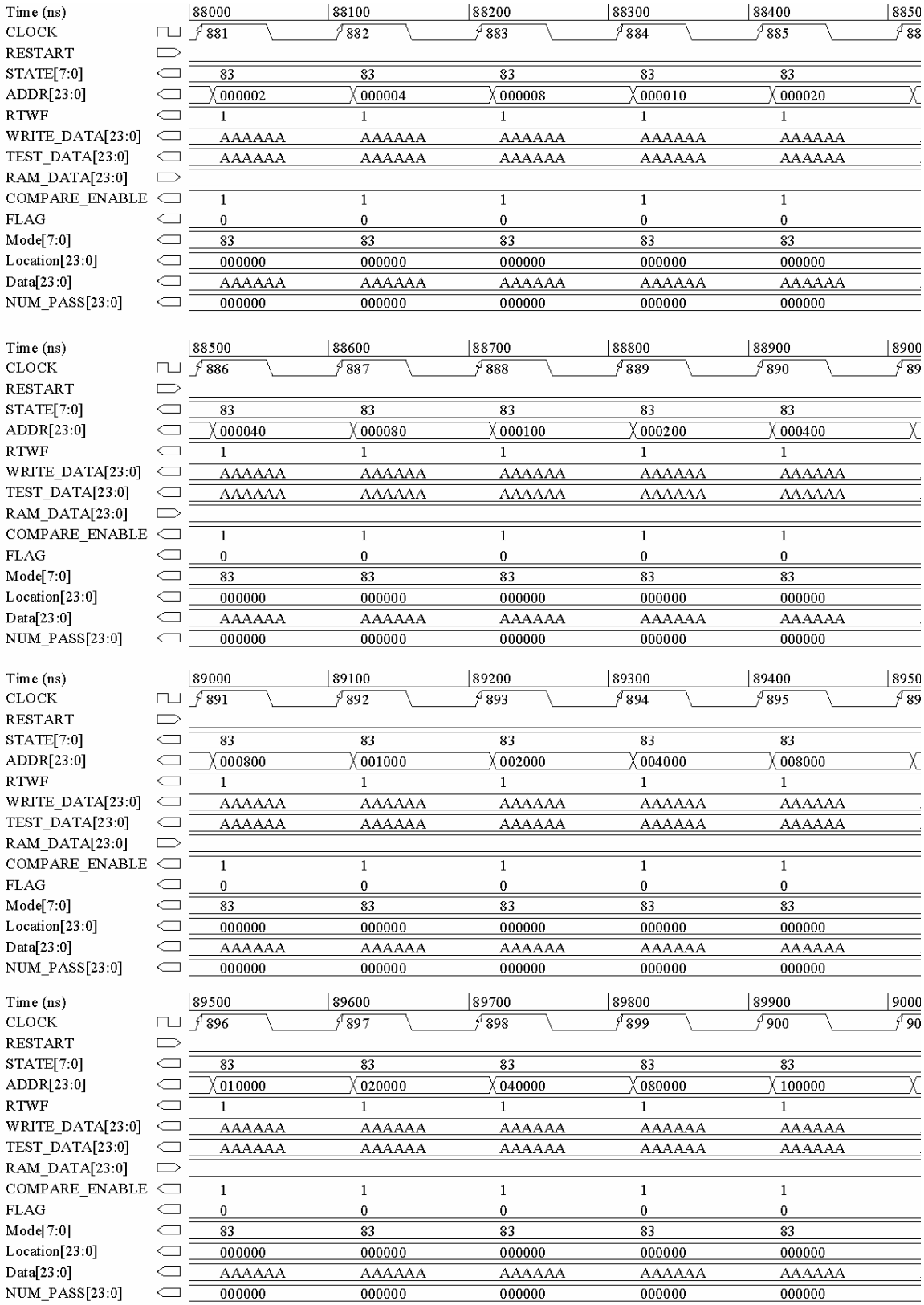

Time (ns) 84000 84100 84200 84300 84400 8450
CLOCK 841 842 843 844 845 84
RESTART
STATE[7:0] 7F 7F 7F 7F 7F
ADDR[23:0] 001000 002000 004000 008000 010000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 7F 7F 7F 7F 7F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

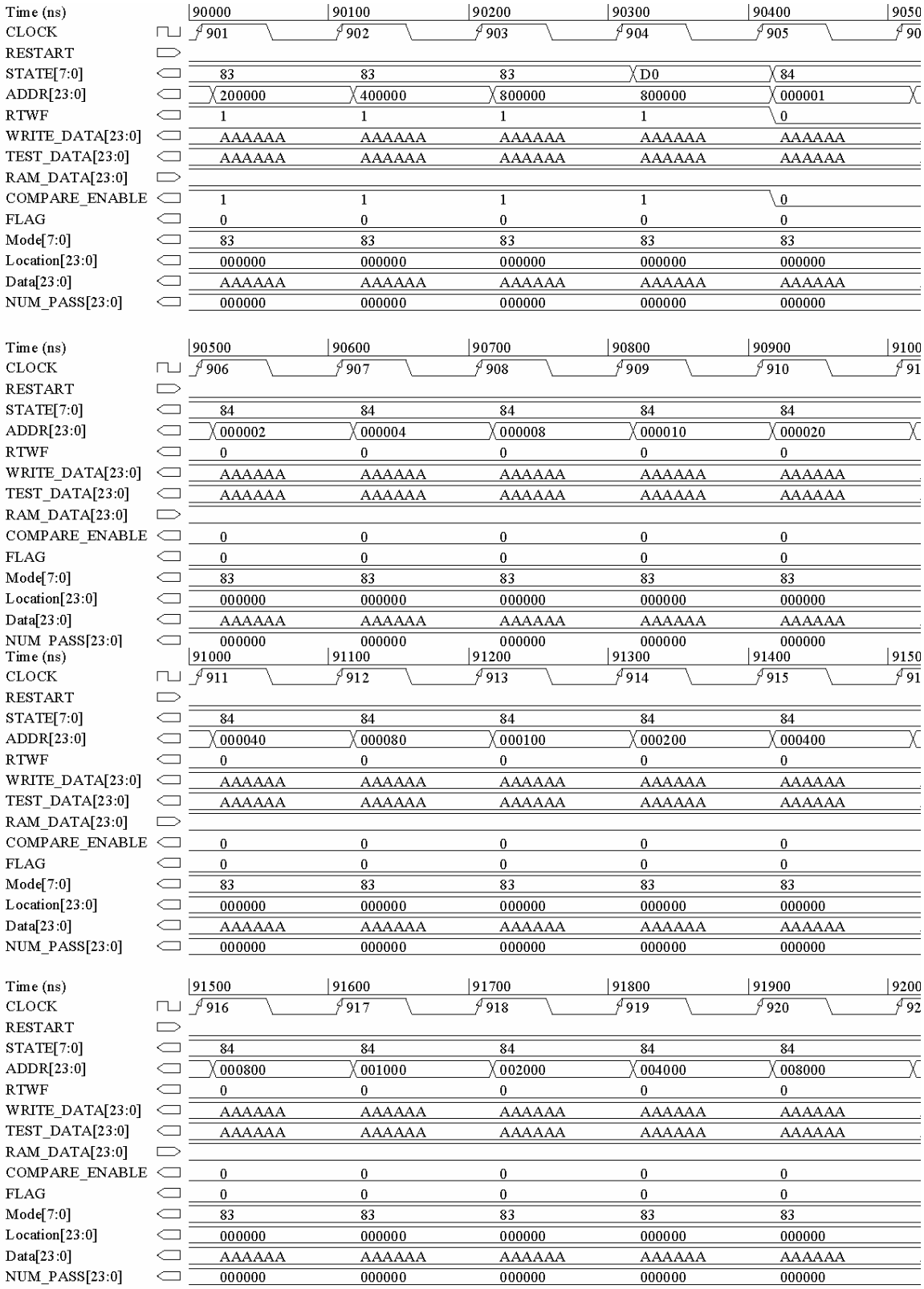

Time (ns) 84500 84600 84700 84800 84900 8500
CLOCK 846 847 848 849 850 85
RESTART
STATE[7:0] 7F 7F 7F 7F 7F
ADDR[23:0] 020000 040000 080000 100000 200000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 7F 7F 7F 7F 7F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

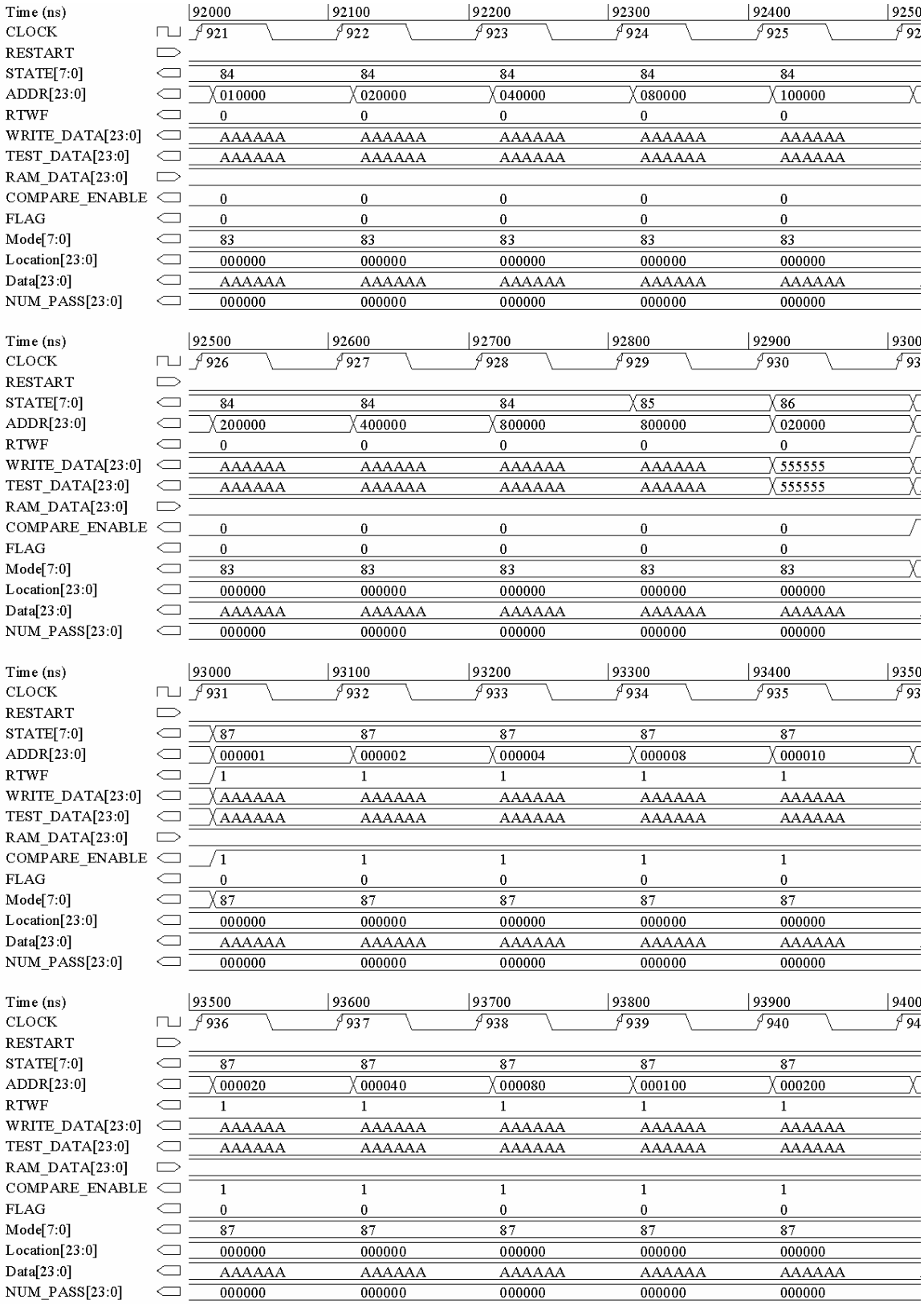

Time (ns) 85000 85100 85200 85300 85400 8550
CLOCK 851 852 853 854 855 85
RESTART
STATE[7:0] 7F 7F CF 80 80
ADDR[23:0] 400000 800000 800000 000001 000002
RTWF 1 1 1 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 0 0
FLAG 0 0 0 0 0
Mode[7:0] 7F 7F 7F 7F 7F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

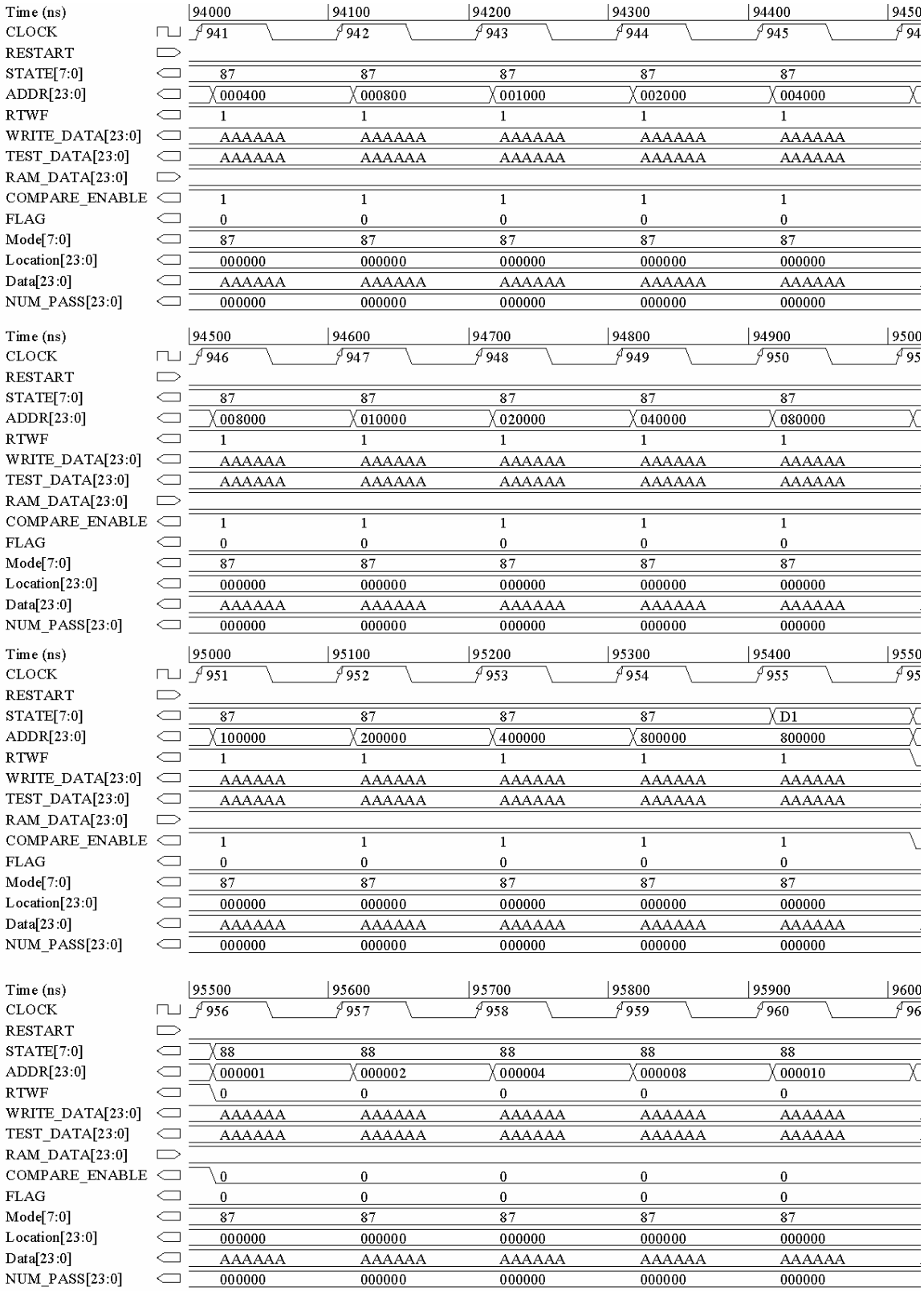

Time (ns) 85500 85600 85700 85800 85900 8600
CLOCK 856 857 858 859 860 86
RESTART
STATE[7:0] 80 80 80 80 80
ADDR[23:0] 000004 000008 000010 000020 000040
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 7F 7F 7F 7F 7F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

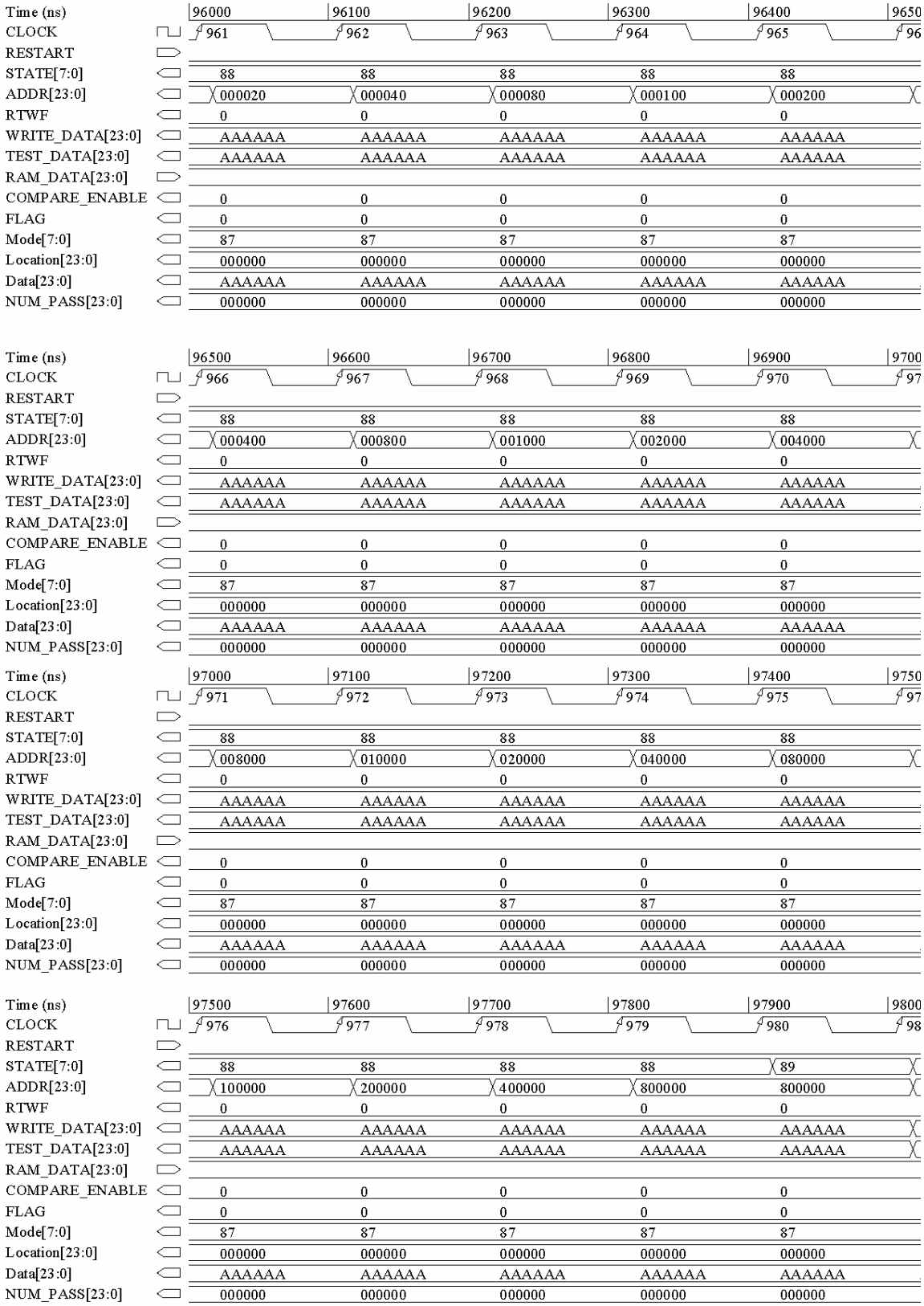

Time (ns) 86000 86100 86200 86300 86400 8650
CLOCK 861 862 863 864 865 86
RESTART
STATE[7:0] 80 80 80 80 80
ADDR[23:0] 000080 000100 000200 000400 000800
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 7F 7F 7F 7F 7F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 86500 86600 86700 86800 86900 8700
CLOCK 866 867 868 869 870 87
RESTART
STATE[7:0] 80 80 80 80 80
ADDR[23:0] 001000 002000 004000 008000 010000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 7F 7F 7F 7F 7F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 87000 87100 87200 87300 87400 8750
CLOCK 871 872 873 874 875 87
RESTART
STATE[7:0] 80 80 80 80 80
ADDR[23:0] 020000 040000 080000 100000 200000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 7F 7F 7F 7F 7F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 87500 87600 87700 87800 87900 8800
CLOCK 876 877 878 879 880 88
RESTART
STATE[7:0] 80 80 81 82 83
ADDR[23:0] 400000 800000 800000 010000 000001
RTWF 0 0 0 0 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA 555555 AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA 555555 AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 1
FLAG 0 0 0 0 0
Mode[7:0] 7F 7F 7F 7F 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

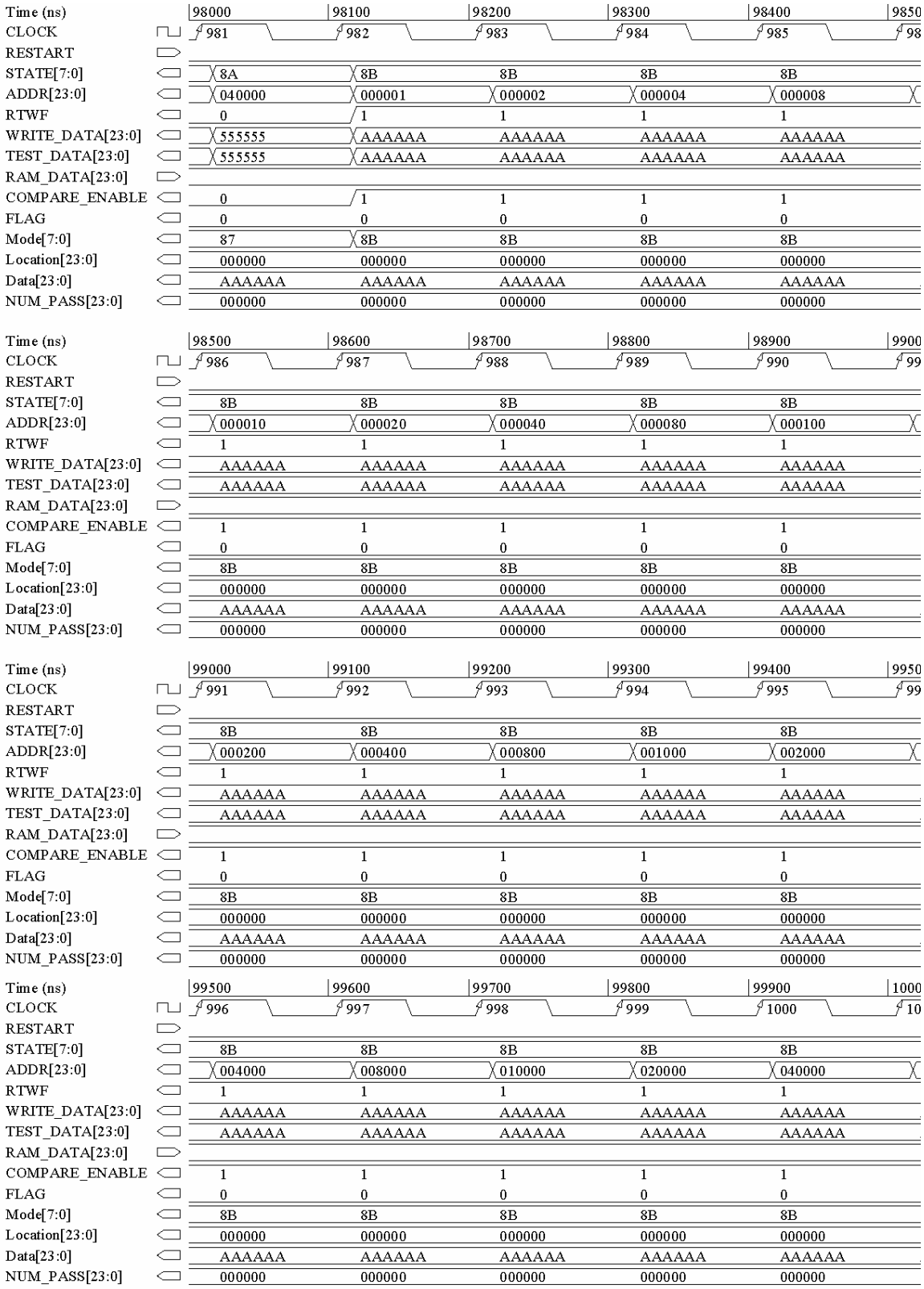

Time (ns) 88000 88100 88200 88300 88400 8850
CLOCK 881 882 883 884 885 88
RESTART
STATE[7:0] 83 83 83 83 83
ADDR[23:0] 000002 000004 000008 000010 000020
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 83 83 83 83 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 88500 88600 88700 88800 88900 8900
CLOCK 886 887 888 889 890 89
RESTART
STATE[7:0] 83 83 83 83 83
ADDR[23:0] 000040 000080 000100 000200 000400
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 83 83 83 83 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 89000 89100 89200 89300 89400 8950
CLOCK 891 892 893 894 895 89
RESTART
STATE[7:0] 83 83 83 83 83
ADDR[23:0] 000800 001000 002000 004000 008000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 83 83 83 83 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 89500 89600 89700 89800 89900 9000
CLOCK 896 897 898 899 900 90
RESTART
STATE[7:0] 83 83 83 83 83
ADDR[23:0] 010000 020000 040000 080000 100000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 83 83 83 83 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

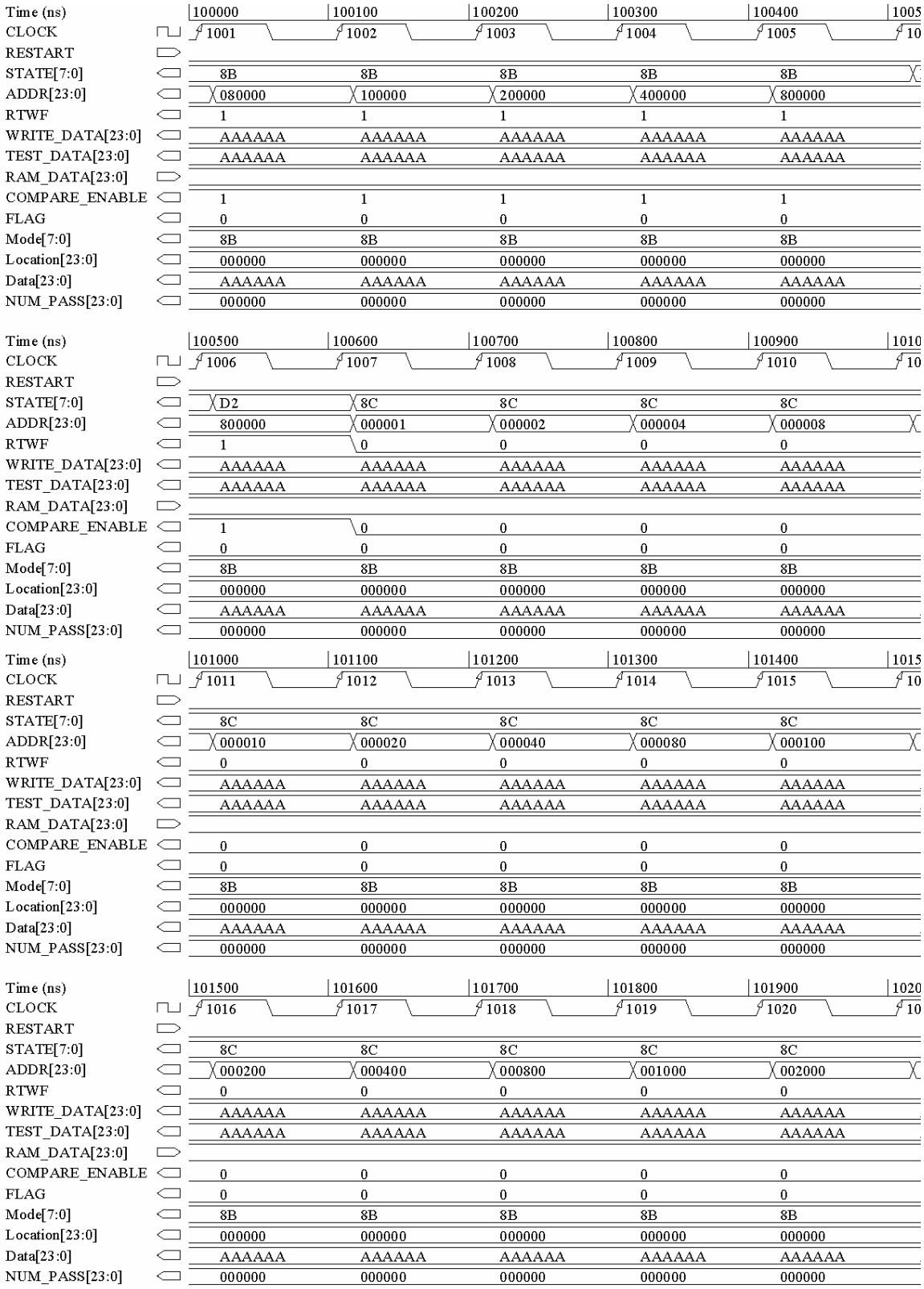

Time (ns) 90000 90100 90200 90300 90400 9050
CLOCK 901 902 903 904 905 90
RESTART
STATE[7:0] 83 83 83 D0 84
ADDR[23:0] 200000 400000 800000 800000 000001
RTWF 1 1 1 1 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 0
FLAG 0 0 0 0 0
Mode[7:0] 83 83 83 83 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 90500 90600 90700 90800 90900 9100
CLOCK 906 907 908 909 910 91
RESTART
STATE[7:0] 84 84 84 84 84
ADDR[23:0] 000002 000004 000008 000010 000020
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 83 83 83 83 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 91000 91100 91200 91300 91400 9150
CLOCK 911 912 913 914 915 91
RESTART
STATE[7:0] 84 84 84 84 84
ADDR[23:0] 000040 000080 000100 000200 000400
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 83 83 83 83 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 91500 91600 91700 91800 91900 9200
CLOCK 916 917 918 919 920 92
RESTART
STATE[7:0] 84 84 84 84 84
ADDR[23:0] 000800 001000 002000 004000 008000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 83 83 83 83 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

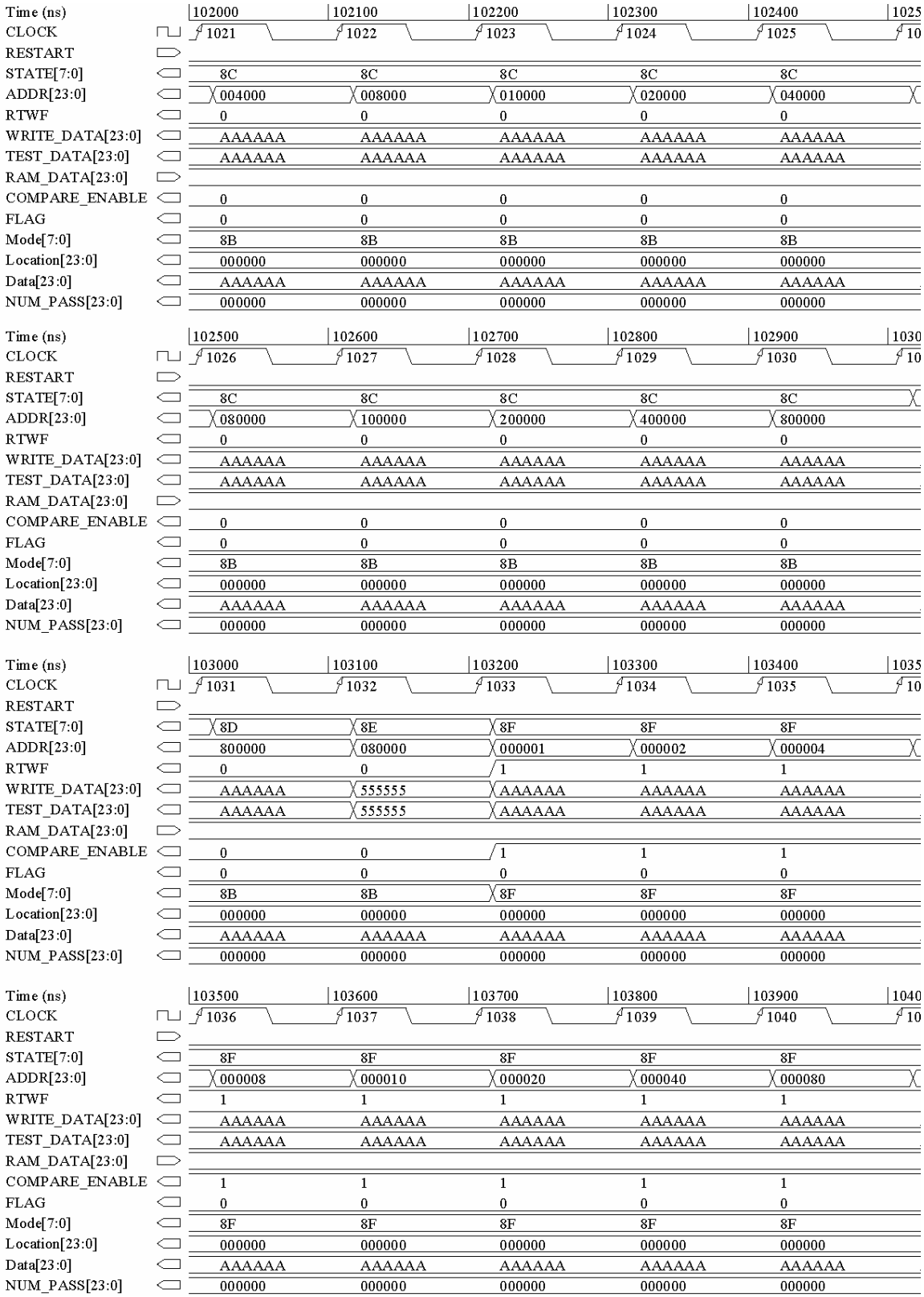

Time (ns) 92000 92100 92200 92300 92400 9250
CLOCK 921 922 923 924 925 92
RESTART
STATE[7:0] 84 84 84 84 84
ADDR[23:0] 010000 020000 040000 080000 100000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 83 83 83 83 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 92500 92600 92700 92800 92900 9300
CLOCK 926 927 928 929 930 93
RESTART
STATE[7:0] 84 84 84 85 86
ADDR[23:0] 200000 400000 800000 800000 020000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA 555555
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA 555555
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 83 83 83 83 83
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 93000 93100 93200 93300 93400 9350
CLOCK 931 932 933 934 935 93
RESTART
STATE[7:0] 87 87 87 87 87
ADDR[23:0] 000001 000002 000004 000008 000010
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 87 87 87 87 87
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 93500 93600 93700 93800 93900 9400
CLOCK 936 937 938 939 940 94
RESTART
STATE[7:0] 87 87 87 87 87
ADDR[23:0] 000020 000040 000080 000100 000200
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 87 87 87 87 87
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

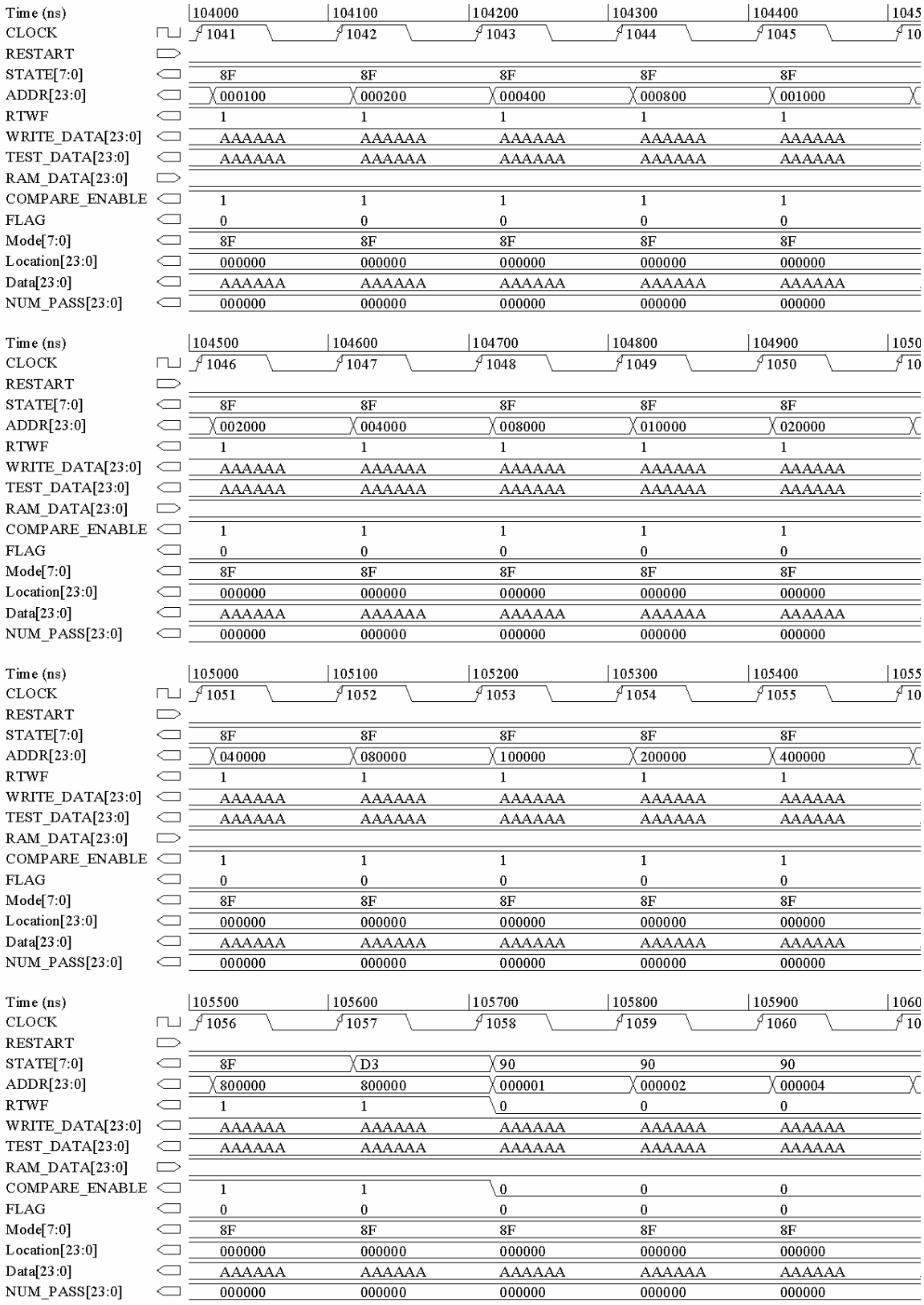

Time (ns) 94000 94100 94200 94300 94400 9450
CLOCK 941 942 943 944 945 94
RESTART
STATE[7:0] 87 87 87 87 87
ADDR[23:0] 000400 000800 001000 002000 004000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 87 87 87 87 87
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 94500 94600 94700 94800 94900 9500
CLOCK 946 947 948 949 950 95
RESTART
STATE[7:0] 87 87 87 87 87
ADDR[23:0] 008000 010000 020000 040000 080000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 87 87 87 87 87
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 95000 95100 95200 95300 95400 9550
CLOCK 951 952 953 954 955 95
RESTART
STATE[7:0] 87 87 87 87 D1
ADDR[23:0] 100000 200000 400000 800000 800000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 87 87 87 87 87
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 95500 95600 95700 95800 95900 9600
CLOCK 956 957 958 959 960 96
RESTART
STATE[7:0] 88 88 88 88 88
ADDR[23:0] 000001 000002 000004 000008 000010
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 87 87 87 87 87
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

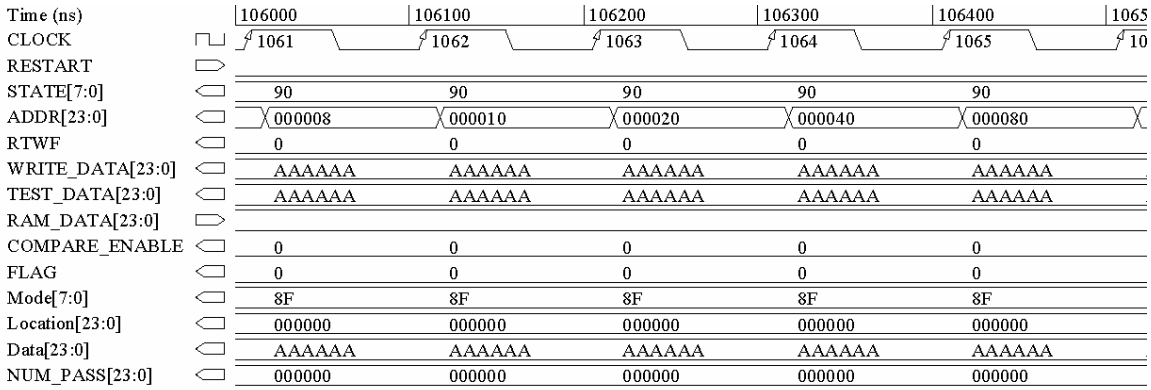

Time (ns) 96000 96100 96200 96300 96400 9650
CLOCK 961 962 963 964 965 96
RESTART
STATE[7:0] 88 88 88 88 88
ADDR[23:0] 000020 000040 000080 000100 000200
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 87 87 87 87 87
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 96500 96600 96700 96800 96900 9700
CLOCK 966 967 968 969 970 97
RESTART
STATE[7:0] 88 88 88 88 88
ADDR[23:0] 000400 000800 001000 002000 004000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 87 87 87 87 87
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 97000 97100 97200 97300 97400 9750
CLOCK 971 972 973 974 975 97
RESTART
STATE[7:0] 88 88 88 88 88
ADDR[23:0] 008000 010000 020000 040000 080000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 87 87 87 87 87
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 97500 97600 97700 97800 97900 9800
CLOCK 976 977 978 979 980 98
RESTART
STATE[7:0] 88 88 88 88 89
ADDR[23:0] 100000 200000 400000 800000 800000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 87 87 87 87 87
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

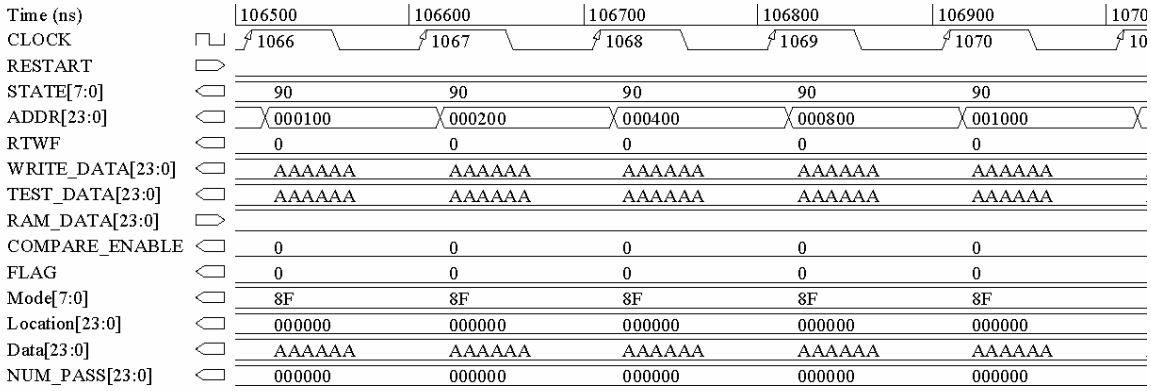

Time (ns) 98000 98100 98200 98300 98400 9850
CLOCK 981 982 983 984 985 98
RESTART
STATE[7:0] 8A 8B 8B 8B 8B
ADDR[23:0] 040000 000001 000002 000004 000008
RTWF 0 1 1 1 1
WRITE_DATA[23:0] 555555 AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] 555555 AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 87 8B 8B 8B 8B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 98500 98600 98700 98800 98900 9900
CLOCK 986 987 988 989 990 99
RESTART
STATE[7:0] 8B 8B 8B 8B 8B
ADDR[23:0] 000010 000020 000040 000080 000100
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 8B 8B 8B 8B 8B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 99000 99100 99200 99300 99400 9950
CLOCK 991 992 993 994 995 99
RESTART
STATE[7:0] 8B 8B 8B 8B 8B
ADDR[23:0] 000200 000400 000800 001000 002000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 8B 8B 8B 8B 8B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 99500 99600 99700 99800 99900 1000
CLOCK 996 997 998 999 1000 10
RESTART
STATE[7:0] 8B 8B 8B 8B 8B
ADDR[23:0] 004000 008000 010000 020000 040000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 8B 8B 8B 8B 8B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

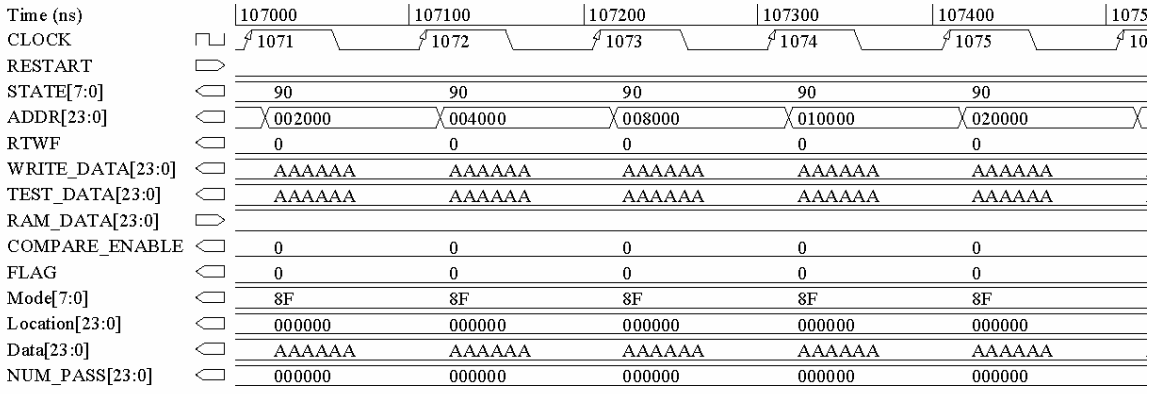

Time (ns) 100000 100100 100200 100300 100400 1005
CLOCK 1001 1002 1003 1004 1005 10
RESTART
STATE[7:0] 8B 8B 8B 8B 8B
ADDR[23:0] 080000 100000 200000 400000 800000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 8B 8B 8B 8B 8B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 100500 100600 100700 100800 100900 1010
CLOCK 1006 1007 1008 1009 1010 10
RESTART
STATE[7:0] D2 8C 8C 8C 8C
ADDR[23:0] 800000 000001 000002 000004 000008
RTWF 1 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 8B 8B 8B 8B 8B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 101000 101100 101200 101300 101400 1015
CLOCK 1011 1012 1013 1014 1015 10
RESTART
STATE[7:0] 8C 8C 8C 8C 8C
ADDR[23:0] 000010 000020 000040 000080 000100
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 8B 8B 8B 8B 8B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 101500 101600 101700 101800 101900 1020
CLOCK 1016 1017 1018 1019 1020 10
RESTART
STATE[7:0] 8C 8C 8C 8C 8C
ADDR[23:0] 000200 000400 000800 001000 002000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 8B 8B 8B 8B 8B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

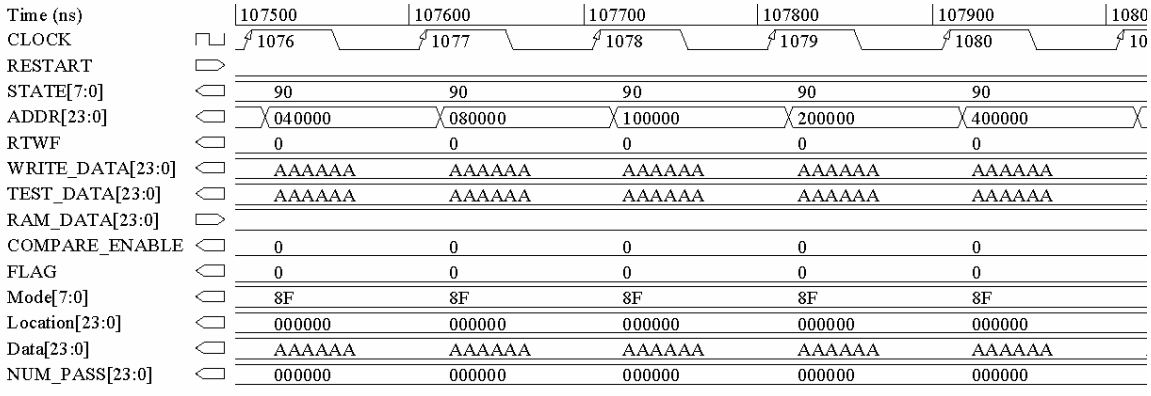

Time (ns) 102000 102100 102200 102300 102400 1025
CLOCK 1021 1022 1023 1024 1025 10
RESTART
STATE[7:0] 8C 8C 8C 8C 8C
ADDR[23:0] 004000 008000 010000 020000 040000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 8B 8B 8B 8B 8B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 102500 102600 102700 102800 102900 1030
CLOCK 1026 1027 1028 1029 1030 10
RESTART
STATE[7:0] 8C 8C 8C 8C 8C
ADDR[23:0] 080000 100000 200000 400000 800000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 8B 8B 8B 8B 8B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 103000 103100 103200 103300 103400 1035
CLOCK 1031 1032 1033 1034 1035 10
RESTART
STATE[7:0] 8D 8E 8F 8F 8F
ADDR[23:0] 800000 080000 000001 000002 000004
RTWF 0 0 1 1 1
WRITE_DATA[23:0] AAAAAA 555555 AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA 555555 AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 8B 8B 8F 8F 8F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 103500 103600 103700 103800 103900 1040
CLOCK 1036 1037 1038 1039 1040 10
RESTART
STATE[7:0] 8F 8F 8F 8F 8F
ADDR[23:0] 000008 000010 000020 000040 000080
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 8F 8F 8F 8F 8F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

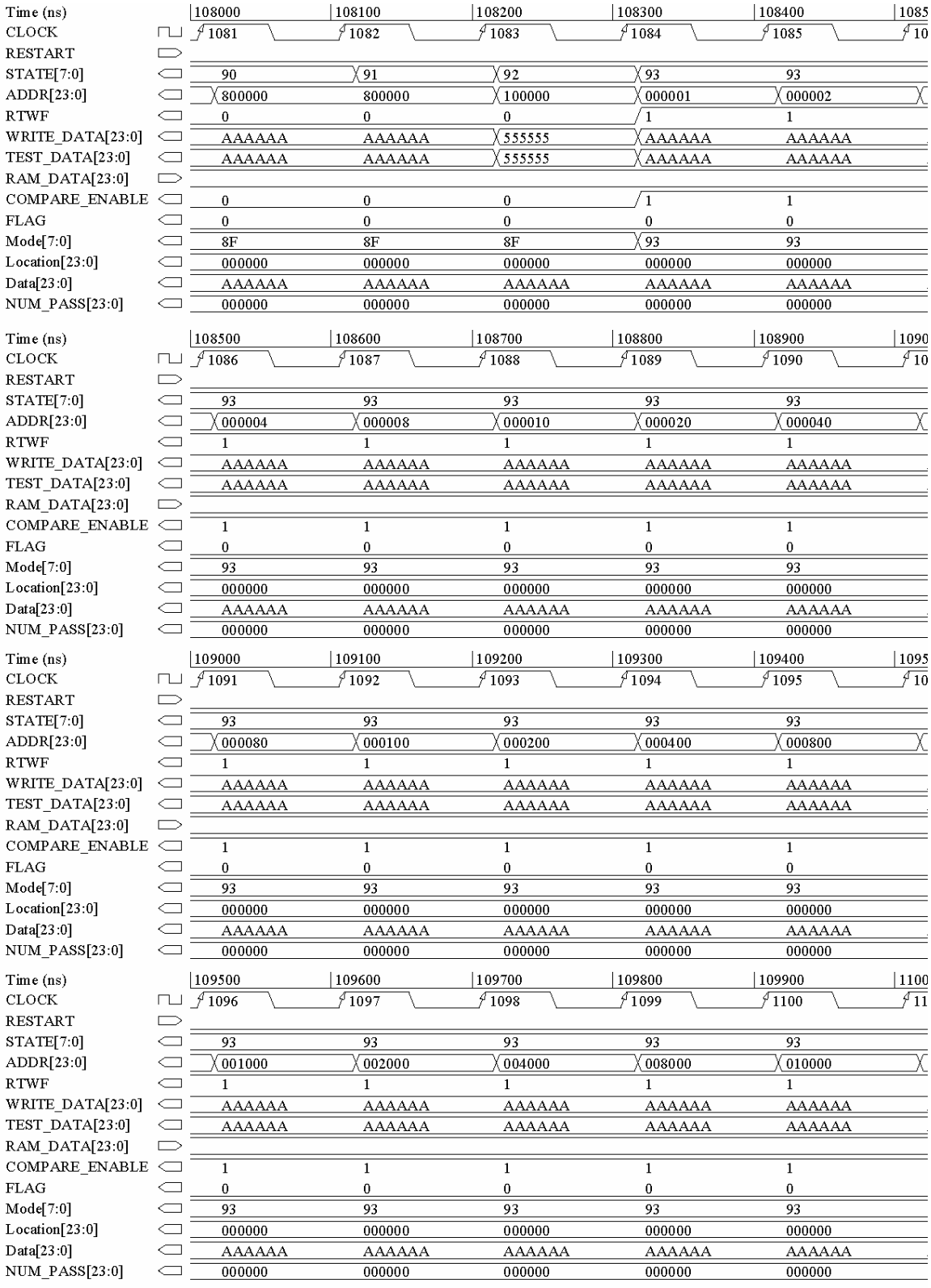

Time (ns) 104000 104100 104200 104300 104400 1045
CLOCK 1041 1042 1043 1044 1045 10
RESTART
STATE[7:0] 8F 8F 8F 8F 8F
ADDR[23:0] 000100 000200 000400 000800 001000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 8F 8F 8F 8F 8F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 104500 104600 104700 104800 104900 1050
CLOCK 1046 1047 1048 1049 1050 10
RESTART
STATE[7:0] 8F 8F 8F 8F 8F
ADDR[23:0] 002000 004000 008000 010000 020000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 8F 8F 8F 8F 8F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 105000 105100 105200 105300 105400 1055
CLOCK 1051 1052 1053 1054 1055 10
RESTART
STATE[7:0] 8F 8F 8F 8F 8F
ADDR[23:0] 040000 080000 100000 200000 400000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 8F 8F 8F 8F 8F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 105500 105600 105700 105800 105900 1060
CLOCK 1056 1057 1058 1059 1060 10
RESTART
STATE[7:0] 8F D3 90 90 90
ADDR[23:0] 800000 800000 000001 000002 000004
RTWF 1 1 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 8F 8F 8F 8F 8F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

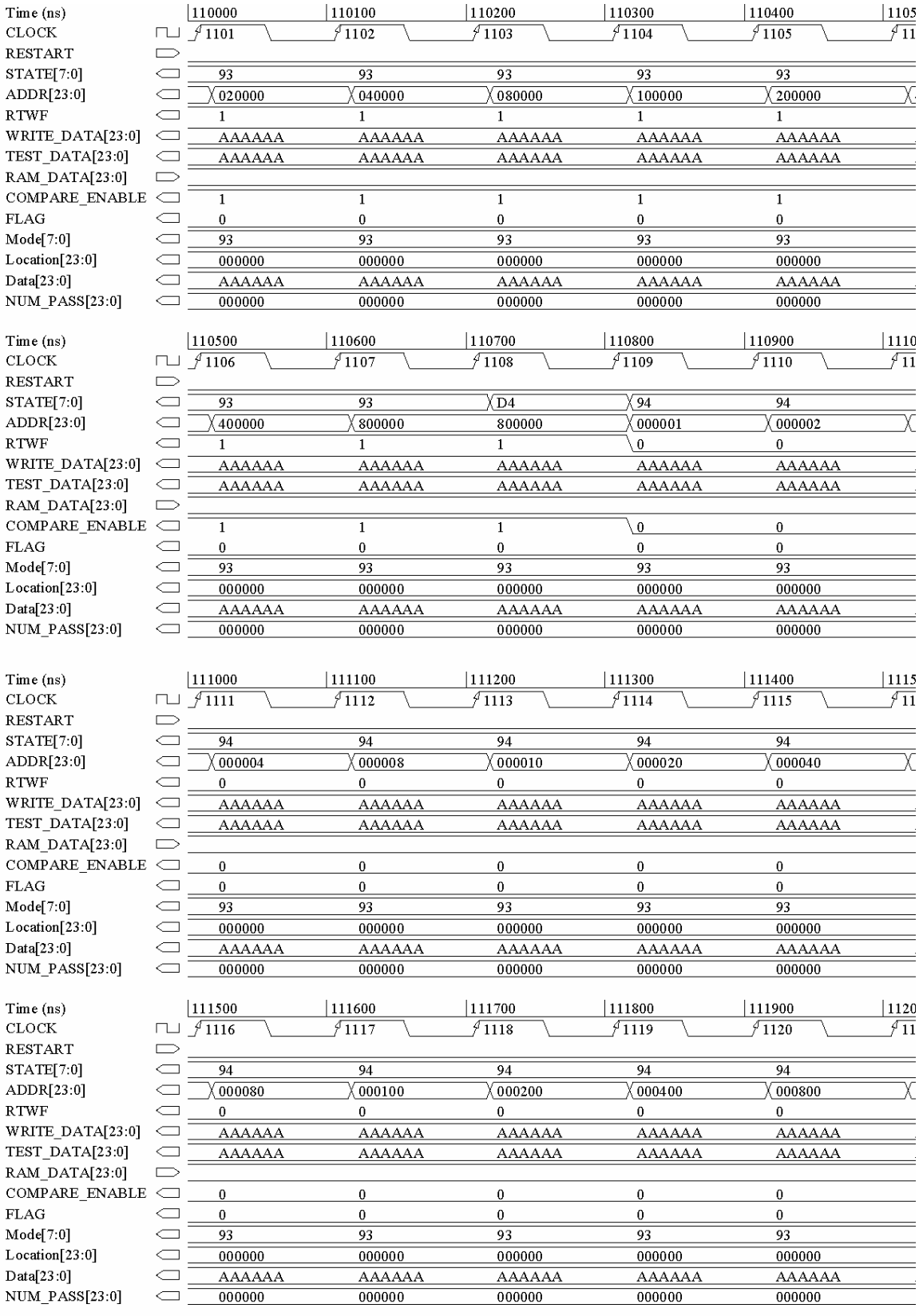

Time (ns) 106000 106100 106200 106300 106400 1065
CLOCK 1061 1062 1063 1064 1065 10
RESTART
STATE[7:0] 90 90 90 90 90
ADDR[23:0] 000008 000010 000020 000040 000080
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 8F 8F 8F 8F 8F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

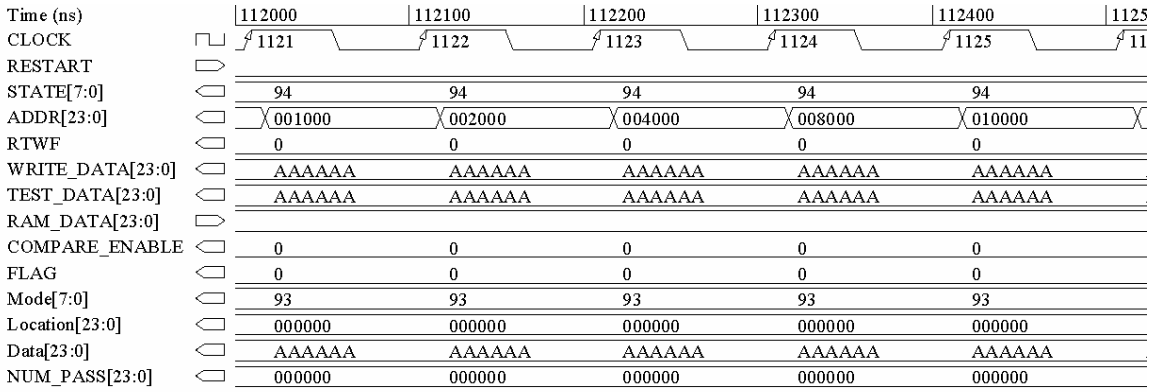

Time (ns) 106500 106600 106700 106800 106900 1070
CLOCK 1066 1067 1068 1069 1070 10
RESTART
STATE[7:0] 90 90 90 90 90
ADDR[23:0] 000100 000200 000400 000800 001000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 8F 8F 8F 8F 8F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

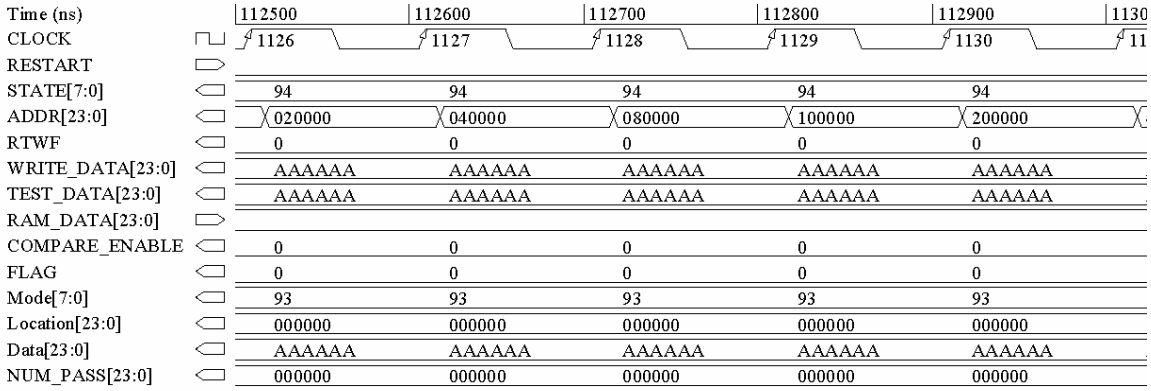

Time (ns) 107000 107100 107200 107300 107400 1075
CLOCK 1071 1072 1073 1074 1075 10
RESTART
STATE[7:0] 90 90 90 90 90
ADDR[23:0] 002000 004000 008000 010000 020000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 8F 8F 8F 8F 8F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

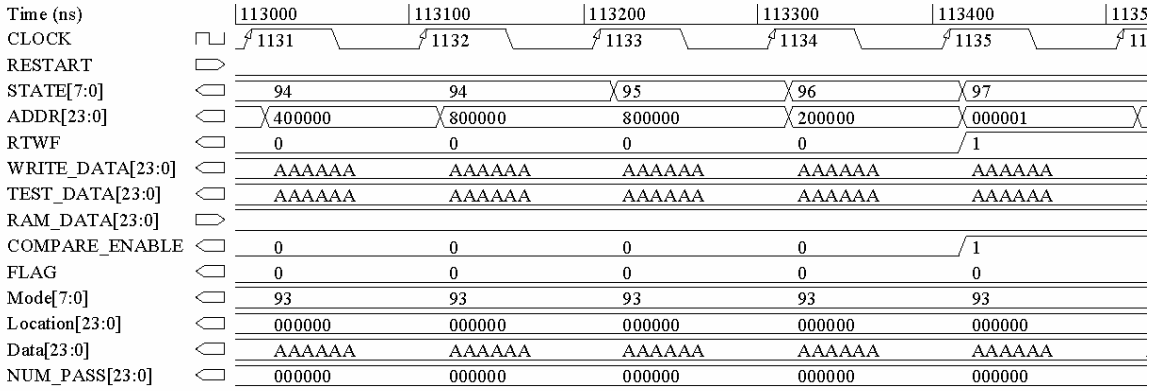

Time (ns) 107500 107600 107700 107800 107900 1080
CLOCK 1076 1077 1078 1079 1080 10
RESTART
STATE[7:0] 90 90 90 90 90
ADDR[23:0] 040000 080000 100000 200000 400000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 8F 8F 8F 8F 8F
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

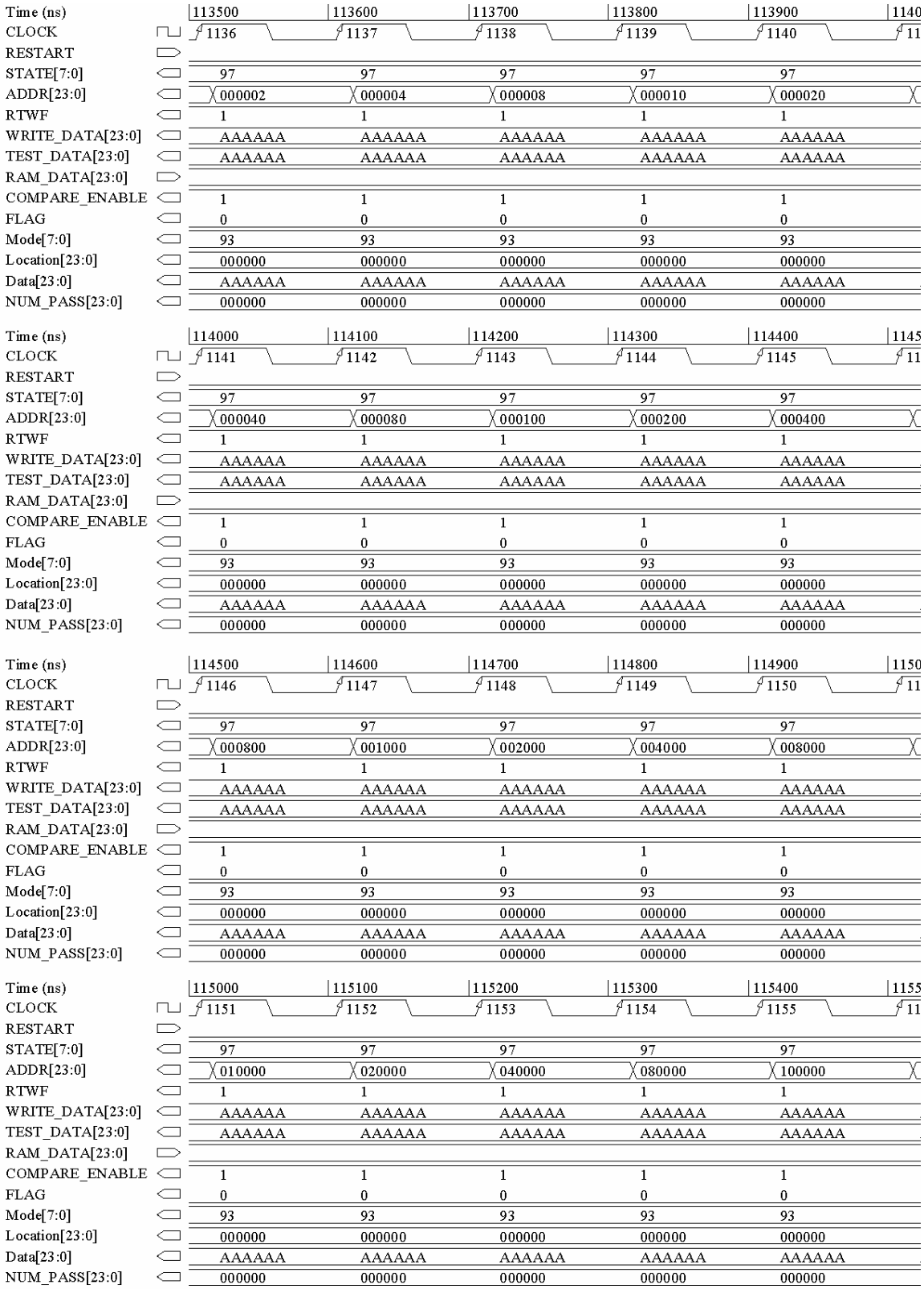

Time (ns) 108000 108100 108200 108300 108400 1085
CLOCK 1081 1082 1083 1084 1085 10
RESTART
STATE[7:0] 90 91 92 93 93
ADDR[23:0] 800000 800000 100000 000001 000002
RTWF 0 0 0 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA 555555 AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA 555555 AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 1 1
FLAG 0 0 0 0 0
Mode[7:0] 8F 8F 8F 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 108500 108600 108700 108800 108900 1090
CLOCK 1086 1087 1088 1089 1090 10
RESTART
STATE[7:0] 93 93 93 93 93
ADDR[23:0] 000004 000008 000010 000020 000040
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 109000 109100 109200 109300 109400 1095
CLOCK 1091 1092 1093 1094 1095 10
RESTART
STATE[7:0] 93 93 93 93 93
ADDR[23:0] 000080 000100 000200 000400 000800
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 109500 109600 109700 109800 109900 1100
CLOCK 1096 1097 1098 1099 1100 11
RESTART
STATE[7:0] 93 93 93 93 93
ADDR[23:0] 001000 002000 004000 008000 010000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

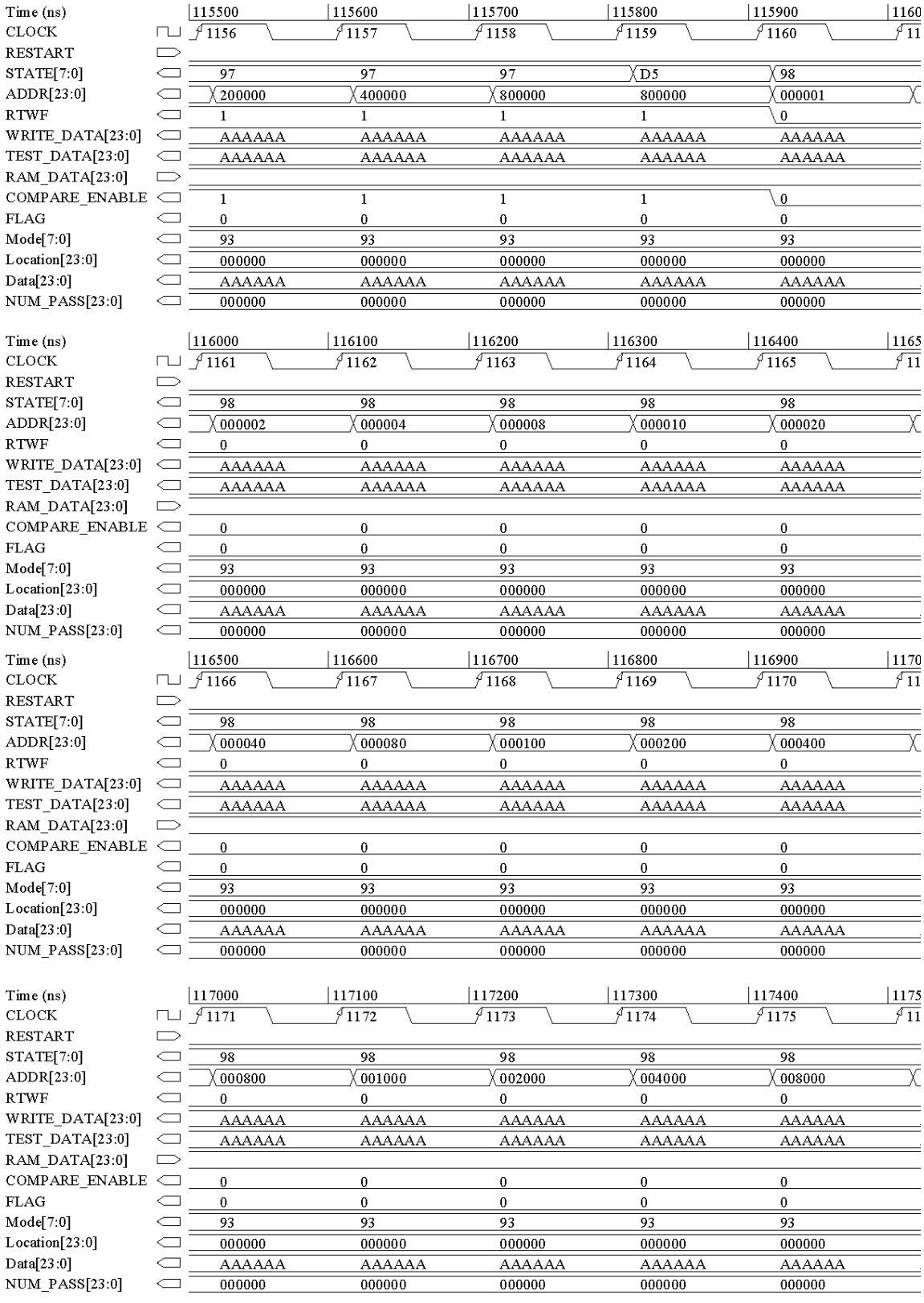

Time (ns) 110000 110100 110200 110300 110400 1105
CLOCK 1101 1102 1103 1104 1105 11
RESTART
STATE[7:0] 93 93 93 93 93
ADDR[23:0] 020000 040000 080000 100000 200000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 110500 110600 110700 110800 110900 1110
CLOCK 1106 1107 1108 1109 1110 11
RESTART
STATE[7:0] 93 93 D4 94 94
ADDR[23:0] 400000 800000 800000 000001 000002
RTWF 1 1 1 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 0 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 111000 111100 111200 111300 111400 1115
CLOCK 1111 1112 1113 1114 1115 11
RESTART
STATE[7:0] 94 94 94 94 94
ADDR[23:0] 000004 000008 000010 000020 000040
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 111500 111600 111700 111800 111900 1120
CLOCK 1116 1117 1118 1119 1120 11
RESTART
STATE[7:0] 94 94 94 94 94
ADDR[23:0] 000080 000100 000200 000400 000800
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

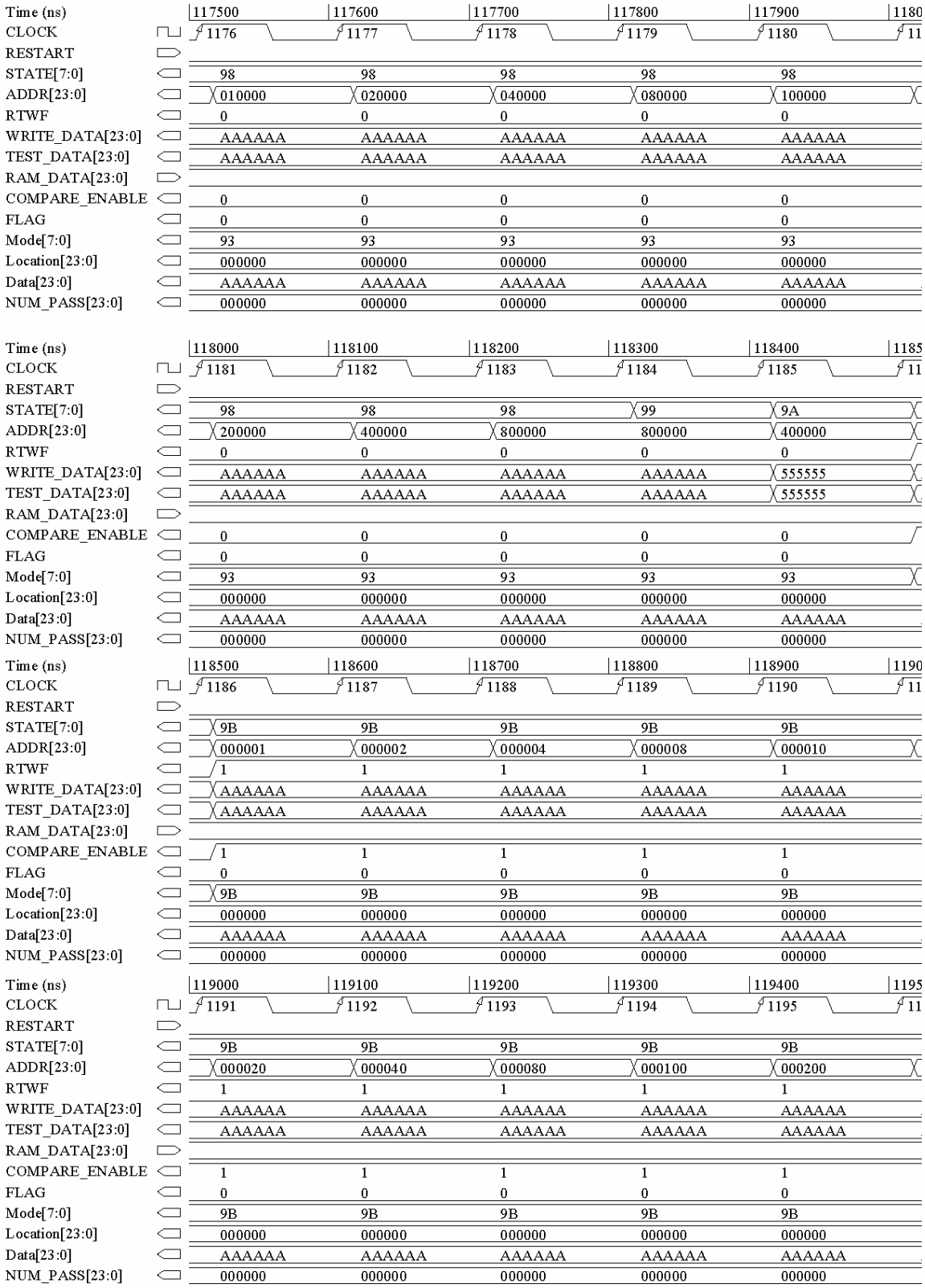

Time (ns) 112000 112100 112200 112300 112400 1125
CLOCK 1121 1122 1123 1124 1125 11
RESTART
STATE[7:0] 94 94 94 94 94
ADDR[23:0] 001000 002000 004000 008000 010000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

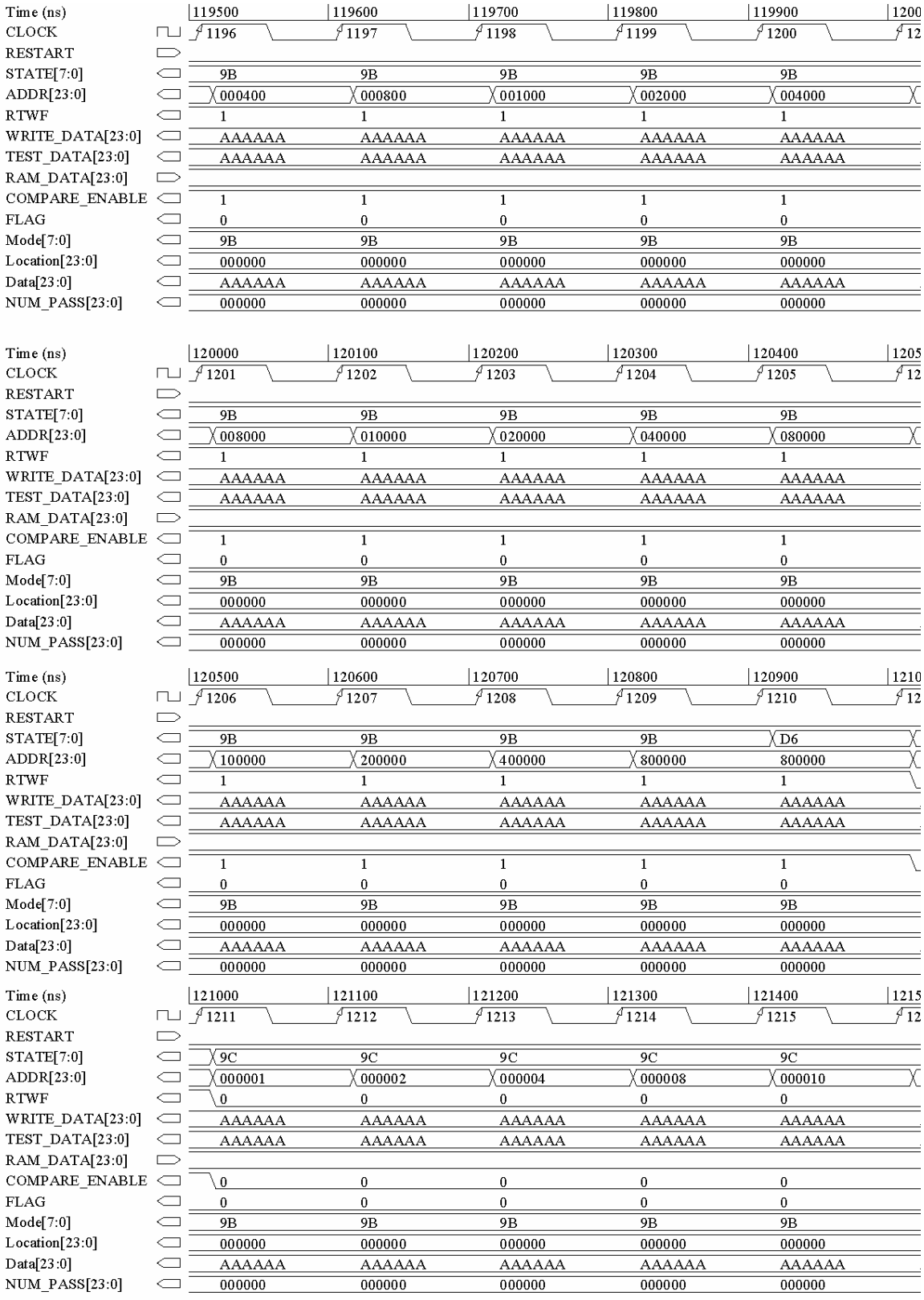

Time (ns) 112500 112600 112700 112800 112900 1130
CLOCK 1126 1127 1128 1129 1130 11
RESTART
STATE[7:0] 94 94 94 94 94
ADDR[23:0] 020000 040000 080000 100000 200000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

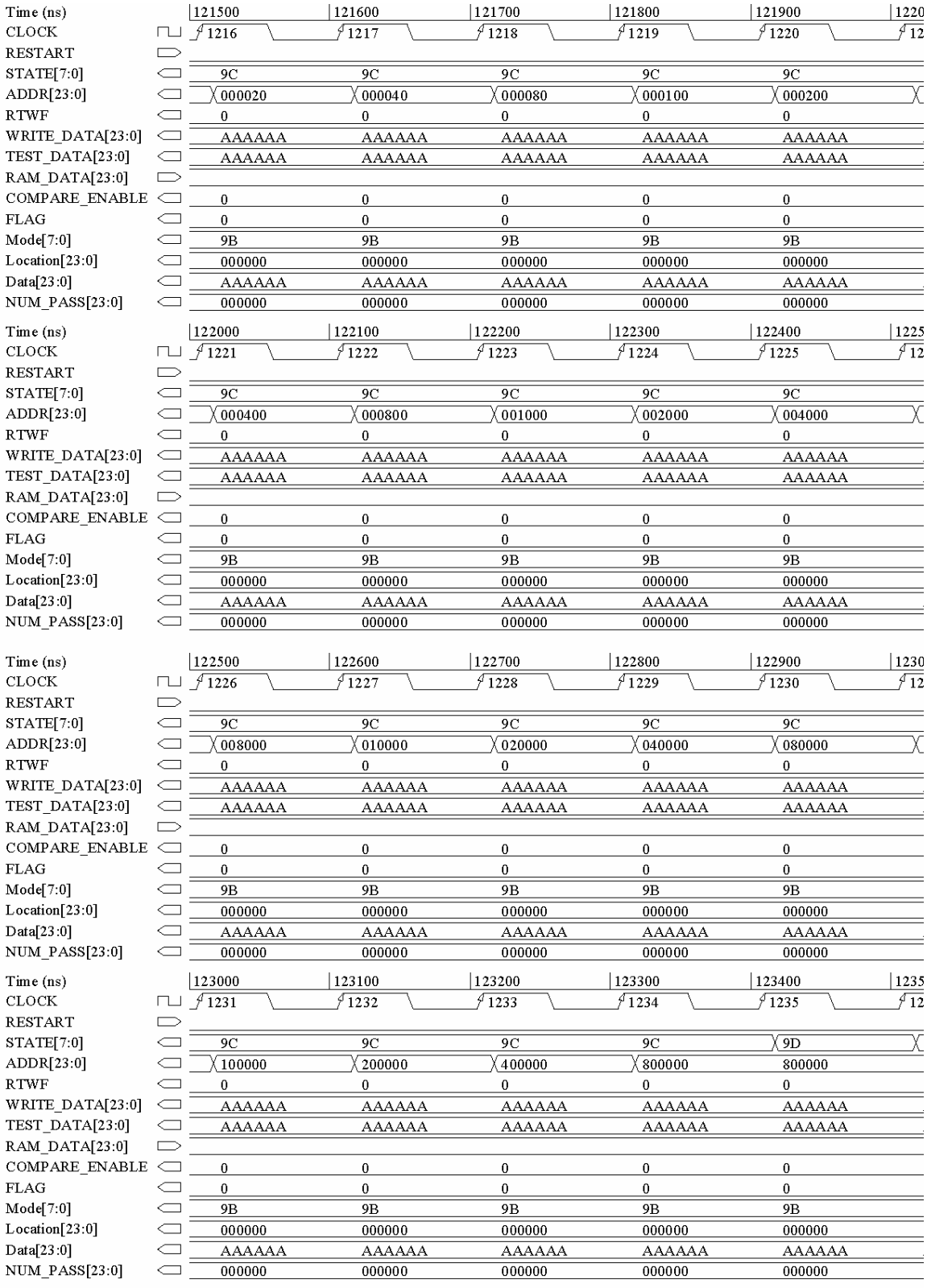

Time (ns) 113000 113100 113200 113300 113400 1135
CLOCK 1131 1132 1133 1134 1135 11
RESTART
STATE[7:0] 94 94 95 96 97
ADDR[23:0] 400000 800000 800000 200000 000001
RTWF 0 0 0 0 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 1
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

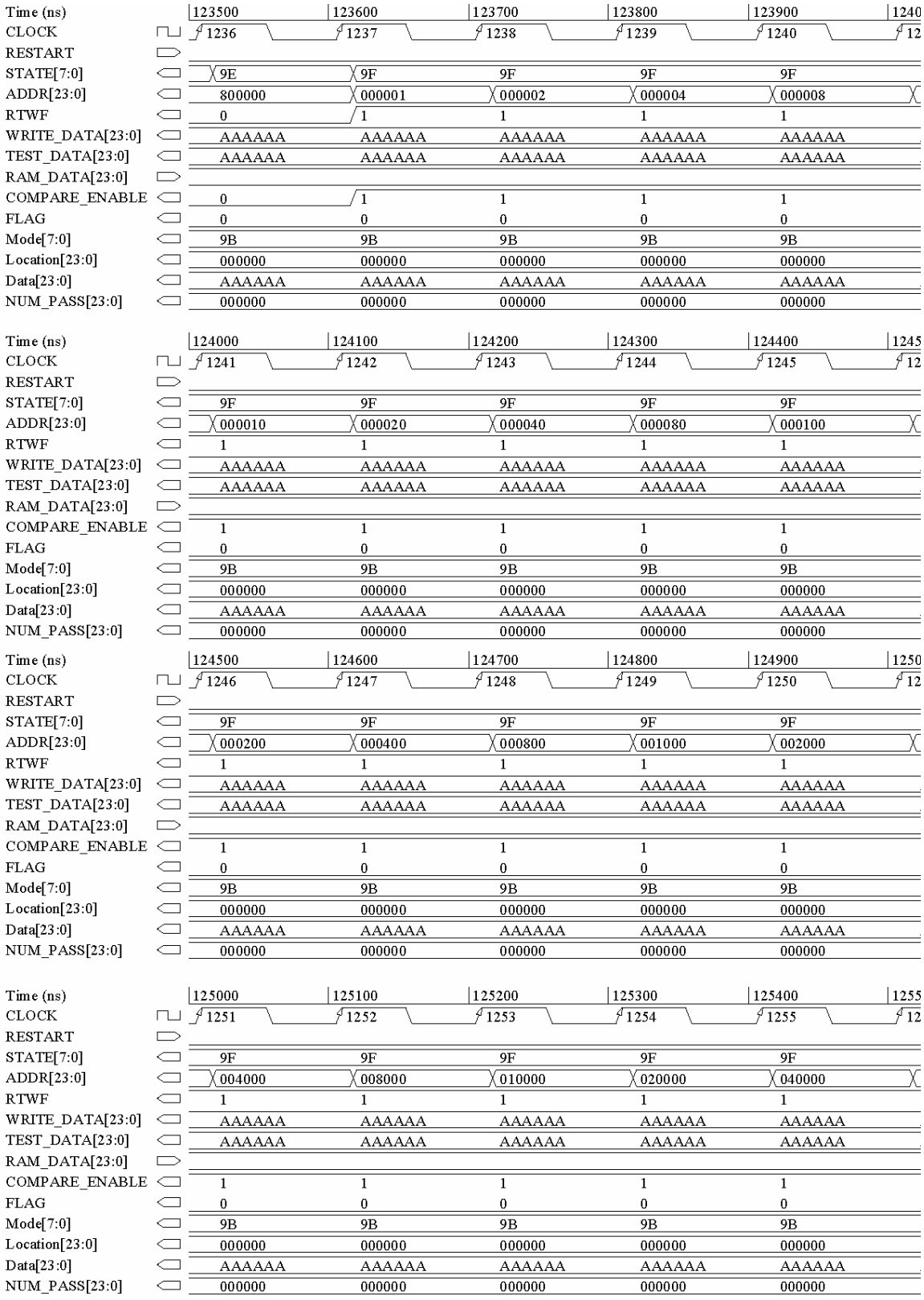

Time (ns) 113500 113600 113700 113800 113900 1140
CLOCK 1136 1137 1138 1139 1140 11
RESTART
STATE[7:0] 97 97 97 97 97
ADDR[23:0] 000002 000004 000008 000010 000020
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 114000 114100 114200 114300 114400 1145
CLOCK 1141 1142 1143 1144 1145 11
RESTART
STATE[7:0] 97 97 97 97 97
ADDR[23:0] 000040 000080 000100 000200 000400
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 114500 114600 114700 114800 114900 1150
CLOCK 1146 1147 1148 1149 1150 11
RESTART
STATE[7:0] 97 97 97 97 97
ADDR[23:0] 000800 001000 002000 004000 008000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 115000 115100 115200 115300 115400 1155
CLOCK 1151 1152 1153 1154 1155 11
RESTART
STATE[7:0] 97 97 97 97 97
ADDR[23:0] 010000 020000 040000 080000 100000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

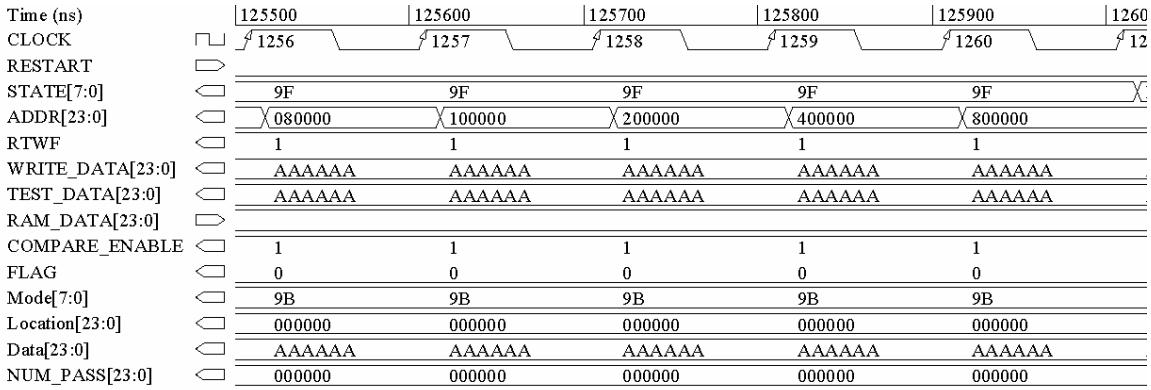

Time (ns) 115500 115600 115700 115800 115900 1160
CLOCK 1156 1157 1158 1159 1160 11
RESTART
STATE[7:0] 97 97 97 D5 98
ADDR[23:0] 200000 400000 800000 800000 000001
RTWF 1 1 1 1 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 116000 116100 116200 116300 116400 1165
CLOCK 1161 1162 1163 1164 1165 11
RESTART
STATE[7:0] 98 98 98 98 98
ADDR[23:0] 000002 000004 000008 000010 000020
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 116500 116600 116700 116800 116900 1170
CLOCK 1166 1167 1168 1169 1170 11
RESTART
STATE[7:0] 98 98 98 98 98
ADDR[23:0] 000040 000080 000100 000200 000400
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 117000 117100 117200 117300 117400 1175
CLOCK 1171 1172 1173 1174 1175 11
RESTART
STATE[7:0] 98 98 98 98 98
ADDR[23:0] 000800 001000 002000 004000 008000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000


Time (ns) 117500 117600 117700 117800 117900 1180
CLOCK 1176 1177 1178 1179 1180 11
RESTART
STATE[7:0] 98 98 98 98 98
ADDR[23:0] 010000 020000 040000 080000 100000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 118000 118100 118200 118300 118400 1185
CLOCK 1181 1182 1183 1184 1185 11
RESTART
STATE[7:0] 98 98 98 99 9A
ADDR[23:0] 200000 400000 800000 800000 400000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA 555555
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA 555555
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 93 93 93 93 93
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 118500 118600 118700 118800 118900 1190
CLOCK 1186 1187 1188 1189 1190 11
RESTART
STATE[7:0] 9B 9B 9B 9B 9B
ADDR[23:0] 000001 000002 000004 000008 000010
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 119000 119100 119200 119300 119400 1195
CLOCK 1191 1192 1193 1194 1195 11
RESTART
STATE[7:0] 9B 9B 9B 9B 9B
ADDR[23:0] 000020 000040 000080 000100 000200
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

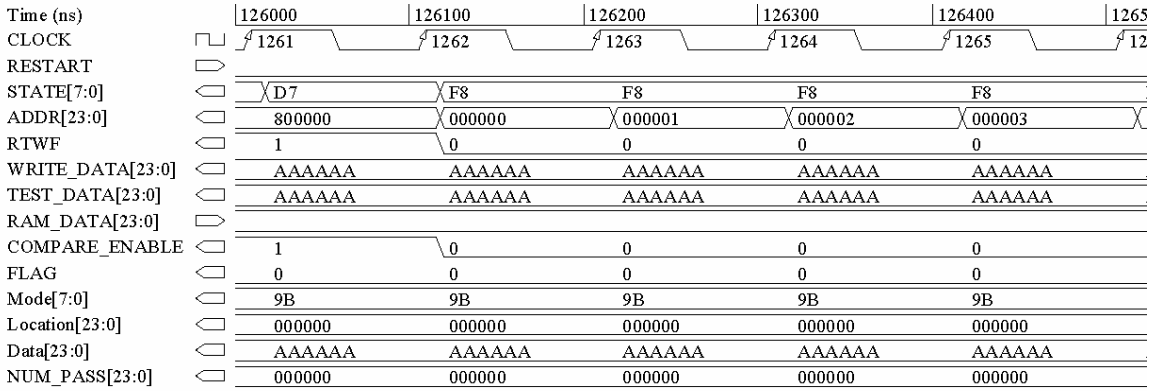

Time (ns) 119500 119600 119700 119800 119900 1200
CLOCK 1196 1197 1198 1199 1200 12
RESTART
STATE[7:0] 9B 9B 9B 9B 9B
ADDR[23:0] 000400 000800 001000 002000 004000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 120000 120100 120200 120300 120400 1205
CLOCK 1201 1202 1203 1204 1205 12
RESTART
STATE[7:0] 9B 9B 9B 9B 9B
ADDR[23:0] 008000 010000 020000 040000 080000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 120500 120600 120700 120800 120900 1210
CLOCK 1206 1207 1208 1209 1210 12
RESTART
STATE[7:0] 9B 9B 9B 9B D6
ADDR[23:0] 100000 200000 400000 800000 800000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 121000 121100 121200 121300 121400 1215
CLOCK 1211 1212 1213 1214 1215 12
RESTART
STATE[7:0] 9C 9C 9C 9C 9C
ADDR[23:0] 000001 000002 000004 000008 000010
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000


Time (ns) 121500 121600 121700 121800 121900 1220
CLOCK 1216 1217 1218 1219 1220 12
RESTART
STATE[7:0] 9C 9C 9C 9C 9C
ADDR[23:0] 000020 000040 000080 000100 000200
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 122000 122100 122200 122300 122400 1225
CLOCK 1221 1222 1223 1224 1225 12
RESTART
STATE[7:0] 9C 9C 9C 9C 9C
ADDR[23:0] 000400 000800 001000 002000 004000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 122500 122600 122700 122800 122900 1230
CLOCK 1226 1227 1228 1229 1230 12
RESTART
STATE[7:0] 9C 9C 9C 9C 9C
ADDR[23:0] 008000 010000 020000 040000 080000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 123000 123100 123200 123300 123400 1235
CLOCK 1231 1232 1233 1234 1235 12
RESTART
STATE[7:0] 9C 9C 9C 9C 9D
ADDR[23:0] 100000 200000 400000 800000 800000
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

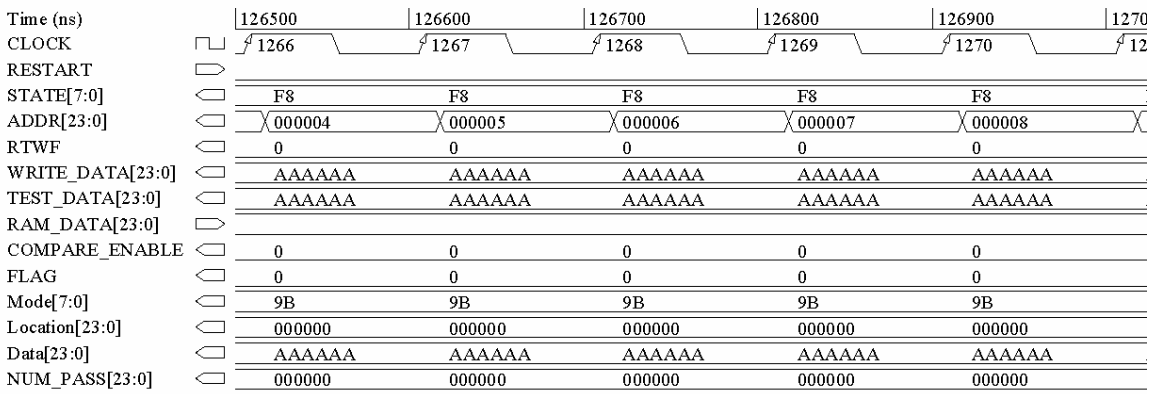

Time (ns) 123500 123600 123700 123800 123900 1240
CLOCK 1236 1237 1238 1239 1240 12
RESTART
STATE[7:0] 9E 9F 9F 9F 9F
ADDR[23:0] 800000 000001 000002 000004 000008
RTWF 0 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 124000 124100 124200 124300 124400 1245
CLOCK 1241 1242 1243 1244 1245 12
RESTART
STATE[7:0] 9F 9F 9F 9F 9F
ADDR[23:0] 000010 000020 000040 000080 000100
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 124500 124600 124700 124800 124900 1250
CLOCK 1246 1247 1248 1249 1250 12
RESTART
STATE[7:0] 9F 9F 9F 9F 9F
ADDR[23:0] 000200 000400 000800 001000 002000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 125000 125100 125200 125300 125400 1255
CLOCK 1251 1252 1253 1254 1255 12
RESTART
STATE[7:0] 9F 9F 9F 9F 9F
ADDR[23:0] 004000 008000 010000 020000 040000
RTWF 1 1 1 1 1
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 1 1 1 1 1
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000



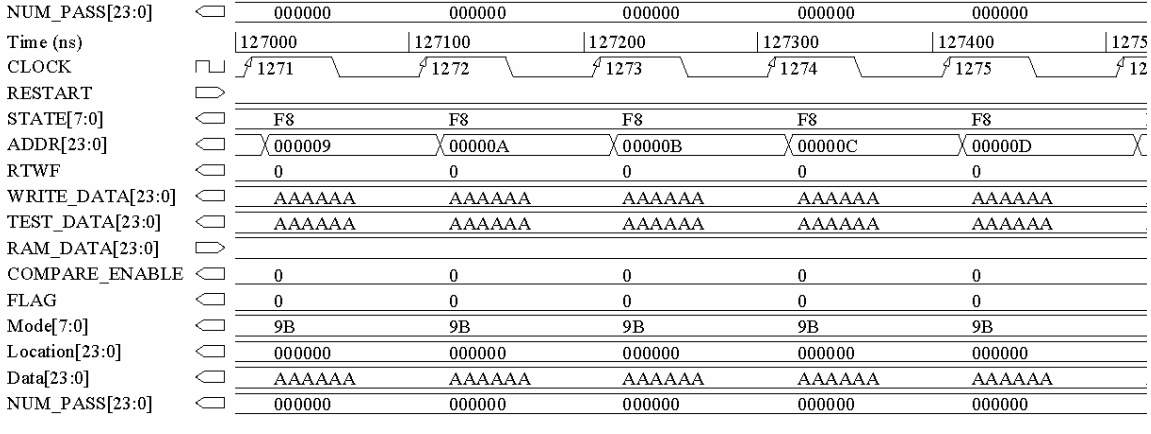
Time (ns)	125500	125600	125700	125800	125900	1260
CLOCK	1256	1257	1258	1259	1260	12
RESTART						
STATE[7:0]	9F	9F	9F	9F	9F	
ADDR[23:0]	080000	100000	200000	400000	800000	
RTWF	1	1	1	1	1	
WRITE_DATA[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
TEST_DATA[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
RAM_DATA[23:0]						
COMPARE_ENABLE	1	1	1	1	1	
FLAG	0	0	0	0	0	
Mode[7:0]	9B	9B	9B	9B	9B	
Location[23:0]	000000	000000	000000	000000	000000	
Data[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
NUM_PASS[23:0]	000000	000000	000000	000000	000000	



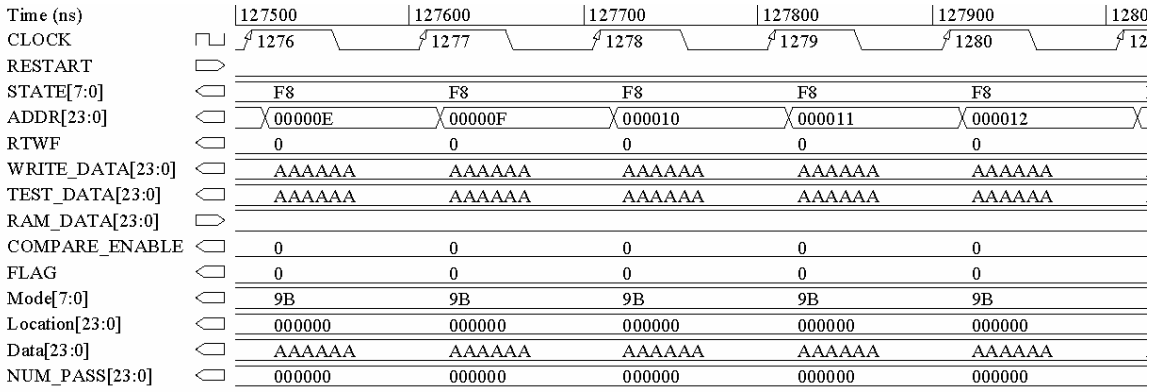
Time (ns)	126000	126100	126200	126300	126400	1265
CLOCK	1261	1262	1263	1264	1265	12
RESTART						
STATE[7:0]	D7	F8	F8	F8	F8	
ADDR[23:0]	800000	000000	000001	000002	000003	
RTWF	1	0	0	0	0	
WRITE_DATA[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
TEST_DATA[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
RAM_DATA[23:0]						
COMPARE_ENABLE	1	0	0	0	0	
FLAG	0	0	0	0	0	
Mode[7:0]	9B	9B	9B	9B	9B	
Location[23:0]	000000	000000	000000	000000	000000	
Data[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
NUM_PASS[23:0]	000000	000000	000000	000000	000000	

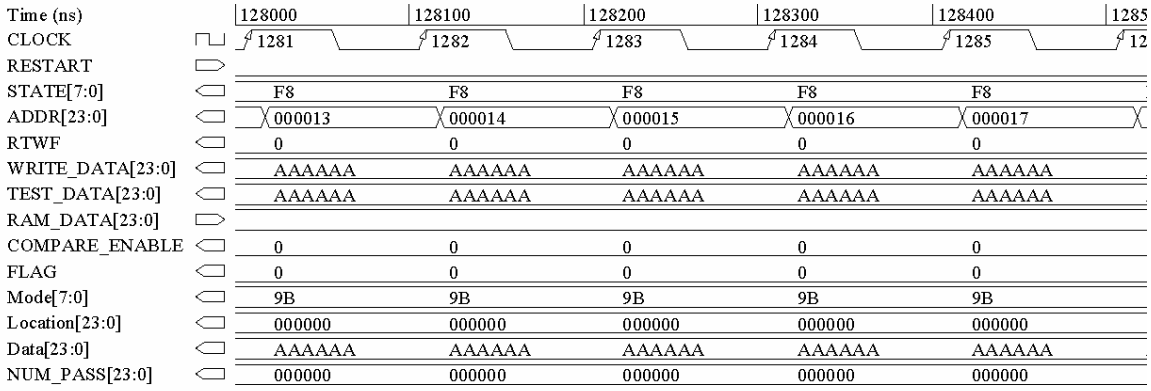


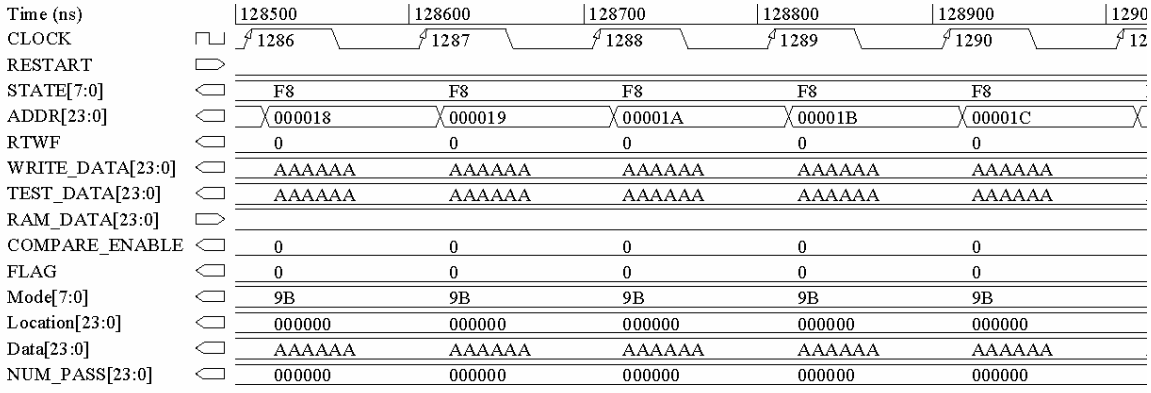
Time (ns)	126500	126600	126700	126800	126900	1270
CLOCK	1266	1267	1268	1269	1270	12
RESTART						
STATE[7:0]	F8	F8	F8	F8	F8	
ADDR[23:0]	000004	000005	000006	000007	000008	
RTWF	0	0	0	0	0	
WRITE_DATA[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
TEST_DATA[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
RAM_DATA[23:0]						
COMPARE_ENABLE	0	0	0	0	0	
FLAG	0	0	0	0	0	
Mode[7:0]	9B	9B	9B	9B	9B	
Location[23:0]	000000	000000	000000	000000	000000	
Data[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
NUM_PASS[23:0]	000000	000000	000000	000000	000000	

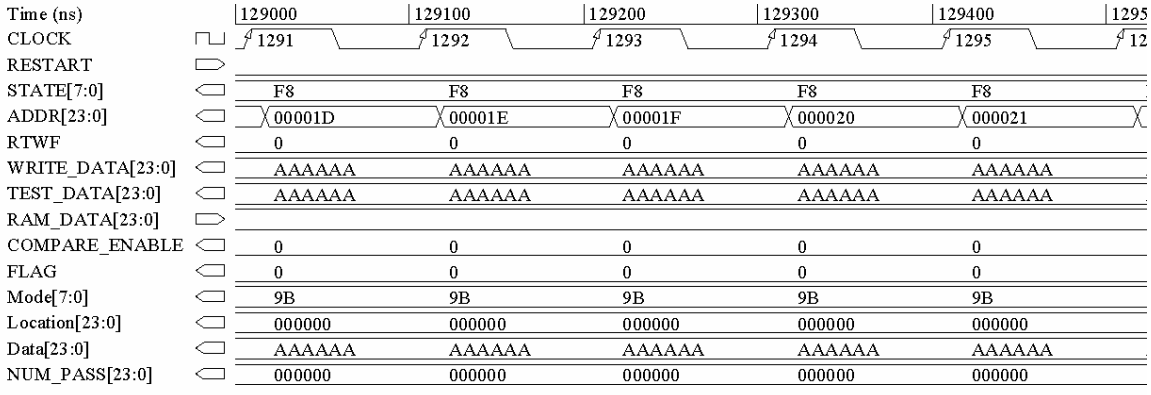


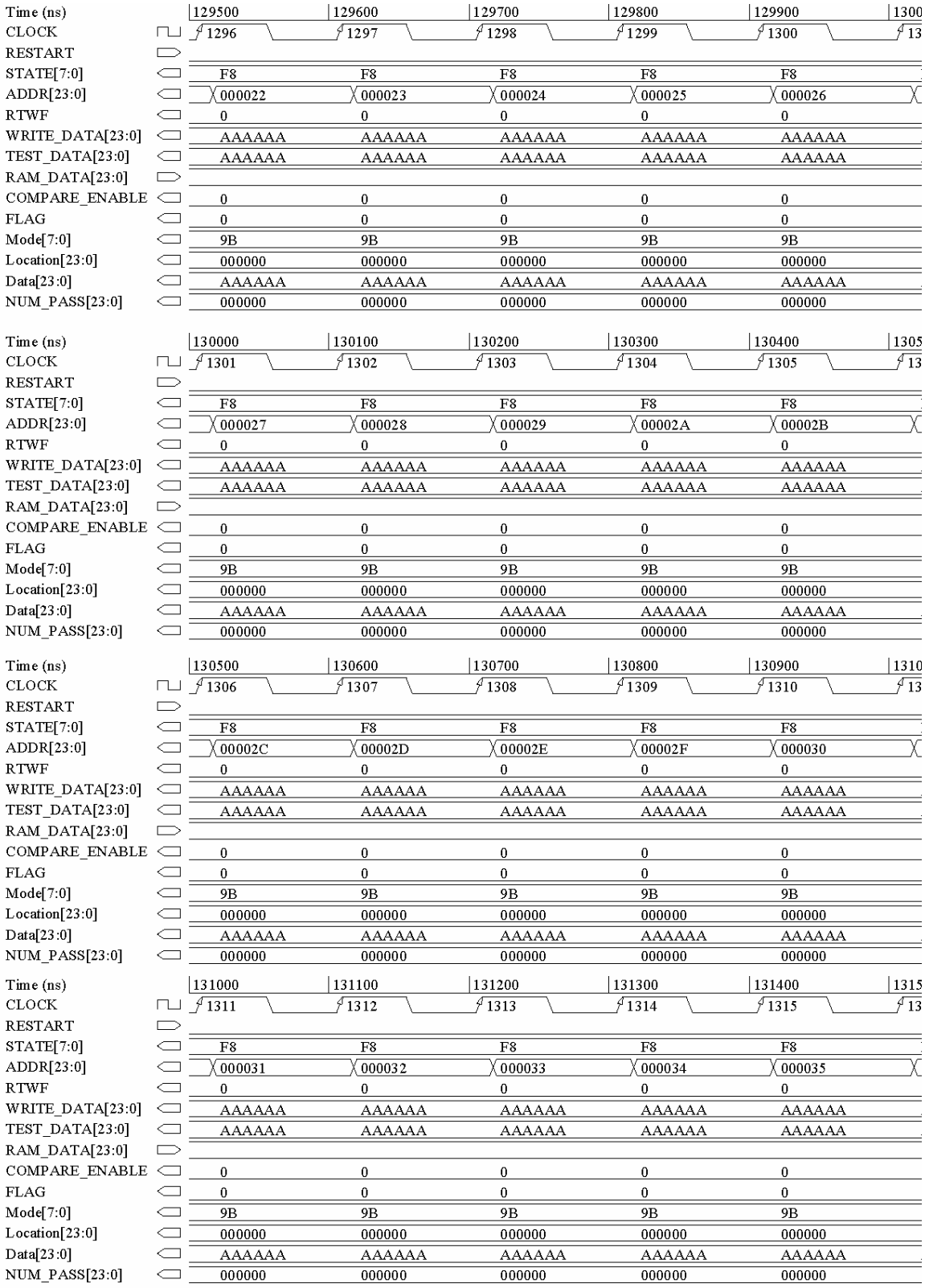
Time (ns)	127000	127100	127200	127300	127400	1275
CLOCK	1271	1272	1273	1274	1275	12
RESTART						
STATE[7:0]	F8	F8	F8	F8	F8	
ADDR[23:0]	000009	00000A	00000B	00000C	00000D	
RTWF	0	0	0	0	0	
WRITE_DATA[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
TEST_DATA[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
RAM_DATA[23:0]						
COMPARE_ENABLE	0	0	0	0	0	
FLAG	0	0	0	0	0	
Mode[7:0]	9B	9B	9B	9B	9B	
Location[23:0]	000000	000000	000000	000000	000000	
Data[23:0]	AAAAAA	AAAAAA	AAAAAA	AAAAAA	AAAAAA	
NUM_PASS[23:0]	000000	000000	000000	000000	000000	

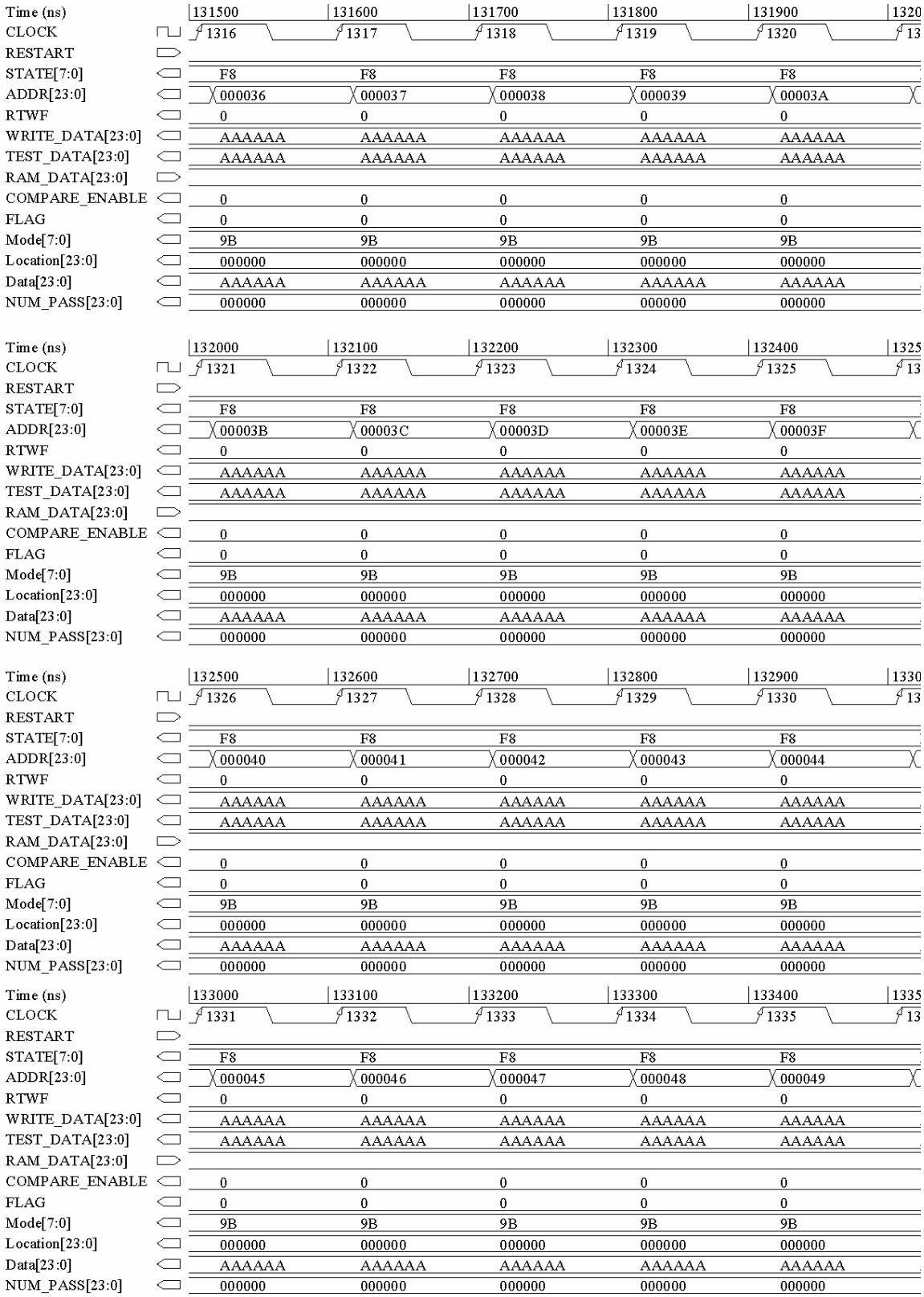

Time (ns) 127500 127600 127700 127800 127900 1280
CLOCK 1276 1277 1278 1279 1280 12
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00000E 00000F 000010 000011 000012
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

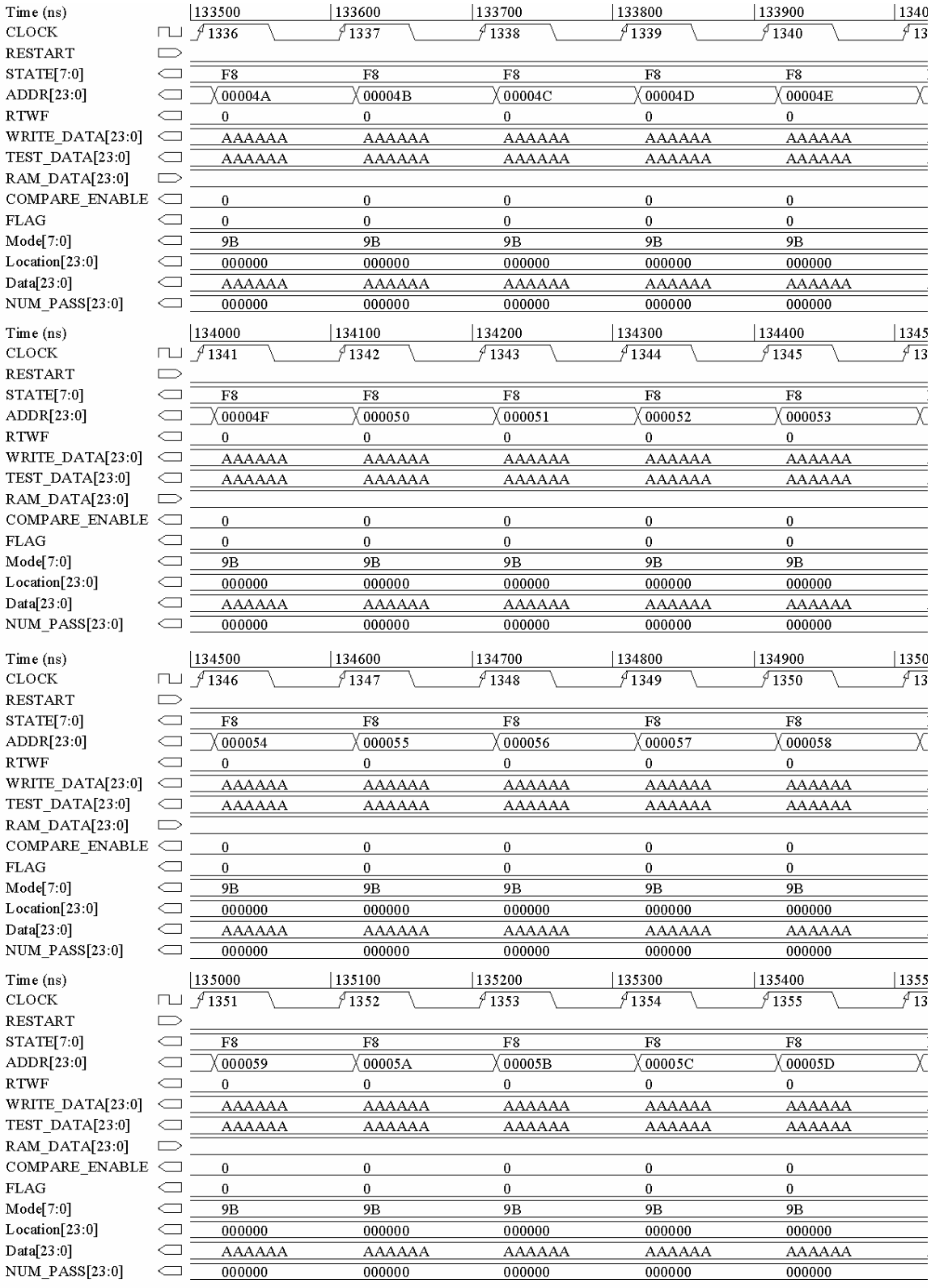

Time (ns) 128000 128100 128200 128300 128400 1285
CLOCK 1281 1282 1283 1284 1285 12
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000013 000014 000015 000016 000017
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

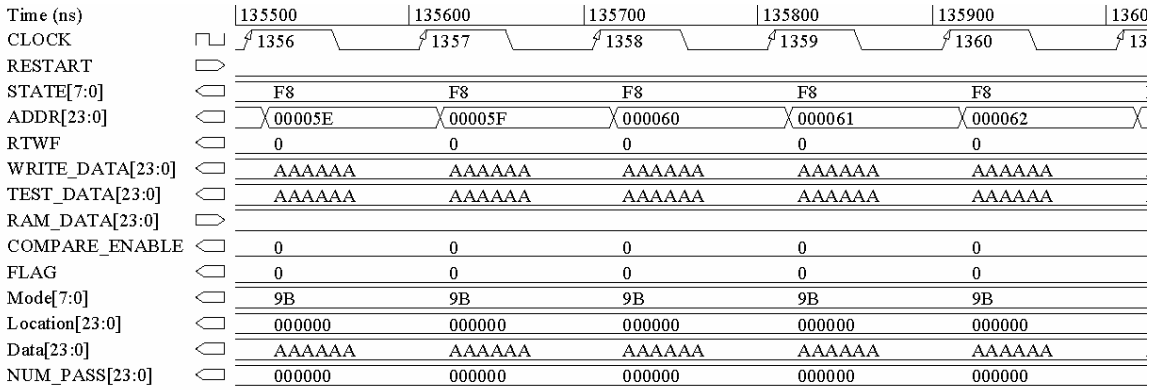

Time (ns) 128500 128600 128700 128800 128900 1290
CLOCK 1286 1287 1288 1289 1290 12
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000018 000019 00001A 00001B 00001C
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

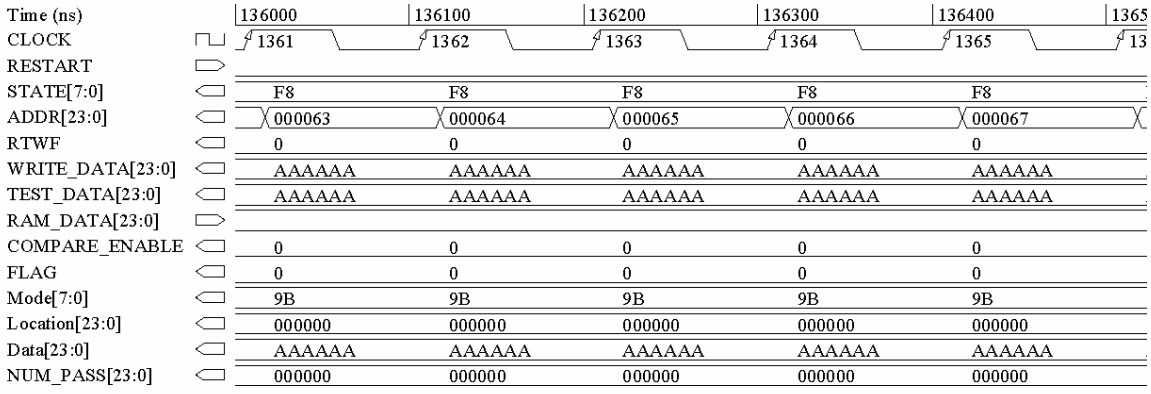

Time (ns) 129000 129100 129200 129300 129400 1295
CLOCK 1291 1292 1293 1294 1295 12
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00001D 00001E 00001F 000020 000021
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

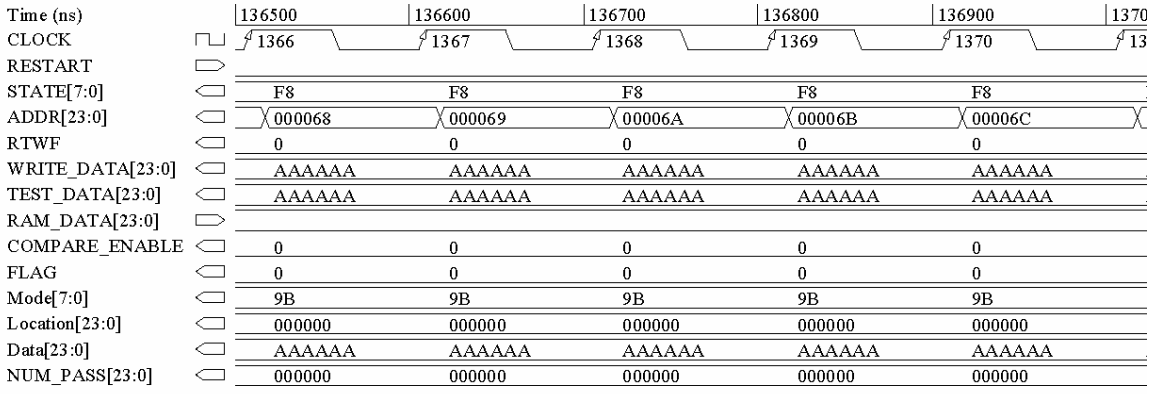

Time (ns) 129500 129600 129700 129800 129900 1300
CLOCK 1296 1297 1298 1299 1300 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000022 000023 000024 000025 000026
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 130000 130100 130200 130300 130400 1305
CLOCK 1301 1302 1303 1304 1305 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000027 000028 000029 00002A 00002B
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 130500 130600 130700 130800 130900 1310
CLOCK 1306 1307 1308 1309 1310 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00002C 00002D 00002E 00002F 000030
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 131000 131100 131200 131300 131400 1315
CLOCK 1311 1312 1313 1314 1315 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000031 000032 000033 000034 000035
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

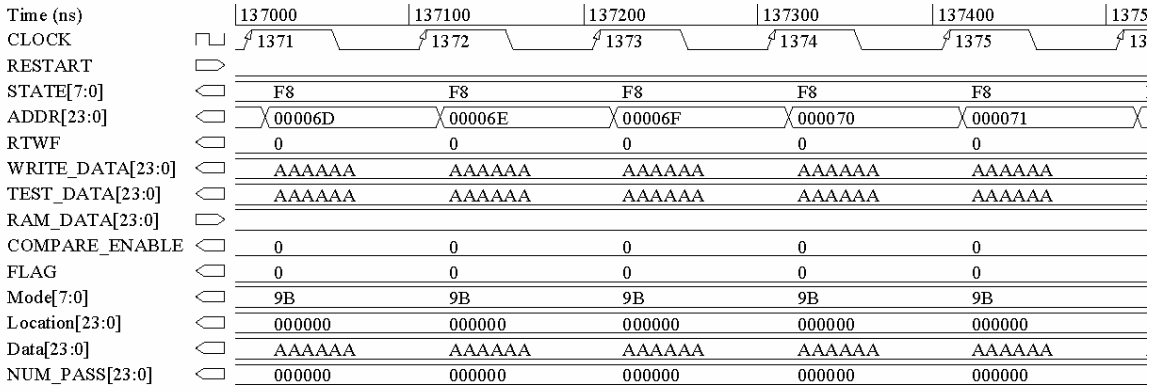

Time (ns) 131500 131600 131700 131800 131900 1320
CLOCK 1316 1317 1318 1319 1320 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000036 000037 000038 000039 00003A
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 132000 132100 132200 132300 132400 1325
CLOCK 1321 1322 1323 1324 1325 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00003B 00003C 00003D 00003E 00003F
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 132500 132600 132700 132800 132900 1330
CLOCK 1326 1327 1328 1329 1330 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000040 000041 000042 000043 000044
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 133000 133100 133200 133300 133400 1335
CLOCK 1331 1332 1333 1334 1335 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000045 000046 000047 000048 000049
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

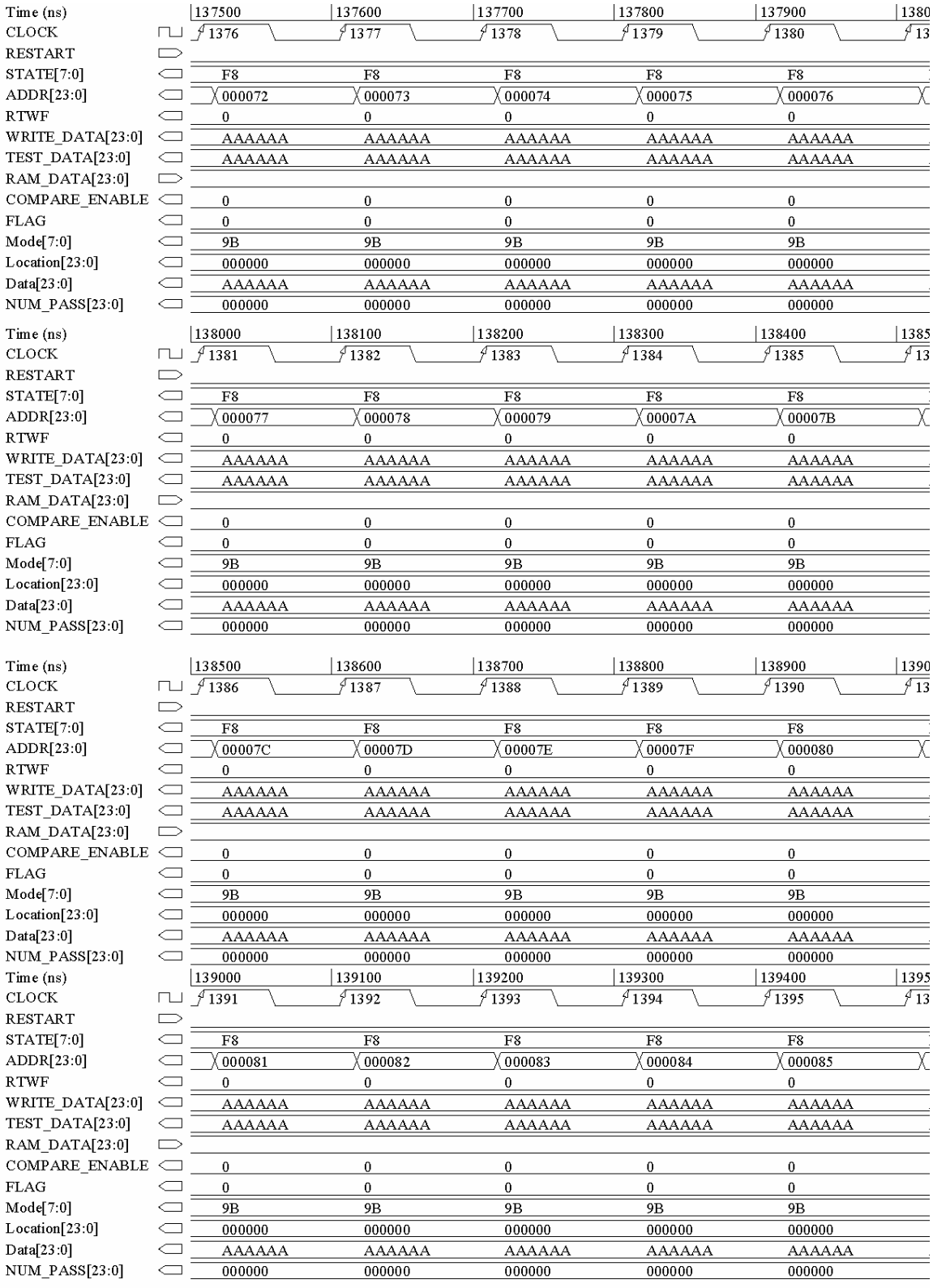

Time (ns) 133500 133600 133700 133800 133900 1340
CLOCK 1336 1337 1338 1339 1340 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00004A 00004B 00004C 00004D 00004E
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 134000 134100 134200 134300 134400 1345
CLOCK 1341 1342 1343 1344 1345 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00004F 000050 000051 000052 000053
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 134500 134600 134700 134800 134900 1350
CLOCK 1346 1347 1348 1349 1350 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000054 000055 000056 000057 000058
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 135000 135100 135200 135300 135400 1355
CLOCK 1351 1352 1353 1354 1355 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000059 00005A 00005B 00005C 00005D
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

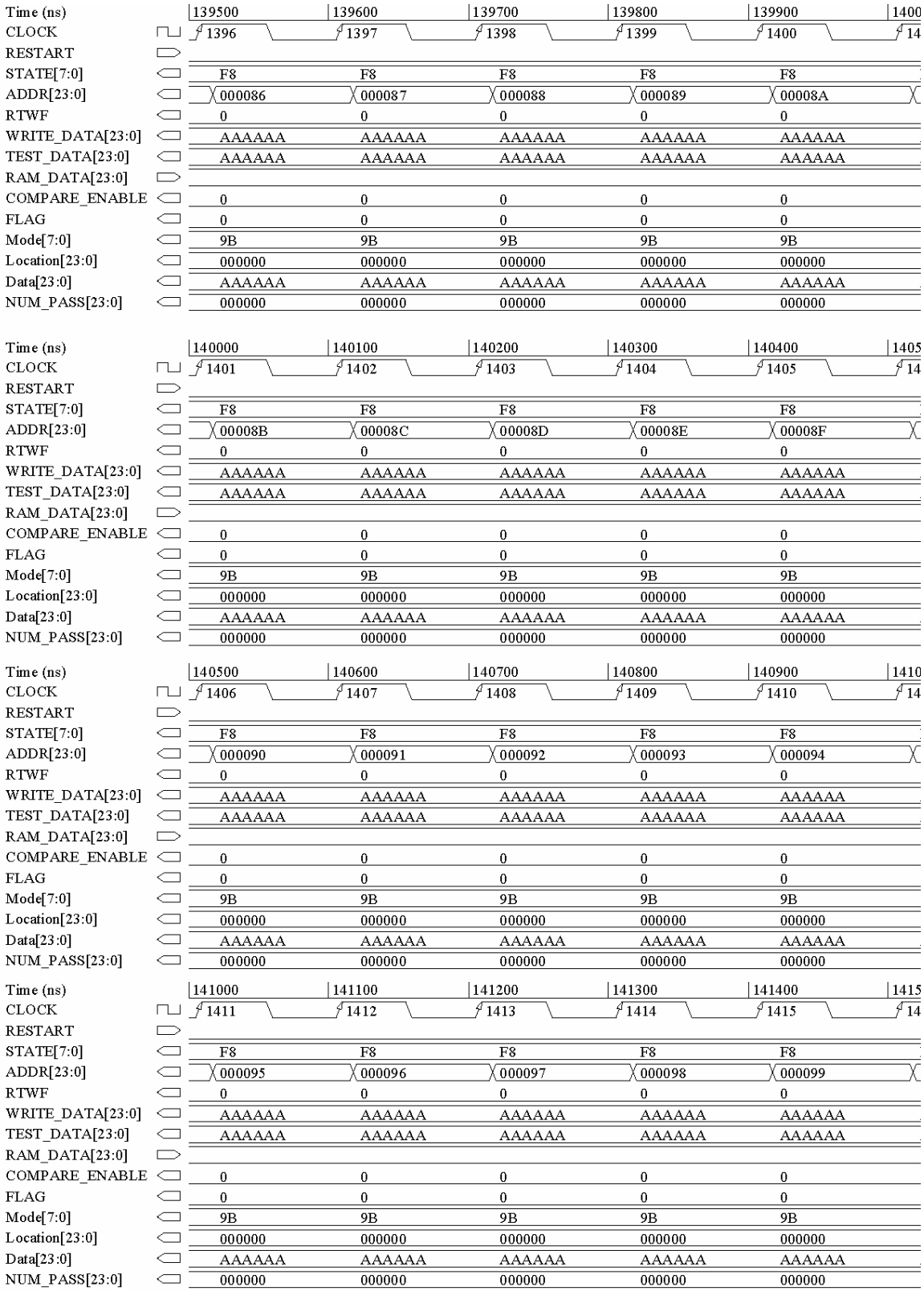

Time (ns) 135500 135600 135700 135800 135900 1360
CLOCK 1356 1357 1358 1359 1360 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00005E 00005F 000060 000061 000062
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

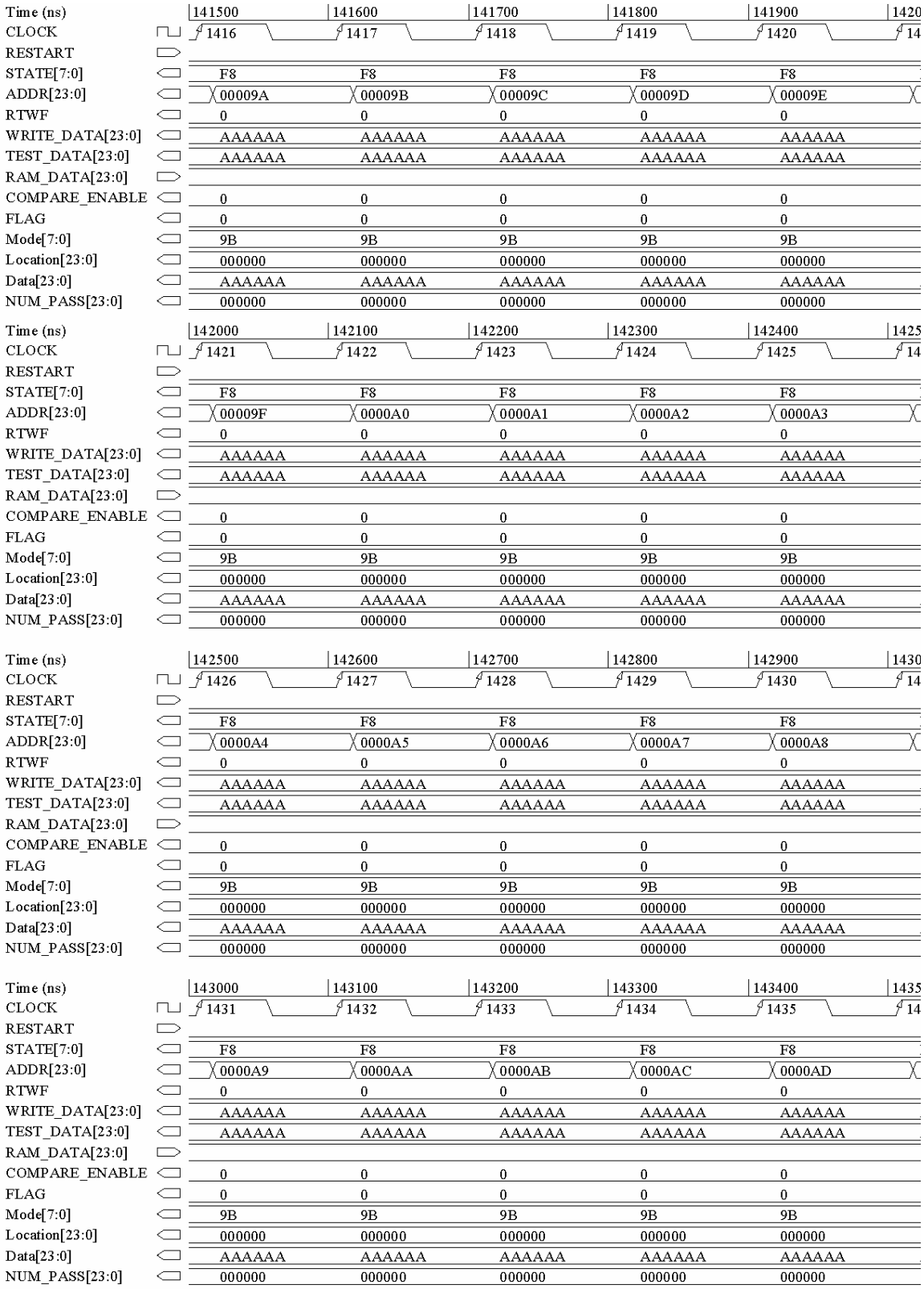

Time (ns) 136000 136100 136200 136300 136400 1365
CLOCK 1361 1362 1363 1364 1365 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000063 000064 000065 000066 000067
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

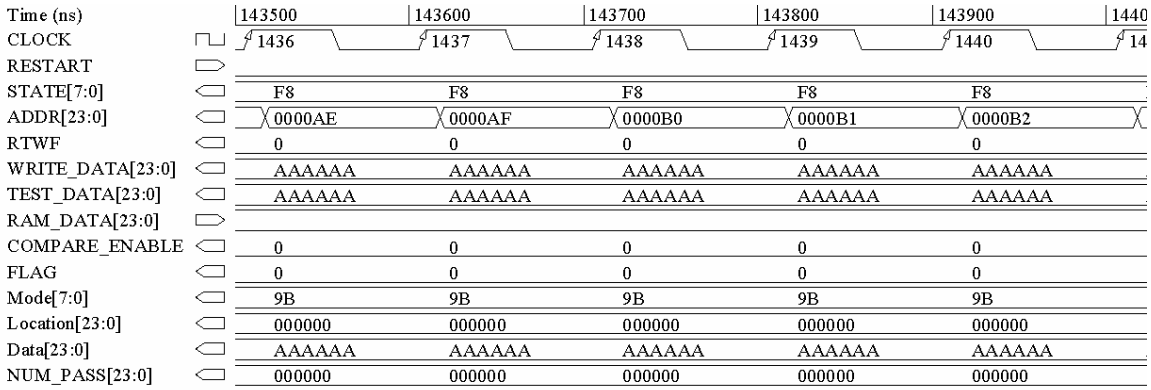

Time (ns) 136500 136600 136700 136800 136900 1370
CLOCK 1366 1367 1368 1369 1370 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000068 000069 00006A 00006B 00006C
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

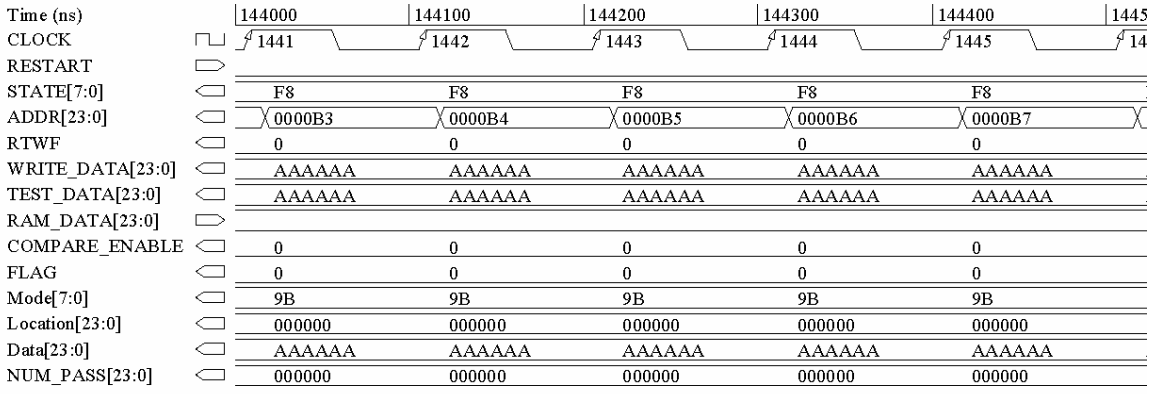

Time (ns) 137000 137100 137200 137300 137400 1375
CLOCK 1371 1372 1373 1374 1375 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00006D 00006E 00006F 000070 000071
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

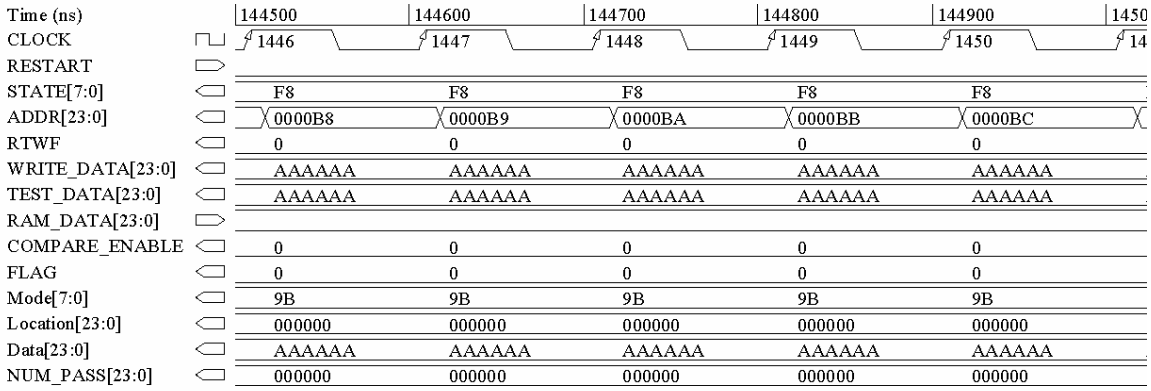

Time (ns) 137500 137600 137700 137800 137900 1380
CLOCK 1376 1377 1378 1379 1380 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000072 000073 000074 000075 000076
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 138000 138100 138200 138300 138400 1385
CLOCK 1381 1382 1383 1384 1385 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000077 000078 000079 00007A 00007B
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 138500 138600 138700 138800 138900 1390
CLOCK 1386 1387 1388 1389 1390 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00007C 00007D 00007E 00007F 000080
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 139000 139100 139200 139300 139400 1395
CLOCK 1391 1392 1393 1394 1395 13
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000081 000082 000083 000084 000085
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

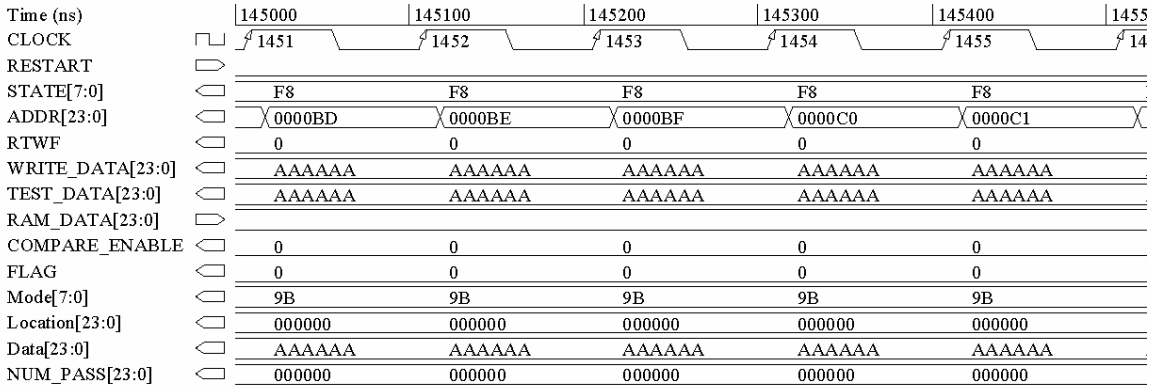

Time (ns) 139500 139600 139700 139800 139900 1400
CLOCK 1396 1397 1398 1399 1400 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000086 000087 000088 000089 00008A
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 140000 140100 140200 140300 140400 1405
CLOCK 1401 1402 1403 1404 1405 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00008B 00008C 00008D 00008E 00008F
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 140500 140600 140700 140800 140900 1410
CLOCK 1406 1407 1408 1409 1410 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000090 000091 000092 000093 000094
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 141000 141100 141200 141300 141400 1415
CLOCK 1411 1412 1413 1414 1415 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 000095 000096 000097 000098 000099
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

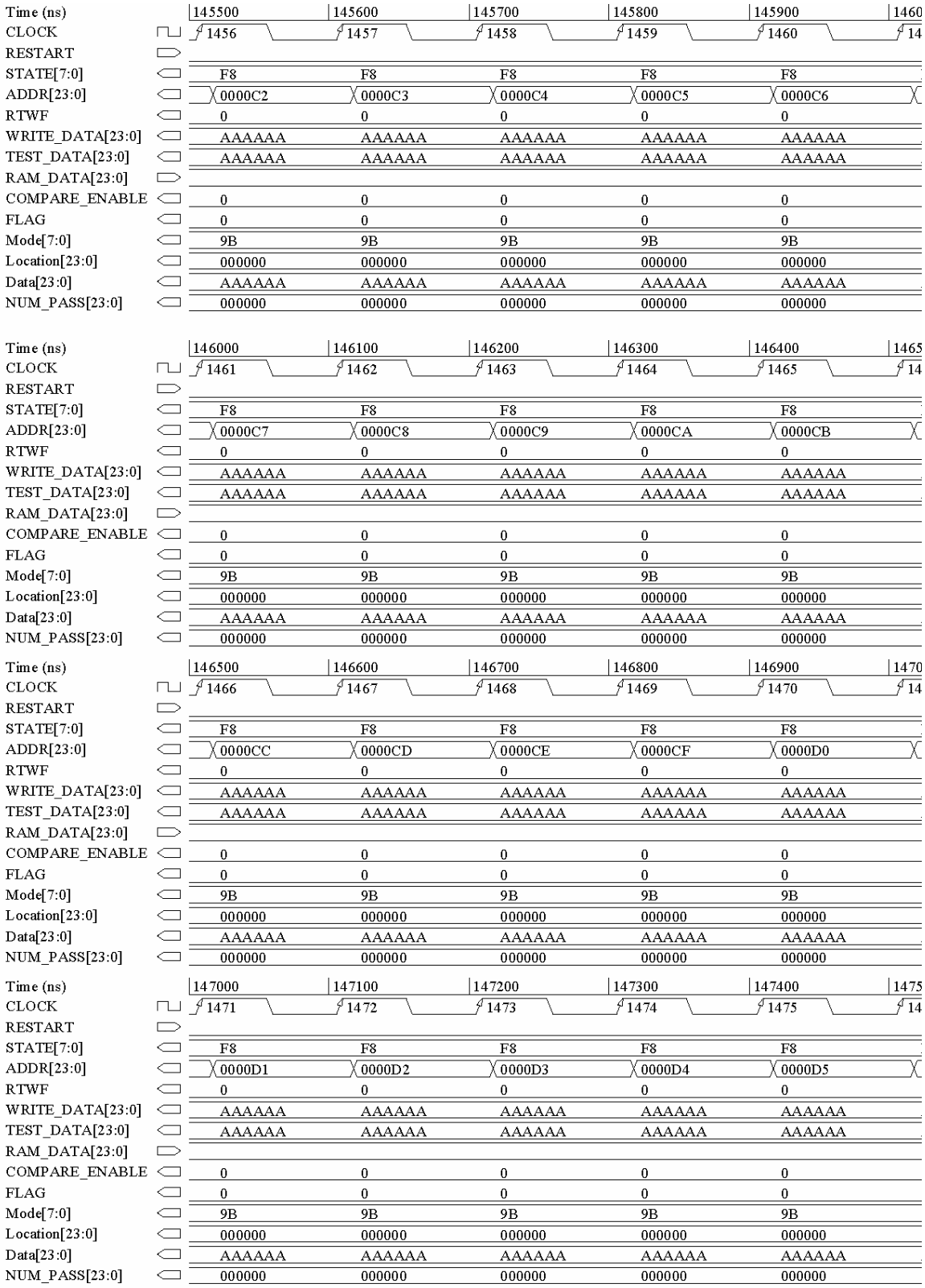

Time (ns) 141500 141600 141700 141800 141900 1420
CLOCK 1416 1417 1418 1419 1420 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00009A 00009B 00009C 00009D 00009E
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 142000 142100 142200 142300 142400 1425
CLOCK 1421 1422 1423 1424 1425 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 00009F 0000A0 0000A1 0000A2 0000A3
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 142500 142600 142700 142800 142900 1430
CLOCK 1426 1427 1428 1429 1430 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000A4 0000A5 0000A6 0000A7 0000A8
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 143000 143100 143200 143300 143400 1435
CLOCK 1431 1432 1433 1434 1435 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000A9 0000AA 0000AB 0000AC 0000AD
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

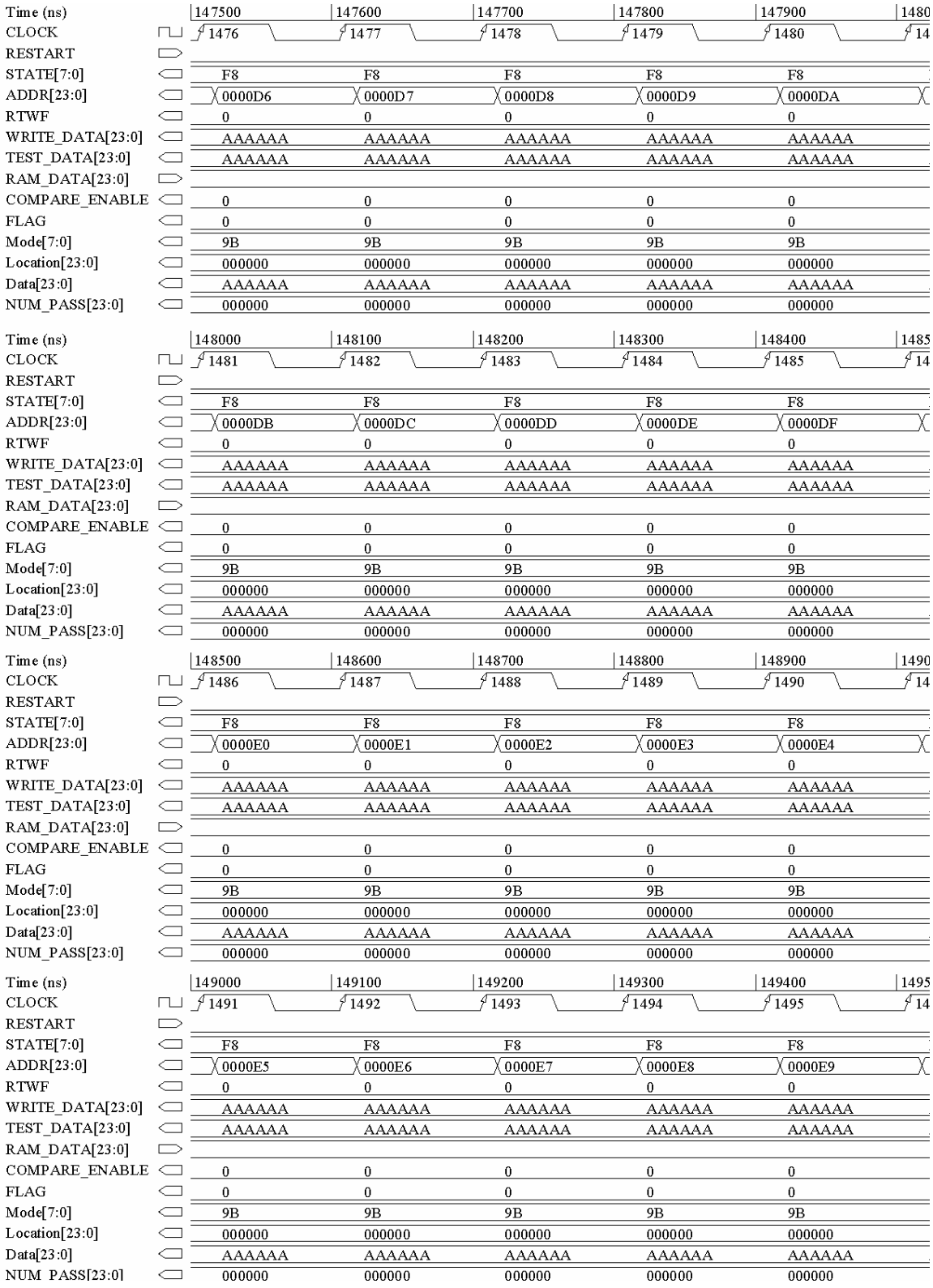

Time (ns) 143500 143600 143700 143800 143900 1440
CLOCK 1436 1437 1438 1439 1440 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000AE 0000AF 0000B0 0000B1 0000B2
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000

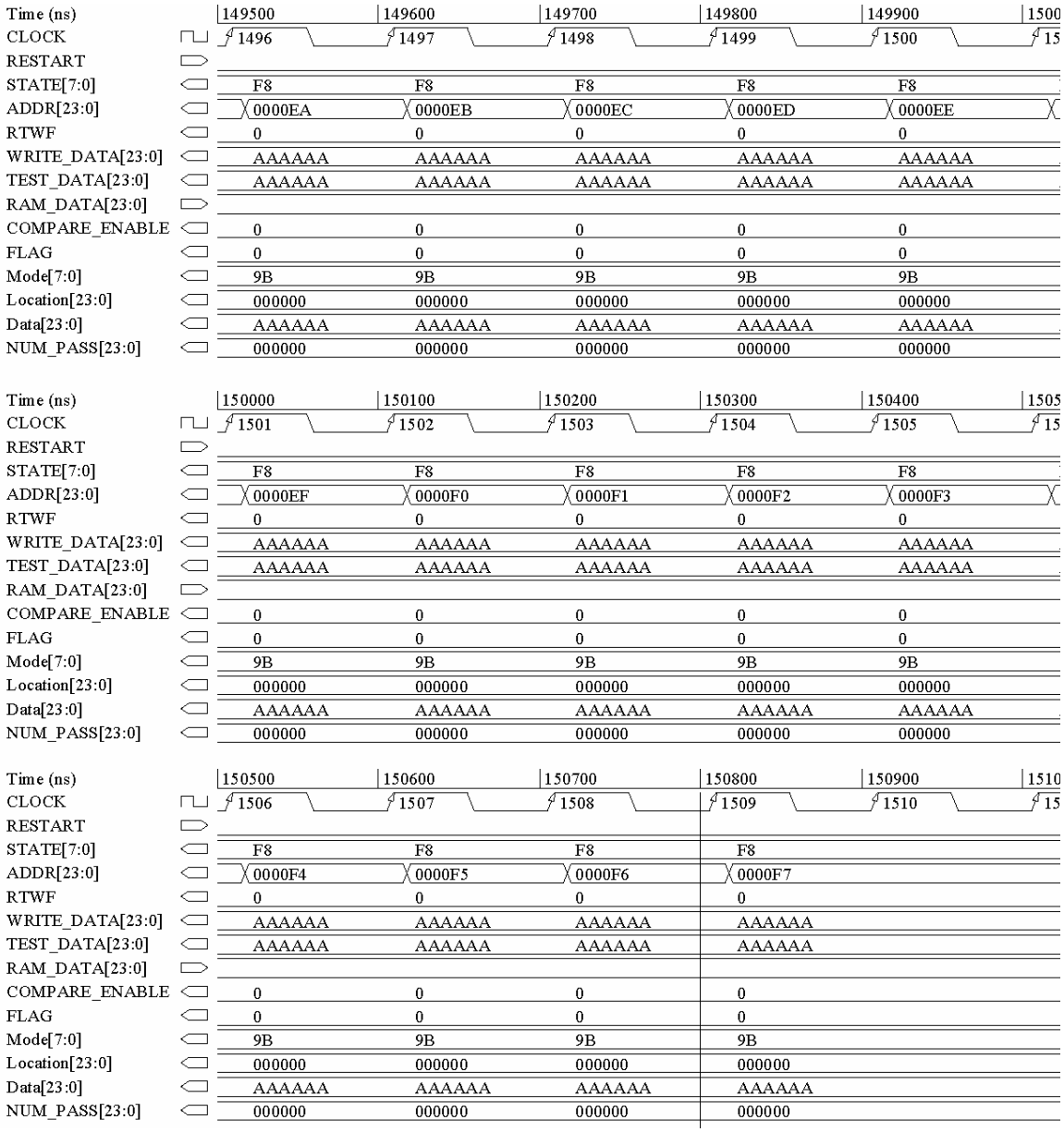

Time (ns) 144000 144100 144200 144300 144400 1445
CLOCK 1441 1442 1443 1444 1445 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000B3 0000B4 0000B5 0000B6 0000B7
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000


Time (ns) 144500 144600 144700 144800 144900 1450
CLOCK 1446 1447 1448 1449 1450 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000B8 0000B9 0000BA 0000BB 0000BC
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000


Time (ns) 145000 145100 145200 145300 145400 1455
CLOCK 1451 1452 1453 1454 1455 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000BD 0000BE 0000BF 0000C0 0000C1
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000


Time (ns) 145500 145600 145700 145800 145900 1460
CLOCK 1456 1457 1458 1459 1460 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000C2 0000C3 0000C4 0000C5 0000C6
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 146000 146100 146200 146300 146400 1465
CLOCK 1461 1462 1463 1464 1465 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000C7 0000C8 0000C9 0000CA 0000CB
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 146500 146600 146700 146800 146900 1470
CLOCK 1466 1467 1468 1469 1470 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000CC 0000CD 0000CE 0000CF 0000D0
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 147000 147100 147200 147300 147400 1475
CLOCK 1471 1472 1473 1474 1475 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000D1 0000D2 0000D3 0000D4 0000D5
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000


Time (ns) 147500 147600 147700 147800 147900 1480
CLOCK 1476 1477 1478 1479 1480 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000D6 0000D7 0000D8 0000D9 0000DA
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 148000 148100 148200 148300 148400 1485
CLOCK 1481 1482 1483 1484 1485 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000DB 0000DC 0000DD 0000DE 0000DF
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 148500 148600 148700 148800 148900 1490
CLOCK 1486 1487 1488 1489 1490 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000E0 0000E1 0000E2 0000E3 0000E4
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000
Time (ns) 149000 149100 149200 149300 149400 1495
CLOCK 1491 1492 1493 1494 1495 14
RESTART
STATE[7:0] F8 F8 F8 F8 F8
ADDR[23:0] 0000E5 0000E6 0000E7 0000E8 0000E9
RTWF 0 0 0 0 0
WRITE_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
TEST_DATA[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
RAM_DATA[23:0]
COMPARE_ENABLE 0 0 0 0 0
FLAG 0 0 0 0 0
Mode[7:0] 9B 9B 9B 9B 9B
Location[23:0] 000000 000000 000000 000000 000000
Data[23:0] AAAAAA AAAAAA AAAAAA AAAAAA AAAAAA
NUM_PASS[23:0] 000000 000000 000000 000000 000000


Time (ns)
149500
149600
149700
149800
149900
1500
CLOCK
1496
1497
1498
1499
1500
15
RESTART
STATE[7:0]
F8
ADDR[23:0]
0000EA
0000EB
0000EC
0000ED
0000EE
RTWF
0
WRITE_DATA[23:0]
AAAAAA
TEST_DATA[23:0]
RAM_DATA[23:0]
COMPARE_ENABLE
FLAG
Mode[7:0]
9B
Location[23:0]
000000
Data[23:0]
NUM_PASS[23:0]
Time (ns)
150000
150100
150200
150300
150400
1505
CLOCK
1501
1502
1503
1504
1505
15
RESTART
STATE[7:0]
F8
ADDR[23:0]
0000EF
0000F0
0000F1
0000F2
0000F3
RTWF
0
WRITE_DATA[23:0]
AAAAAA
TEST_DATA[23:0]
RAM_DATA[23:0]
COMPARE_ENABLE
FLAG
Mode[7:0]
9B
Location[23:0]
000000
Data[23:0]
NUM_PASS[23:0]
Time (ns)
150500
150600
150700
150800
150900
1510
CLOCK
1506
1507
1508
1509
1510
15
RESTART
STATE[7:0]
F8
ADDR[23:0]
0000F4
0000F5
0000F6
0000F7
RTWF
0
WRITE_DATA[23:0]
AAAAAA
TEST_DATA[23:0]
RAM_DATA[23:0]
COMPARE_ENABLE
FLAG
Mode[7:0]
9B
Location[23:0]
000000
Data[23:0]
NUM_PASS[23:0]
000000

B. PATTERN MODULE

1. VHDL Code

```
--------------------------------------------------------------------------
-- filename: Pattern.vhd
-- written by: Charles Hulme
--
-- This Program generates the patterns that will be written
-- to memory or compared to what is stored in memory.
--
-- Which pattern is chosen is driven by the 8 bit input, STATE.
-- Each pattern will use a series of if/then statements to define
-- the pattern sequence.
--
-- The patterns are:
--   Sliding 1's. A one is moved through each
--     of the bit positions in the 24 bit outputs.
--     e.g. 000000000000000000000000000001
--          000000000000000000000000000010
--          000000000000000000000000000100
--                    ...
--          100000000000000000000000000000
--
--   Alternating 1's and 0's. A single bit stream pattern
--    of alternating 1's and 0's (A constant value not a pattern).
--    e.g.  10101010101010101010101010101010
--
--   Alternating 0's and 1's. A single bit stream pattern
--    of alternating 0's and 1's (A constant value not a pattern).
--    e.g.  01010101010101010101010101010101
--
--------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity PATTERN is
    port (
        CLOCK: in STD_LOGIC;
        RESTART: in STD_LOGIC;
        STATE: in STD_LOGIC_VECTOR (7 downto 0);
         FLAG: IN STD_LOGIC;
        TEST_DATA: out STD_LOGIC_VECTOR (23 downto 0);
        WRITE_DATA: out STD_LOGIC_VECTOR (23 downto 0)
       );
end PATTERN;

architecture PATTERN_arch of PATTERN is

begin

FCN: process(RESTART,FLAG,STATE)
```

```
begin

  if RESTART = '1' then
       WRITE_DATA <= "00000000000000000000000000";
       TEST_DATA <= "00000000000000000000000000";
  elsif STATE = "00000000" then  -- In state Wait_State, no pattern
       WRITE_DATA <= "00000000000000000000000000";
       TEST_DATA <= "00000000000000000000000000";
  elsif STATE = "00000001" then  -- In state Delay_Count, no pattern
       WRITE_DATA <= "00000000000000000000000000";
       TEST_DATA <= "00000000000000000000000000";

  --Starting Data Test Pattern
  elsif STATE = "00000010" then  -- writing pattern
       WRITE_DATA <= "00000000000000000000000000";
       TEST_DATA <= "00000000000000000000000000";
  elsif STATE = "00000011" then  -- reading pattern
       WRITE_DATA <= "00000000000000000000000000";
       TEST_DATA <= "00000000000000000000000000";

  elsif STATE = "00000100" then
       WRITE_DATA <= "00000000000000000000000001";
       TEST_DATA <= "00000000000000000000000001";
  elsif STATE = "00000101" then
       WRITE_DATA <= "00000000000000000000000001";
       TEST_DATA <= "00000000000000000000000001";

  elsif STATE = "00000110" then
       WRITE_DATA <= "00000000000000000000000010";
       TEST_DATA <= "00000000000000000000000010";
  elsif STATE = "00000111" then
       WRITE_DATA <= "00000000000000000000000010";
       TEST_DATA <= "00000000000000000000000010";

  elsif STATE = "00001000" then
       WRITE_DATA <= "00000000000000000000000100";
       TEST_DATA <= "00000000000000000000000100";
  elsif STATE = "00001001" then
       WRITE_DATA <= "00000000000000000000000100";
       TEST_DATA <= "00000000000000000000000100";

  elsif STATE = "00001010" then
       WRITE_DATA <= "00000000000000000000001000";
       TEST_DATA <= "00000000000000000000001000";
  elsif STATE = "00001011" then
       WRITE_DATA <= "00000000000000000000001000";
       TEST_DATA <= "00000000000000000000001000";

  elsif STATE = "00001100" then
       WRITE_DATA <= "00000000000000000000010000";
       TEST_DATA <= "00000000000000000000010000";
  elsif STATE = "00001101" then
       WRITE_DATA <= "00000000000000000000010000";
       TEST_DATA <= "00000000000000000000010000";
```

```
elsif STATE = "00001110" then
     WRITE_DATA <= "00000000000000000100000";
     TEST_DATA <= "00000000000000000100000";
elsif STATE = "00001111" then
     WRITE_DATA <= "00000000000000000100000";
     TEST_DATA <= "00000000000000000100000";

elsif STATE = "00010000" then
     WRITE_DATA <= "00000000000000001000000";
     TEST_DATA <= "00000000000000001000000";
elsif STATE = "00010001" then
     WRITE_DATA <= "00000000000000001000000";
     TEST_DATA <= "00000000000000001000000";

elsif STATE = "00010010" then
     WRITE_DATA <= "00000000000000010000000";
     TEST_DATA <= "00000000000000010000000";
elsif STATE = "00010011" then
     WRITE_DATA <= "00000000000000010000000";
     TEST_DATA <= "00000000000000010000000";

elsif STATE = "00010100" then
     WRITE_DATA <= "00000000000000100000000";
     TEST_DATA <= "00000000000000100000000";
elsif STATE = "00010101" then
     WRITE_DATA <= "00000000000000100000000";
     TEST_DATA <= "00000000000000100000000";

elsif STATE = "00010110" then
     WRITE_DATA <= "00000000000001000000000";
     TEST_DATA <= "00000000000001000000000";
elsif STATE = "00010111" then
     WRITE_DATA <= "00000000000001000000000";
     TEST_DATA <= "00000000000001000000000";

elsif STATE = "00011000" then
     WRITE_DATA <= "00000000000010000000000";
     TEST_DATA <= "00000000000010000000000";
elsif STATE = "00011001" then
     WRITE_DATA <= "00000000000010000000000";
     TEST_DATA <= "00000000000010000000000";

elsif STATE = "00011010" then
     WRITE_DATA <= "00000000000100000000000";
     TEST_DATA <= "00000000000100000000000";
elsif STATE = "00011011" then
     WRITE_DATA <= "00000000000100000000000";
     TEST_DATA <= "00000000000100000000000";

elsif STATE = "00011100" then
     WRITE_DATA <= "00000000001000000000000";
     TEST_DATA <= "00000000001000000000000";
elsif STATE = "00011101" then
```

```
     WRITE_DATA <= "00000000000010000000000000";
     TEST_DATA <= "00000000000010000000000000";

elsif STATE = "00011110" then
     WRITE_DATA <= "00000000000100000000000000";
     TEST_DATA <= "00000000000100000000000000";
elsif STATE = "00011111" then
     WRITE_DATA <= "00000000000100000000000000";
     TEST_DATA <= "00000000000100000000000000";

elsif STATE = "00100000" then
     WRITE_DATA <= "00000000001000000000000000";
     TEST_DATA <= "00000000001000000000000000";
elsif STATE = "00100001" then
     WRITE_DATA <= "00000000001000000000000000";
     TEST_DATA <= "00000000001000000000000000";

elsif STATE = "00100010" then
     WRITE_DATA <= "00000000010000000000000000";
     TEST_DATA <= "00000000010000000000000000";
elsif STATE = "00100011" then
     WRITE_DATA <= "00000000010000000000000000";
     TEST_DATA <= "00000000010000000000000000";

elsif STATE = "00100100" then
     WRITE_DATA <= "00000001000000000000000000";
     TEST_DATA <= "00000001000000000000000000";
elsif STATE = "00100101" then
     WRITE_DATA <= "00000001000000000000000000";
     TEST_DATA <= "00000001000000000000000000";

elsif STATE = "00100110" then
     WRITE_DATA <= "00000010000000000000000000";
     TEST_DATA <= "00000010000000000000000000";
elsif STATE = "00100111" then
     WRITE_DATA <= "00000010000000000000000000";
     TEST_DATA <= "00000010000000000000000000";

elsif STATE = "00101000" then
     WRITE_DATA <= "00000100000000000000000000";
     TEST_DATA <= "00000100000000000000000000";
elsif STATE = "00101001" then
     WRITE_DATA <= "00000100000000000000000000";
     TEST_DATA <= "00000100000000000000000000";

elsif STATE = "00101010" then
     WRITE_DATA <= "00001000000000000000000000";
     TEST_DATA <= "00001000000000000000000000";
elsif STATE = "00101011" then
     WRITE_DATA <= "00001000000000000000000000";
     TEST_DATA <= "00001000000000000000000000";

elsif STATE = "00101100" then
     WRITE_DATA <= "00010000000000000000000000";
```

```
     TEST_DATA <= "00010000000000000000000000";
elsif STATE = "00101101" then
     WRITE_DATA <= "00010000000000000000000000";
     TEST_DATA <= "00010000000000000000000000";

elsif STATE = "00101110" then
     WRITE_DATA <= "00100000000000000000000000";
     TEST_DATA <= "00100000000000000000000000";
elsif STATE = "00101111" then
     WRITE_DATA <= "00100000000000000000000000";
     TEST_DATA <= "00100000000000000000000000";

elsif STATE = "00110000" then
     WRITE_DATA <= "01000000000000000000000000";
     TEST_DATA <= "01000000000000000000000000";
elsif STATE = "00110001" then
     WRITE_DATA <= "01000000000000000000000000";
     TEST_DATA <= "01000000000000000000000000";

elsif STATE = "00110010" then
     WRITE_DATA <= "10000000000000000000000000";
     TEST_DATA <= "10000000000000000000000000";
elsif STATE = "00110011" then
     WRITE_DATA <= "10000000000000000000000000";
     TEST_DATA <= "10000000000000000000000000";  --Last pattern in
                                                 -- sliding 1s

--Starting Address Pattern
elsif STATE = "00111110" then                  -- In state
     WRITE_DATA <= "00000000000000000000000000";-- Wait_State1
     TEST_DATA <= "00000000000000000000000000";
elsif STATE = "00111111" then                  -- In state
     WRITE_DATA <= "10101010101010101010101010";--Delay_Count1 thru 4
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01000000" then                  -- In state
     WRITE_DATA <= "10101010101010101010101010";-- Address_W1
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01000001" then                  -- In state
     WRITE_DATA <= "10101010101010101010101010";-- Address_WL1
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01000010" then                  -- In state
     WRITE_DATA <= "01010101010101010101010101";-- Address_WI1
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01000011" then                  -- In state
     WRITE_DATA <= "10101010101010101010101010";-- Address_R1
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11000000" then                  -- In state
     WRITE_DATA <= "10101010101010101010101010";-- Address_RL1
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01000100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01000101" then
     WRITE_DATA <= "10101010101010101010101010";
```

```
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01000110" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01000111" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11000001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01001000" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01001001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01001010" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01001011" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11000010" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01001100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01001101" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01001110" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01001111" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11000011" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01010000" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01010001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01010010" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01010011" then
     WRITE_DATA <= "10101010101010101010101010";
```

```
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11000100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01010100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01010101" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01010110" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01010111" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11000101" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01011000" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01011001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10011010" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01011011" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11000110" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01011100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01011101" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01011110" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01011111" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11000111" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01100000" then
```

```
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01100001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01100010" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01100011" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11001000" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01100100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01100101" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01100110" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01100111" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11001001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01101000" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01101001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01101010" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01101011" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11001010" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01101100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01101101" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01101110" then
```

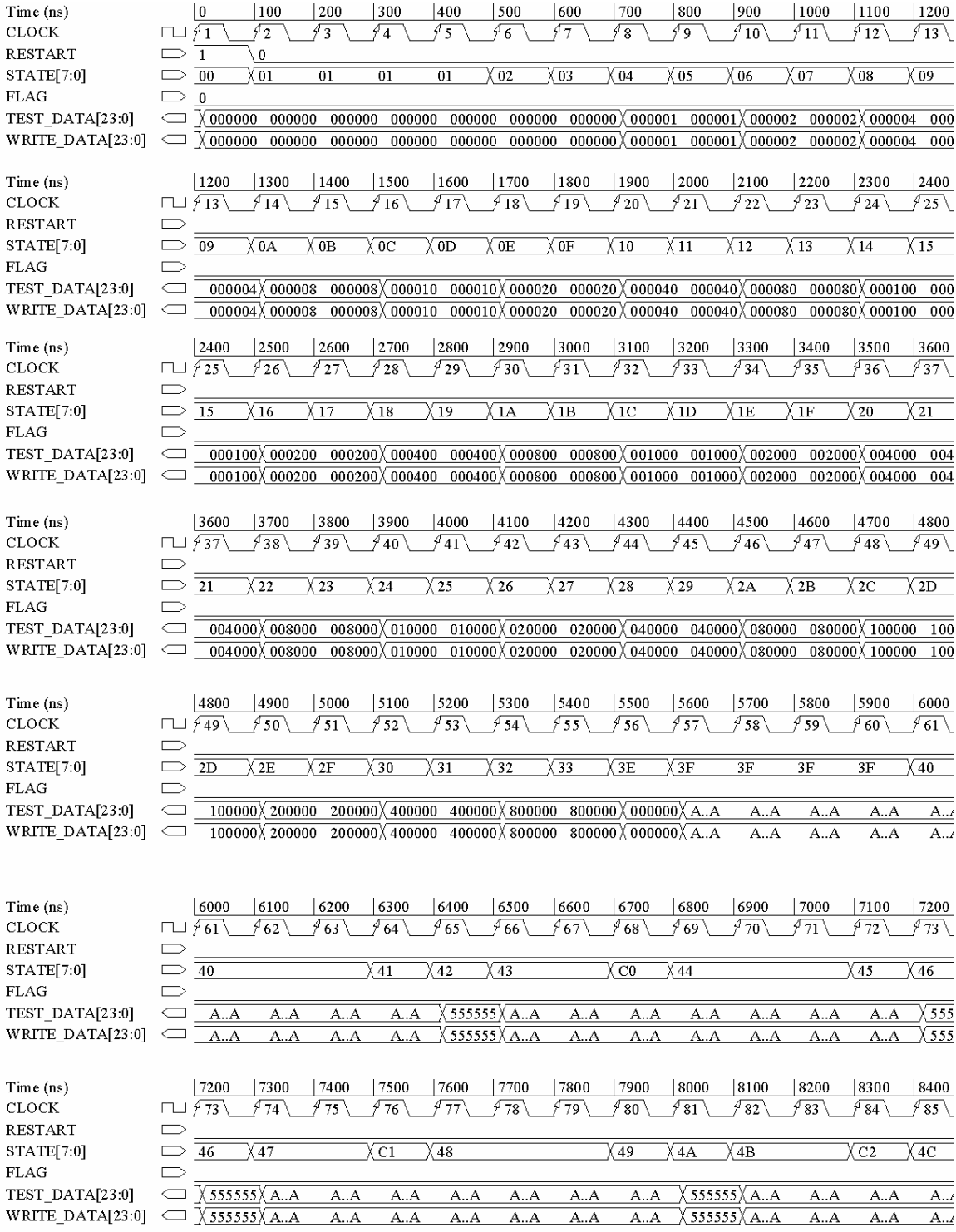
```
     WRITE_DATA <= "010101010101010101010101";
     TEST_DATA <= "010101010101010101010101";
elsif STATE = "01101111" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";
elsif STATE = "11001011" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";

elsif STATE = "01110000" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";
elsif STATE = "01110001" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";
elsif STATE = "01110010" then
     WRITE_DATA <= "010101010101010101010101";
     TEST_DATA <= "010101010101010101010101";
elsif STATE = "01110011" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";
elsif STATE = "11001100" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";

elsif STATE = "01110100" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";
elsif STATE = "01110101" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";
elsif STATE = "01110110" then
     WRITE_DATA <= "010101010101010101010101";
     TEST_DATA <= "010101010101010101010101";
elsif STATE = "01110111" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";
elsif STATE = "11001101" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";

elsif STATE = "01111000" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";
elsif STATE = "01111001" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";
elsif STATE = "01111010" then
     WRITE_DATA <= "010101010101010101010101";
     TEST_DATA <= "010101010101010101010101";
elsif STATE = "01111011" then
     WRITE_DATA <= "101010101010101010101010";
     TEST_DATA <= "101010101010101010101010";
elsif STATE = "11001110" then
```

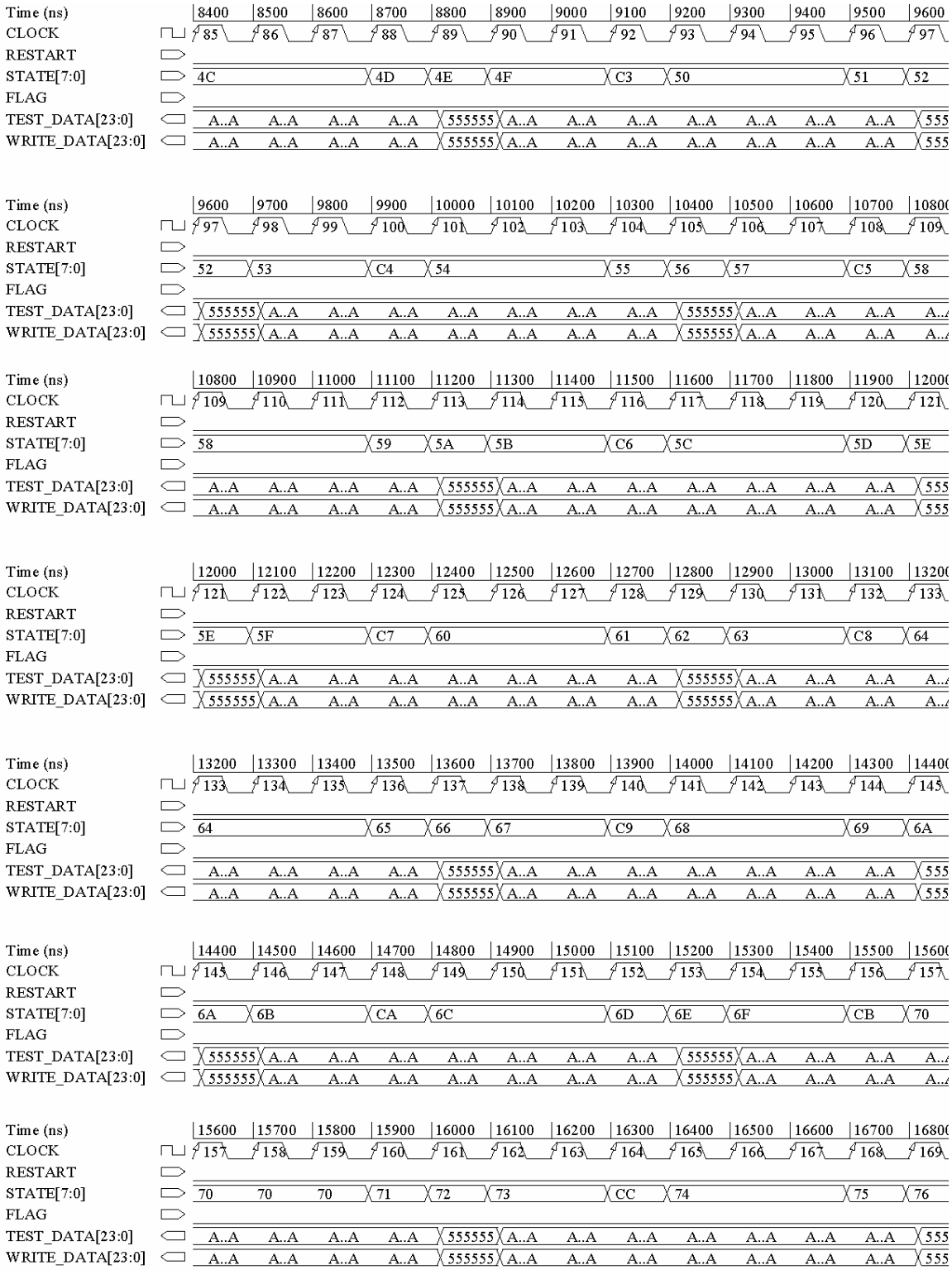
```
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "01111100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01111101" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "01111110" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "01111111" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11001111" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "10000000" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10000001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10000010" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "10000011" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11010000" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "10000100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10000101" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10000110" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "10000111" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11010001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "10001000" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
```

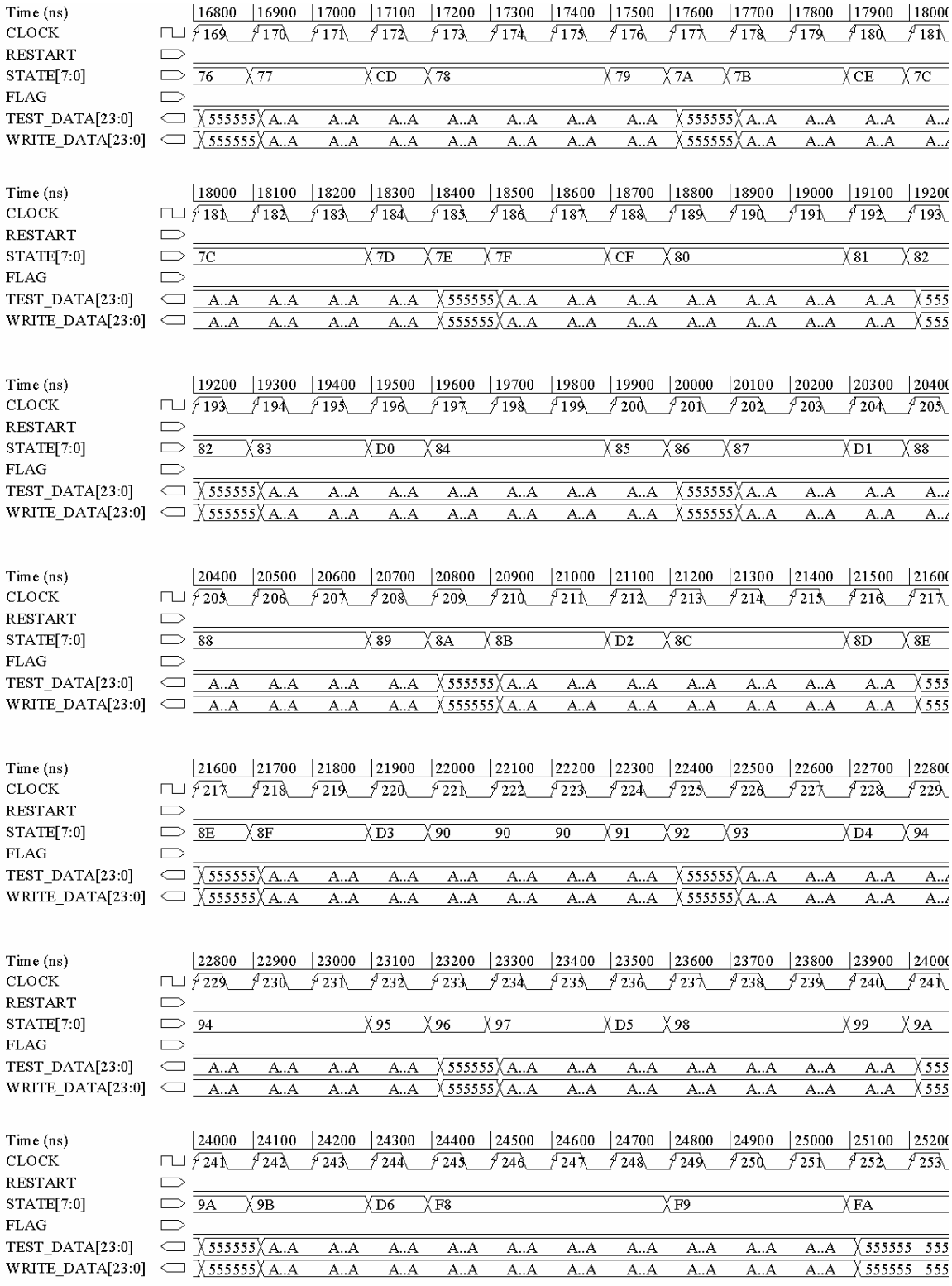
```
elsif STATE = "10001001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10001010" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "10001011" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11010010" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "10001100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10001101" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10001110" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "10001111" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11010011" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "10010000" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10010001" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10010010" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
elsif STATE = "10010011" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "11010100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";

elsif STATE = "10010100" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "10010101" then
     WRITE_DATA <= "10101010101010101010101010";
     TEST_DATA <= "10101010101010101010101010";
elsif STATE = "1010110" then
     WRITE_DATA <= "01010101010101010101010101";
     TEST_DATA <= "01010101010101010101010101";
```

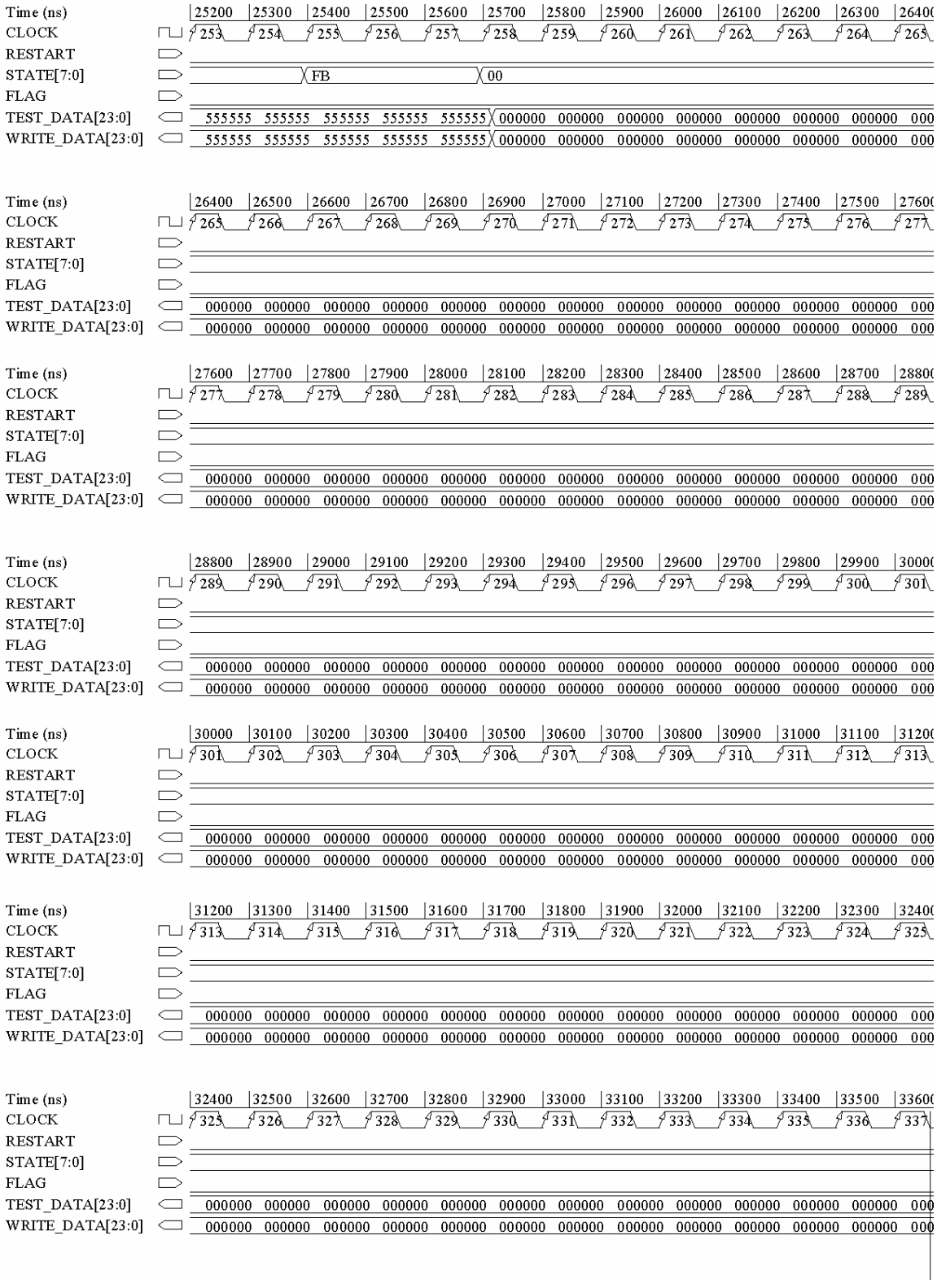
```
    elsif STATE = "10010111" then
         WRITE_DATA <= "10101010101010101010101010";
         TEST_DATA <= "10101010101010101010101010";
    elsif STATE = "11010101" then
         WRITE_DATA <= "10101010101010101010101010";
         TEST_DATA <= "10101010101010101010101010";

    elsif STATE = "10011000" then
         WRITE_DATA <= "10101010101010101010101010";
         TEST_DATA <= "10101010101010101010101010";
    elsif STATE = "10011001" then
         WRITE_DATA <= "10101010101010101010101010";
         TEST_DATA <= "10101010101010101010101010";
    elsif STATE = "10011010" then
         WRITE_DATA <= "01010101010101010101010101";
         TEST_DATA <= "01010101010101010101010101";
    elsif STATE = "10011011" then
         WRITE_DATA <= "10101010101010101010101010";
         TEST_DATA <= "10101010101010101010101010";
    elsif STATE = "11010110" then
         WRITE_DATA <= "10101010101010101010101010";
         TEST_DATA <= "10101010101010101010101010";

    --Starting Memory Test
    elsif STATE = "11111000" then                -- In state
         WRITE_DATA <= "10101010101010101010101010";-- Memory_WU
         TEST_DATA <= "10101010101010101010101010";

    elsif STATE = "11111001" then                -- In state
         WRITE_DATA <= "10101010101010101010101010";-- Memory_RU
         TEST_DATA <= "10101010101010101010101010";

    elsif STATE = "11111010" then                -- In state
         WRITE_DATA <= "01010101010101010101010101";-- Memory_WD
         TEST_DATA <= "01010101010101010101010101";

    elsif STATE = "11111011" then                -- In state
         WRITE_DATA <= "01010101010101010101010101";-- Memory_RD
         TEST_DATA <= "01010101010101010101010101";

  --elsif STATE = "11111000" then  -- In state Pass, no pattern
  --elsif STATE = "11111111" then  -- In state Freeze, no pattern

    end if;

end process FCN;


end PATTERN_arch;
```

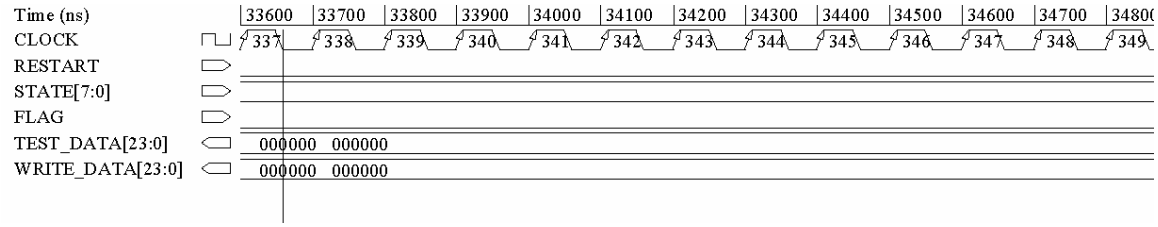
2. Test-Bench Waveform




Time (ns) 8400 8500 8600 8700 8800 8900 9000 9100 9200 9300 9400 9500 9600
CLOCK 85 86 87 88 89 90 91 92 93 94 95 96 97
RESTART
STATE[7:0] 4C 4D 4E 4F C3 50 51 52
FLAG
TEST_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
WRITE_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
Time (ns) 9600 9700 9800 9900 10000 10100 10200 10300 10400 10500 10600 10700 10800
CLOCK 97 98 99 100 101 102 103 104 105 106 107 108 109
RESTART
STATE[7:0] 52 53 C4 54 55 56 57 C5 58
FLAG
TEST_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
WRITE_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
Time (ns) 10800 10900 11000 11100 11200 11300 11400 11500 11600 11700 11800 11900 12000
CLOCK 109 110 111 112 113 114 115 116 117 118 119 120 121
RESTART
STATE[7:0] 58 59 5A 5B C6 5C 5D 5E
FLAG
TEST_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
WRITE_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
Time (ns) 12000 12100 12200 12300 12400 12500 12600 12700 12800 12900 13000 13100 13200
CLOCK 121 122 123 124 125 126 127 128 129 130 131 132 133
RESTART
STATE[7:0] 5E 5F C7 60 61 62 63 C8 64
FLAG
TEST_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
WRITE_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
Time (ns) 13200 13300 13400 13500 13600 13700 13800 13900 14000 14100 14200 14300 14400
CLOCK 133 134 135 136 137 138 139 140 141 142 143 144 145
RESTART
STATE[7:0] 64 65 66 67 C9 68 69 6A
FLAG
TEST_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
WRITE_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
Time (ns) 14400 14500 14600 14700 14800 14900 15000 15100 15200 15300 15400 15500 15600
CLOCK 145 146 147 148 149 150 151 152 153 154 155 156 157
RESTART
STATE[7:0] 6A 6B CA 6C 6D 6E 6F CB 70
FLAG
TEST_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
WRITE_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
Time (ns) 15600 15700 15800 15900 16000 16100 16200 16300 16400 16500 16600 16700 16800
CLOCK 157 158 159 160 161 162 163 164 165 166 167 168 169
RESTART
STATE[7:0] 70 70 70 71 72 73 CC 74 75 76
FLAG
TEST_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
WRITE_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555


Time (ns) 16800 16900 17000 17100 17200 17300 17400 17500 17600 17700 17800 17900 18000
CLOCK 169 170 171 172 173 174 175 176 177 178 179 180 181
RESTART
STATE[7:0] 76 77 CD 78 79 7A 7B CE 7C
FLAG
TEST_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
WRITE_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
Time (ns) 18000 18100 18200 18300 18400 18500 18600 18700 18800 18900 19000 19100 19200
CLOCK 181 182 183 184 185 186 187 188 189 190 191 192 193
RESTART
STATE[7:0] 7C 7D 7E 7F CF 80 81 82
FLAG
TEST_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
WRITE_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
Time (ns) 19200 19300 19400 19500 19600 19700 19800 19900 20000 20100 20200 20300 20400
CLOCK 193 194 195 196 197 198 199 200 201 202 203 204 205
RESTART
STATE[7:0] 82 83 D0 84 85 86 87 D1 88
FLAG
TEST_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
WRITE_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
Time (ns) 20400 20500 20600 20700 20800 20900 21000 21100 21200 21300 21400 21500 21600
CLOCK 205 206 207 208 209 210 211 212 213 214 215 216 217
RESTART
STATE[7:0] 88 89 8A 8B D2 8C 8D 8E
FLAG
TEST_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
WRITE_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
Time (ns) 21600 21700 21800 21900 22000 22100 22200 22300 22400 22500 22600 22700 22800
CLOCK 217 218 219 220 221 222 223 224 225 226 227 228 229
RESTART
STATE[7:0] 8E 8F D3 90 90 90 91 92 93 D4 94
FLAG
TEST_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
WRITE_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A 555555 A..A A..A A..A A..
Time (ns) 22800 22900 23000 23100 23200 23300 23400 23500 23600 23700 23800 23900 24000
CLOCK 229 230 231 232 233 234 235 236 237 238 239 240 241
RESTART
STATE[7:0] 94 95 96 97 D5 98 99 9A
FLAG
TEST_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
WRITE_DATA[23:0] A..A A..A A..A A..A 555555 A..A A..A A..A A..A A..A A..A A..A 555
Time (ns) 24000 24100 24200 24300 24400 24500 24600 24700 24800 24900 25000 25100 25200
CLOCK 241 242 243 244 245 246 247 248 249 250 251 252 253
RESTART
STATE[7:0] 9A 9B D6 F8 F9 FA
FLAG
TEST_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A A..A A..A A..A 555555 555
WRITE_DATA[23:0] 555555 A..A A..A A..A A..A A..A A..A A..A A..A A..A A..A 555555 555

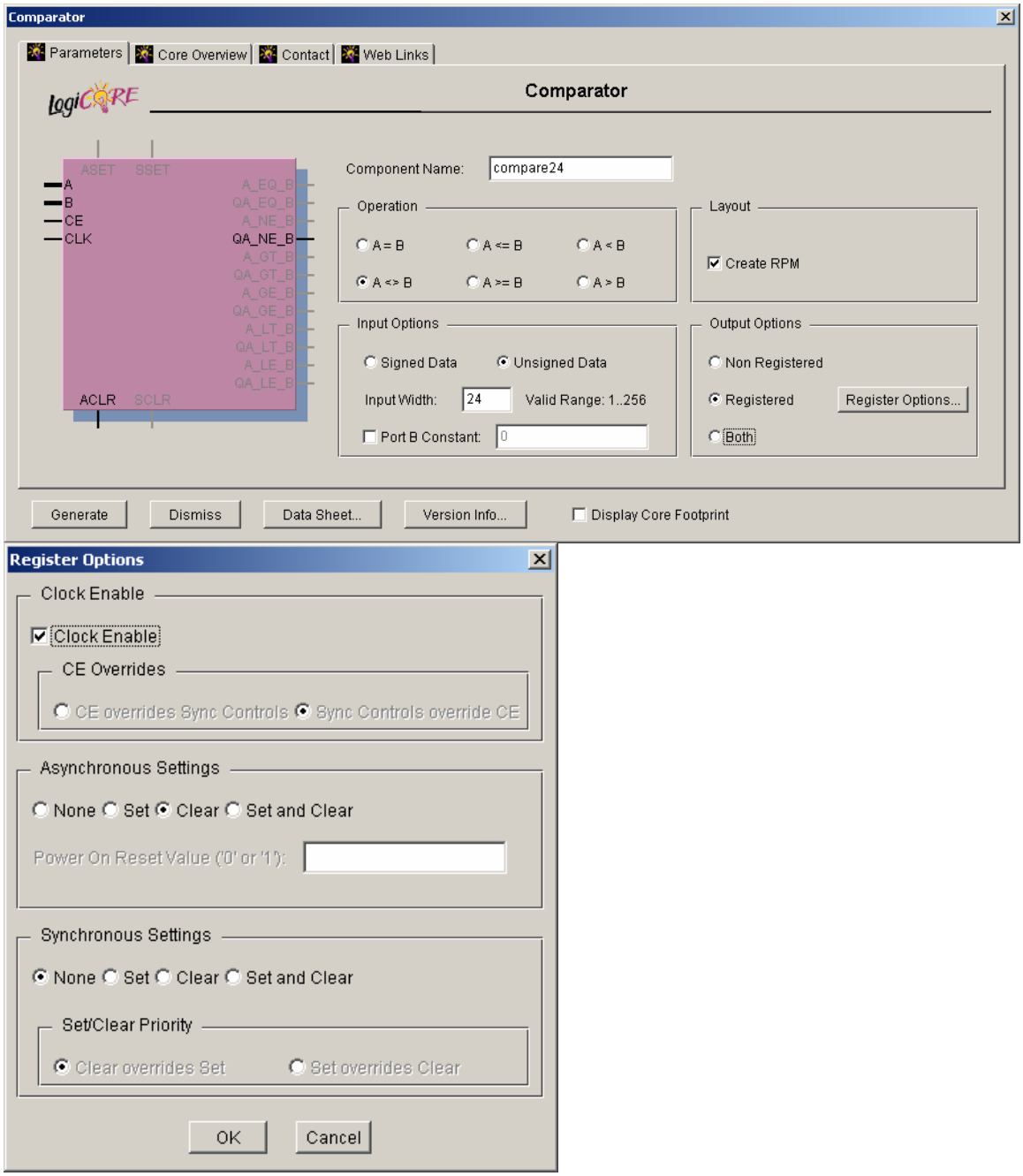

Time (ns)
25200 25300 25400 25500 25600 25700 25800 25900 26000 26100 26200 26300 26400
CLOCK
253 254 255 256 257 258 259 260 261 262 263 264 265
RESTART
STATE[7:0]
FB
00
FLAG
TEST_DATA[23:0]
555555 555555 555555 555555 555555 000000 000000 000000 000000 000000 000000 000000 000
WRITE_DATA[23:0]
555555 555555 555555 555555 555555 000000 000000 000000 000000 000000 000000 000000 000
Time (ns)
26400 26500 26600 26700 26800 26900 27000 27100 27200 27300 27400 27500 27600
CLOCK
265 266 267 268 269 270 271 272 273 274 275 276 277
RESTART
STATE[7:0]
FLAG
TEST_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
WRITE_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
Time (ns)
27600 27700 27800 27900 28000 28100 28200 28300 28400 28500 28600 28700 28800
CLOCK
277 278 279 280 281 282 283 284 285 286 287 288 289
RESTART
STATE[7:0]
FLAG
TEST_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
WRITE_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
Time (ns)
28800 28900 29000 29100 29200 29300 29400 29500 29600 29700 29800 29900 30000
CLOCK
289 290 291 292 293 294 295 296 297 298 299 300 301
RESTART
STATE[7:0]
FLAG
TEST_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
WRITE_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
Time (ns)
30000 30100 30200 30300 30400 30500 30600 30700 30800 30900 31000 31100 31200
CLOCK
301 302 303 304 305 306 307 308 309 310 311 312 313
RESTART
STATE[7:0]
FLAG
TEST_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
WRITE_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
Time (ns)
31200 31300 31400 31500 31600 31700 31800 31900 32000 32100 32200 32300 32400
CLOCK
313 314 315 316 317 318 319 320 321 322 323 324 325
RESTART
STATE[7:0]
FLAG
TEST_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
WRITE_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
Time (ns)
32400 32500 32600 32700 32800 32900 33000 33100 33200 33300 33400 33500 33600
CLOCK
325 326 327 328 329 330 331 332 333 334 335 336 337
RESTART
STATE[7:0]
FLAG
TEST_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000
WRITE_DATA[23:0]
000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000


Time (ns)
33600
33700
33800
33900
34000
34100
34200
34300
34400
34500
34600
34700
CLOCK
337
338
339
340
341
342
343
344
345
346
347
348
RESTART
STATE[7:0]
FLAG
TEST_DATA[23:0]
000000
000000
WRITE_DATA[23:0]
000000
000000

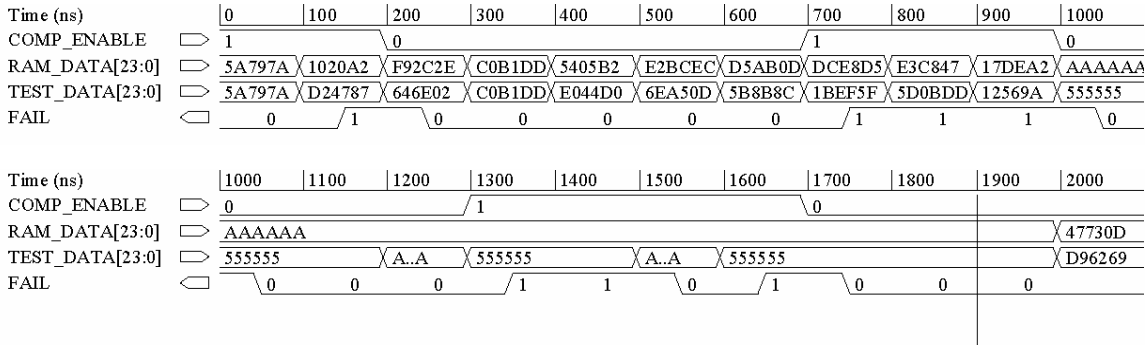
C. COMPARATOR MODULE

1. Schematic Diagram

The Xilinx LogiCORE Comparator is used to create one of the following comparison logic functions: A=B, A<>B, A<=B, A>=B, A<B, or A>B. A and B are external ports ranging in width from 1 to 256 bits, and B can optionally be set to a constant value. The module operates on signed or unsigned data.



2. Test-Bench Waveform



D. STATE MACHINE MODULE

1. VHDL Code

```
--------------------------------------------------------------------
-- filename: state_machine.vhd
-- written by: Charles Hulme
--
-- This state machine controls the flow of the different
-- tests that are to be executed and establishes a bit pattern
-- for each test that can be used in the pattern generator to
-- decode the correct pattern to use in each state.
--
-- Failing a test does not stop the state machine, but by asserting
-- the Flag signal the state will be captured by the status module
--
--------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity STATE_MACHINE is
    port (
        CLOCK: in STD_LOGIC;
        RESTART: in STD_LOGIC;
        FLAG: in STD_LOGIC;
        ADDR: in STD_LOGIC_VECTOR (23 downto 0);
        STATE: out STD_LOGIC_VECTOR (7 downto 0);
         PASS_ENABLE: out STD_LOGIC
        );
end STATE_MACHINE;

architecture STATE_MACHINE_arch of STATE_MACHINE is

type FSM_type is (WAIT_STATE,DELAY_COUNT1,DELAY_COUNT2,DELAY_COUNT3,
                     DELAY_COUNT4,
                     DATA_W1,DATA_R1,DATA_C1,DATA_W2,DATA_R2,DATA_C2,
                     DATA_W3,DATA_R3,DATA_C3,DATA_W4,DATA_R4,DATA_C4,
                     DATA_W5,DATA_R5,DATA_C5,DATA_W6,DATA_R6,DATA_C6,
                     DATA_W7,DATA_R7,DATA_C7,DATA_W8,DATA_R8,DATA_C8,
                    DATA_W9,DATA_R9,DATA_C9,DATA_W10,DATA_R10,DATA_C10,
            DATA_W11,DATA_R11,DATA_C11,DATA_W12,DATA_R12,DATA_C12,
            DATA_W13,DATA_R13,DATA_C13,DATA_W14,DATA_R14,DATA_C14,
            DATA_W15,DATA_R15,DATA_C15,DATA_W16,DATA_R16,DATA_C16,
            DATA_W17,DATA_R17,DATA_C17,DATA_W18,DATA_R18,DATA_C18,
            DATA_W19,DATA_R19,DATA_C19,DATA_W20,DATA_R20,DATA_C20,
            DATA_W21,DATA_R21,DATA_C21,DATA_W22,DATA_R22,DATA_C22,
            DATA_W23,DATA_R23,DATA_C23,DATA_W24,DATA_R24,DATA_C24,
            DATA_W25,DATA_R25,DATA_C25,WAIT_STATE1,DELAY_COUNT11,
            DELAY_COUNT12,DELAY_COUNT13,DELAY_COUNT14,
            ADDRESS_W1,ADDRESS_WL1,ADDRESS_WI1,ADDRESS_R1,ADDRESS_RL1,
            ADDRESS_W2,ADDRESS_WL2,ADDRESS_WI2,ADDRESS_R2,ADDRESS_RL2,
            ADDRESS_W3,ADDRESS_WL3,ADDRESS_WI3,ADDRESS_R3,ADDRESS_RL3,
            ADDRESS_W4,ADDRESS_WL4,ADDRESS_WI4,ADDRESS_R4,ADDRESS_RL4,
            ADDRESS_W5,ADDRESS_WL5,ADDRESS_WI5,ADDRESS_R5,ADDRESS_RL5,
```

```
            ADDRESS_W6,ADDRESS_WL6,ADDRESS_WI6,ADDRESS_R6,ADDRESS_RL6,
            ADDRESS_W7,ADDRESS_WL7,ADDRESS_WI7,ADDRESS_R7,ADDRESS_RL7,
            ADDRESS_W8,ADDRESS_WL8,ADDRESS_WI8,ADDRESS_R8,ADDRESS_RL8,
            ADDRESS_W9,ADDRESS_WL9,ADDRESS_WI9,ADDRESS_R9,ADDRESS_RL9,
      ADDRESS_W10,ADDRESS_WL10,ADDRESS_WI10,ADDRESS_R10,ADDRESS_RL10,
      ADDRESS_W11,ADDRESS_WL11,ADDRESS_WI11,ADDRESS_R11,ADDRESS_RL11,
      ADDRESS_W12,ADDRESS_WL12,ADDRESS_WI12,ADDRESS_R12,ADDRESS_RL12,
      ADDRESS_W13,ADDRESS_WL13,ADDRESS_WI13,ADDRESS_R13,ADDRESS_RL13,
      ADDRESS_W14,ADDRESS_WL14,ADDRESS_WI14,ADDRESS_R14,ADDRESS_RL14,
      ADDRESS_W15,ADDRESS_WL15,ADDRESS_WI15,ADDRESS_R15,ADDRESS_RL15,
      ADDRESS_W16,ADDRESS_WL16,ADDRESS_WI16,ADDRESS_R16,ADDRESS_RL16,
      ADDRESS_W17,ADDRESS_WL17,ADDRESS_WI17,ADDRESS_R17,ADDRESS_RL17,
      ADDRESS_W18,ADDRESS_WL18,ADDRESS_WI18,ADDRESS_R18,ADDRESS_RL18,
      ADDRESS_W19,ADDRESS_WL19,ADDRESS_WI19,ADDRESS_R19,ADDRESS_RL19,
      ADDRESS_W20,ADDRESS_WL20,ADDRESS_WI20,ADDRESS_R20,ADDRESS_RL20,
      ADDRESS_W21,ADDRESS_WL21,ADDRESS_WI21,ADDRESS_R21,ADDRESS_RL21,
      ADDRESS_W22,ADDRESS_WL22,ADDRESS_WI22,ADDRESS_R22,ADDRESS_RL22,
      ADDRESS_W23,ADDRESS_WL23,ADDRESS_WI23,ADDRESS_R23,ADDRESS_RL23,
      ADDRESS_W24,ADDRESS_WL24,ADDRESS_WI24,ADDRESS_R24,ADDRESS_RL24,
      MEMORY_WU,MEMORY_RU,MEMORY_WD,MEMORY_RD,MEMORY_WLU,MEMORY_RLU,
      MEMORY_WLD,MEMORY_RLD,FREEZE,PASS);

signal Curr_State, Next_State : FSM_Type;

begin

-- Process that implements the Next State Logic
nxtStProc: process(Curr_State,RESTART,FLAG,ADDR)

  begin

    if RESTART = '1' then
      Next_State <= WAIT_STATE;
    else
      case Curr_State is

        --Delay counter
        when WAIT_STATE =>
          Next_State <= DELAY_COUNT1;

        when DELAY_COUNT1 =>
          Next_State <= DELAY_COUNT2;

        when DELAY_COUNT2 =>
          Next_State <= DELAY_COUNT3;

        when DELAY_COUNT3 =>
          Next_State <= DELAY_COUNT4;

        when DELAY_COUNT4 =>
          Next_State <= DATA_W1;

        --Data Test
        when DATA_W1 =>  -- Write pattern to Memory
```

```
    Next_State <= DATA_R1;
when DATA_R1 =>  -- Read pattern from Memory
    Next_State <= DATA_W2;

when DATA_W2 =>
    Next_State <= DATA_R2;
when DATA_R2 =>
    Next_State <= DATA_W3;

when DATA_W3 =>
    Next_State <= DATA_R3;
when DATA_R3 =>
    Next_State <= DATA_W4;

when DATA_W4 =>
    Next_State <= DATA_R4;
when DATA_R4 =>
    Next_State <= DATA_W5;

when DATA_W5 =>
    Next_State <= DATA_R5;
when DATA_R5 =>
    Next_State <= DATA_W6;

when DATA_W6 =>
    Next_State <= DATA_R6;
when DATA_R6 =>
    Next_State <= DATA_W7;

when DATA_W7 =>
    Next_State <= DATA_R7;
when DATA_R7 =>
    Next_State <= DATA_W8;

when DATA_W8 =>
    Next_State <= DATA_R8;
when DATA_R8 =>
    Next_State <= DATA_W9;

when DATA_W9 =>
    Next_State <= DATA_R9;
when DATA_R9 =>
    Next_State <= DATA_W10;

when DATA_W10 =>
    Next_State <= DATA_R10;
when DATA_R10 =>
    Next_State <= DATA_W11;

when DATA_W11 =>
    Next_State <= DATA_R11;
when DATA_R11 =>
    Next_State <= DATA_W12;
```

```
when DATA_W12 =>
    Next_State <= DATA_R12;
when DATA_R12 =>
    Next_State <= DATA_W13;

when DATA_W13=>
    Next_State <= DATA_R13;
when DATA_R13=>
    Next_State <= DATA_W14;

when DATA_W14=>
    Next_State <= DATA_R14;
when DATA_R14=>
    Next_State <= DATA_W15;

when DATA_W15=>
    Next_State <= DATA_R15;
when DATA_R15=>
    Next_State <= DATA_W16;

when DATA_W16=>
    Next_State <= DATA_R16;
when DATA_R16=>
    Next_State <= DATA_W17;

when DATA_W17=>
    Next_State <= DATA_R17;
when DATA_R17=>
    Next_State <= DATA_W18;

when DATA_W18=>
    Next_State <= DATA_R18;
when DATA_R18=>
    Next_State <= DATA_W19;

when DATA_W19=>
    Next_State <= DATA_R19;
when DATA_R19=>
    Next_State <= DATA_W20;

when DATA_W20=>
    Next_State <= DATA_R20;
when DATA_R20=>
    Next_State <= DATA_W21;

when DATA_W21=>
    Next_State <= DATA_R21;
when DATA_R21=>
    Next_State <= DATA_W22;

when DATA_W22 =>
    Next_State <= DATA_R22;
when DATA_R22 =>
    Next_State <= DATA_W23;
```

```
when DATA_W23 =>
    Next_State <= DATA_R23;
when DATA_R23 =>
    Next_State <= DATA_W24;

when DATA_W24 =>
    Next_State <= DATA_R24;
when DATA_R24 =>
    Next_State <= DATA_W25;

when DATA_W25 =>
    Next_State <= DATA_R25;
when DATA_R25 =>
    Next_State <= WAIT_STATE1;

-- Delay counter
when WAIT_STATE1 =>
  Next_State <= DELAY_COUNT11;

when DELAY_COUNT11 =>
  Next_State <= DELAY_COUNT12;

when DELAY_COUNT12 =>
  Next_State <= DELAY_COUNT13;

when DELAY_COUNT13 =>
  Next_State <= DELAY_COUNT14;

when DELAY_COUNT14 =>
  Next_State <= ADDRESS_W1;

--Address Test
when ADDRESS_W1 => --write data to all Power of 2 addresses
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_WL1;
    else
    Next_State <= ADDRESS_W1;
  end if;
 when ADDRESS_WL1 => --write data to last Power of 2 address
      Next_State <= ADDRESS_WI1;
when ADDRESS_WI1 => --write an inverted copy to one address
    Next_State <= ADDRESS_R1;
when ADDRESS_R1 => --read data from all Power of 2 addresses
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_RL1;
    else
    Next_State <= ADDRESS_R1;
  end if;
 when ADDRESS_RL1 => --read data from last Power of 2 address
    Next_State <= ADDRESS_W2;

when ADDRESS_W2 =>
  if ADDR = "10000000000000000000000" then
```

```
    Next_State <= ADDRESS_WL2;
    else
    Next_State <= ADDRESS_W2;
  end if;
 when ADDRESS_WL2 =>
      Next_State <= ADDRESS_WI2;
 when ADDRESS_WI2 =>
    Next_State <= ADDRESS_R2;
when ADDRESS_R2 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_RL2;
    else
    Next_State <= ADDRESS_R2;
  end if;
 when ADDRESS_RL2 =>
    Next_State <= ADDRESS_W3;

when ADDRESS_W3 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL3;
    else
    Next_State <= ADDRESS_W3;
  end if;
when ADDRESS_WL3 =>
      Next_State <= ADDRESS_WI3;
when ADDRESS_WI3 =>
    Next_State <= ADDRESS_R3;
when ADDRESS_R3 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_RL3;
    else
    Next_State <= ADDRESS_R3;
  end if;
 when ADDRESS_RL3 =>
    Next_State <= ADDRESS_W4;

when ADDRESS_W4 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL4;
    else
    Next_State <= ADDRESS_W4;
  end if;
when ADDRESS_WL4 =>
      Next_State <= ADDRESS_WI4;
when ADDRESS_WI4 =>
    Next_State <= ADDRESS_R4;
when ADDRESS_R4 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_RL4;
    else
    Next_State <= ADDRESS_R4;
  end if;
 when ADDRESS_RL4 =>
    Next_State <= ADDRESS_W5;
```

```
when  ADDRESS_W5 =>
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_WL5;
    else
    Next_State <=  ADDRESS_W5;
  end if;
when ADDRESS_WL5 =>
      Next_State <= ADDRESS_WI5;
when ADDRESS_WI5 =>
    Next_State <=  ADDRESS_R5;
when  ADDRESS_R5 =>
  if ADDR = "10000000000000000000000" then
    Next_State <=  ADDRESS_RL5;
    else
    Next_State <=  ADDRESS_R5;
  end if;
 when  ADDRESS_RL5 =>
    Next_State <=  ADDRESS_W6;

when  ADDRESS_W6 =>
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_WL6;
    else
    Next_State <=  ADDRESS_W6;
  end if;
when ADDRESS_WL6 =>
      Next_State <= ADDRESS_WI6;
when ADDRESS_WI6 =>
    Next_State <=  ADDRESS_R6;
when  ADDRESS_R6 =>
  if ADDR = "10000000000000000000000" then
    Next_State <=  ADDRESS_RL6;
    else
    Next_State <=  ADDRESS_R6;
  end if;
 when  ADDRESS_RL6 =>
    Next_State <=  ADDRESS_W7;

when  ADDRESS_W7 =>
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_WL7;
    else
    Next_State <=  ADDRESS_W7;
  end if;
when ADDRESS_WL7 =>
      Next_State <= ADDRESS_WI7;
when ADDRESS_WI7 =>
    Next_State <=  ADDRESS_R7;
when  ADDRESS_R7 =>
  if ADDR = "10000000000000000000000" then
    Next_State <=  ADDRESS_RL7;
    else
    Next_State <=  ADDRESS_R7;
```

```
  end if;
 when  ADDRESS_RL7 =>
    Next_State <=  ADDRESS_W8;

when  ADDRESS_W8 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL8;
    else
    Next_State <=  ADDRESS_W8;
  end if;
when ADDRESS_WL8 =>
      Next_State <= ADDRESS_WI8;
when ADDRESS_WI8 =>
    Next_State <=  ADDRESS_R8;
when  ADDRESS_R8 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <=  ADDRESS_RL8;
    else
    Next_State <=  ADDRESS_R8;
  end if;
 when  ADDRESS_RL8 =>
    Next_State <=  ADDRESS_W9;

when  ADDRESS_W9 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL9;
    else
    Next_State <=  ADDRESS_W9;
  end if;
when ADDRESS_WL9 =>
      Next_State <= ADDRESS_WI9;
when ADDRESS_WI9 =>
    Next_State <=  ADDRESS_R9;
when  ADDRESS_R9 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <=  ADDRESS_RL9;
    else
    Next_State <=  ADDRESS_R9;
  end if;
 when  ADDRESS_RL9 =>
    Next_State <=  ADDRESS_W10;

when  ADDRESS_W10 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL10;
    else
    Next_State <=  ADDRESS_W10;
  end if;
when ADDRESS_WL10 =>
      Next_State <= ADDRESS_WI10;
when ADDRESS_WI10 =>
    Next_State <=  ADDRESS_R10;
when  ADDRESS_R10 =>
  if ADDR = "10000000000000000000000000" then
```

```
    Next_State <=  ADDRESS_RL10;
    else
    Next_State <=  ADDRESS_R10;
  end if;
 when  ADDRESS_RL10 =>
    Next_State <=  ADDRESS_W11;

when  ADDRESS_W11 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL11;
    else
    Next_State <=  ADDRESS_W11;
  end if;
when ADDRESS_WL11 =>
      Next_State <= ADDRESS_WI11;
when ADDRESS_WI11 =>
    Next_State <=  ADDRESS_R11;
when  ADDRESS_R11 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <=  ADDRESS_RL11;
    else
    Next_State <=  ADDRESS_R11;
  end if;
 when  ADDRESS_RL11 =>
    Next_State <=  ADDRESS_W12;

when  ADDRESS_W12 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL12;
    else
    Next_State <=  ADDRESS_W12;
  end if;
when ADDRESS_WL12 =>
      Next_State <= ADDRESS_WI12;
when ADDRESS_WI12 =>
    Next_State <=  ADDRESS_R12;
when  ADDRESS_R12=>
  if ADDR = "10000000000000000000000000" then
    Next_State <=  ADDRESS_RL12;
    else
    Next_State <=  ADDRESS_R12;
  end if;
 when  ADDRESS_RL12 =>
    Next_State <=  ADDRESS_W13;

when  ADDRESS_W13 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL13;
    else
    Next_State <=  ADDRESS_W13;
  end if;
when ADDRESS_WL13 =>
      Next_State <= ADDRESS_WI13;
when ADDRESS_WI13 =>
```

```
    Next_State <=  ADDRESS_R13;
when  ADDRESS_R13=>
  if ADDR = "10000000000000000000000" then
    Next_State <=  ADDRESS_RL13;
    else
    Next_State <=  ADDRESS_R13;
  end if;
 when  ADDRESS_RL13 =>
    Next_State <=  ADDRESS_W14;

when  ADDRESS_W14 =>
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_WL14;
    else
    Next_State <=  ADDRESS_W14;
  end if;
when ADDRESS_WL14 =>
      Next_State <= ADDRESS_WI14;
when ADDRESS_WI14 =>
    Next_State <=  ADDRESS_R14;
when  ADDRESS_R14=>
  if ADDR = "10000000000000000000000" then
    Next_State <=  ADDRESS_RL14;
    else
    Next_State <=  ADDRESS_R14;
  end if;
 when  ADDRESS_RL14 =>
    Next_State <=  ADDRESS_W15;

when  ADDRESS_W15 =>
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_WL15;
    else
    Next_State <=  ADDRESS_W15;
  end if;
when ADDRESS_WL15 =>
      Next_State <= ADDRESS_WI15;
when ADDRESS_WI15 =>
    Next_State <=  ADDRESS_R15;
when  ADDRESS_R15=>
  if ADDR = "10000000000000000000000" then
    Next_State <=  ADDRESS_RL15;
    else
    Next_State <=  ADDRESS_R15;
  end if;
 when  ADDRESS_RL15 =>
    Next_State <=  ADDRESS_W16;

when  ADDRESS_W16 =>
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_WL16;
    else
    Next_State <=  ADDRESS_W16;
  end if;
```

```
when ADDRESS_WL16 =>
      Next_State <= ADDRESS_WI16;
when ADDRESS_WI16 =>
    Next_State <=  ADDRESS_R16;
when  ADDRESS_R16=>
  if ADDR = "10000000000000000000000000" then
    Next_State <=  ADDRESS_RL16;
    else
    Next_State <=  ADDRESS_R16;
  end if;
 when  ADDRESS_RL16 =>
    Next_State <=  ADDRESS_W17;

when  ADDRESS_W17 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL17;
    else
    Next_State <=  ADDRESS_W17;
  end if;
when ADDRESS_WL17 =>
      Next_State <= ADDRESS_WI17;
when ADDRESS_WI17 =>
    Next_State <=  ADDRESS_R17;
when  ADDRESS_R17=>
  if ADDR = "10000000000000000000000000" then
    Next_State <=  ADDRESS_RL17;
    else
    Next_State <=  ADDRESS_R17;
  end if;
 when  ADDRESS_RL17 =>
    Next_State <=  ADDRESS_W18;

when  ADDRESS_W18 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL18;
    else
    Next_State <=  ADDRESS_W18;
  end if;
when ADDRESS_WL18 =>
      Next_State <= ADDRESS_WI18;
when ADDRESS_WI18 =>
    Next_State <=  ADDRESS_R18;
when  ADDRESS_R18=>
  if ADDR = "10000000000000000000000000" then
    Next_State <=  ADDRESS_RL18;
    else
    Next_State <=  ADDRESS_R18;
  end if;
 when  ADDRESS_RL18 =>
    Next_State <=  ADDRESS_W19;

when  ADDRESS_W19 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL19;
```

```
    else
    Next_State <=  ADDRESS_W19;
  end if;
when ADDRESS_WL19 =>
      Next_State <= ADDRESS_WI19;
when ADDRESS_WI19 =>
    Next_State <=  ADDRESS_R19;
when  ADDRESS_R19=>
  if ADDR = "10000000000000000000000000" then
    Next_State <=  ADDRESS_RL19;
    else
    Next_State <=  ADDRESS_R19;
  end if;
 when  ADDRESS_RL19 =>
    Next_State <=  ADDRESS_W20;

when  ADDRESS_W20 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL20;
    else
    Next_State <=  ADDRESS_W20;
  end if;
when ADDRESS_WL20 =>
      Next_State <= ADDRESS_WI20;
when ADDRESS_WI20 =>
    Next_State <=  ADDRESS_R20;
when  ADDRESS_R20 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <=  ADDRESS_RL20;
    else
    Next_State <=  ADDRESS_R20;
  end if;
 when  ADDRESS_RL20 =>
    Next_State <=  ADDRESS_W21;

when  ADDRESS_W21 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <= ADDRESS_WL21;
    else
    Next_State <=  ADDRESS_W21;
  end if;
when ADDRESS_WL21 =>
      Next_State <= ADDRESS_WI21;
when ADDRESS_WI21 =>
    Next_State <=  ADDRESS_R21;
when  ADDRESS_R21 =>
  if ADDR = "10000000000000000000000000" then
    Next_State <=  ADDRESS_RL21;
    else
    Next_State <=  ADDRESS_R21;
  end if;
 when  ADDRESS_RL21 =>
    Next_State <=  ADDRESS_W22;
```

```
when  ADDRESS_W22 =>
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_WL22;
    else
    Next_State <=  ADDRESS_W22;
  end if;
when ADDRESS_WL22 =>
      Next_State <= ADDRESS_WI22;
when ADDRESS_WI22 =>
    Next_State <=  ADDRESS_R22;
when  ADDRESS_R22 =>
  if ADDR = "10000000000000000000000" then
    Next_State <=  ADDRESS_RL22;
    else
    Next_State <=  ADDRESS_R22;
  end if;
 when  ADDRESS_RL22 =>
    Next_State <=  ADDRESS_W23;

when  ADDRESS_W23 =>
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_WL23;
    else
    Next_State <=  ADDRESS_W23;
  end if;
when ADDRESS_WL23 =>
      Next_State <= ADDRESS_WI23;
when ADDRESS_WI23 =>
    Next_State <=  ADDRESS_R23;
when  ADDRESS_R23 =>
  if ADDR = "10000000000000000000000" then
    Next_State <=  ADDRESS_RL23;
    else
    Next_State <=  ADDRESS_R23;
  end if;
 when  ADDRESS_RL23 =>
    Next_State <=  ADDRESS_W24;

when  ADDRESS_W24 =>
  if ADDR = "10000000000000000000000" then
    Next_State <= ADDRESS_WL24;
    else
    Next_State <=  ADDRESS_W24;
  end if;
when ADDRESS_WL24 =>
      Next_State <= ADDRESS_WI24;
when ADDRESS_WI24 =>
    Next_State <=  ADDRESS_R24;
when  ADDRESS_R24 =>
  if ADDR = "10000000000000000000000" then
    Next_State <=  ADDRESS_RL24;
    else
    Next_State <=  ADDRESS_R24;
  end if;
```

```
 when  ADDRESS_RL24 =>
    Next_State <=  MEMORY_WU;

--Memory Test
when MEMORY_WU =>
  if ADDR = "111111111111111111111111" then
    Next_State <= MEMORY_WLU;
    else
      Next_State <= MEMORY_WU;
  end if;

    when MEMORY_WLU =>
      Next_State <= MEMORY_RU;

when MEMORY_RU =>
  if ADDR = "111111111111111111111111" then
    Next_State <= MEMORY_RLU;
    else
      Next_State <= MEMORY_RU;
  end if;

    when MEMORY_RLU =>
      Next_State <= MEMORY_WD;

when MEMORY_WD =>
  if ADDR = "000000000000000000000000" then
    Next_State <= MEMORY_WLD;
  else
    Next_State <= MEMORY_WD;
  end if;

    when MEMORY_WLD =>
      Next_State <= MEMORY_RD;

when MEMORY_RD =>
  if ADDR = "000000000000000000000000" then
    Next_State <= MEMORY_RLD;
  else
    Next_State <= MEMORY_RD;
  end if;

    when MEMORY_RLD =>
      Next_State <= PASS;

--If all tests pass
when PASS =>
  Next_State <= DELAY_COUNT1;

 --If a test fails
when FREEZE =>
  if RESTART = '1' then
      Next_State <= WAIT_STATE;
    else
      NEXT_STATE <= FREEZE;
```

```
            end if;

        when others => Next_State <= WAIT_STATE;

        end case;
    end if;

end process nxtStProc;

  --Process to register current state
  curStProc: process (CLOCK,RESTART,Next_State)
  begin
    if (RESTART = '1') then
       Curr_State <= WAIT_STATE;
    elsif (CLOCK'event and CLOCK ='1') then
       Curr_State <= Next_State;
    end if;
  end process curStProc;

  --Process to generate outputs
  outConProc: process(Curr_State)
  begin

    -- Set default value of ZERO for all output signals
    STATE <= "00000000";
    PASS_ENABLE <= '0';

    -- Set certain outputs depending on which state currently in
    case Curr_State is

      when WAIT_STATE =>
         STATE <= "00000000";

      when DELAY_COUNT1 =>
         STATE <= "00000001";

      when DELAY_COUNT2 =>
         STATE <= "00000001";

      when DELAY_COUNT3 =>
         STATE <= "00000001";

      when DELAY_COUNT4 =>
         STATE <= "00000001";

    --Starting Data Test
      when DATA_W1 =>   -- write data
         STATE <= "00000010";
      when DATA_R1 =>   -- read data
         STATE <= "00000011";

      when DATA_W2 =>
         STATE <= "00000100";
      when DATA_R2 =>
```

```
   STATE <= "00000101";

when DATA_W3 =>
   STATE <= "00000110";
when DATA_R3 =>
   STATE <= "00000111";

when DATA_W4 =>
   STATE <= "00001000";
when DATA_R4 =>
   STATE <= "00001001";

when DATA_W5 =>
   STATE <= "00001010";
when DATA_R5 =>
   STATE <= "00001011";

when DATA_W6 =>
   STATE <= "00001100";
when DATA_R6 =>
   STATE <= "00001101";

when DATA_W7 =>
   STATE <= "00001110";
when DATA_R7 =>
   STATE <= "00001111";

when DATA_W8 =>
   STATE <= "00010000";
when DATA_R8 =>
   STATE <= "00010001";

when DATA_W9 =>
   STATE <= "00010010";
when DATA_R9 =>
   STATE <= "00010011";

when DATA_W10 =>
   STATE <= "00010100";
when DATA_R10 =>
   STATE <= "00010101";

when DATA_W11 =>
   STATE <= "00010110";
when DATA_R11 =>
   STATE <= "00010111";

when DATA_W12 =>
   STATE <= "00011000";
when DATA_R12 =>
   STATE <= "00011001";

when DATA_W13 =>
   STATE <= "00011010";
```

```
when DATA_R13 =>
   STATE <= "00011011";

when DATA_W14 =>
   STATE <= "00011100";
when DATA_R14 =>
   STATE <= "00011101";

when DATA_W15 =>
   STATE <= "00011110";
when DATA_R15 =>
   STATE <= "00011111";

when DATA_W16 =>
   STATE <= "00100000";
when DATA_R16 =>
   STATE <= "00100001";

when DATA_W17 =>
   STATE <= "00100010";
when DATA_R17 =>
   STATE <= "00100011";

when DATA_W18 =>
   STATE <= "00100100";
when DATA_R18 =>
   STATE <= "00100101";

when DATA_W19 =>
   STATE <= "00100110";
when DATA_R19 =>
   STATE <= "00100111";

when DATA_W20 =>
   STATE <= "00101000";
when DATA_R20 =>
   STATE <= "00101001";

when DATA_W21 =>
   STATE <= "00101010";
when DATA_R21 =>
   STATE <= "00101011";

when DATA_W22 =>
   STATE <= "00101100";
when DATA_R22 =>
   STATE <= "00101101";

when DATA_W23 =>
   STATE <= "00101110";
when DATA_R23 =>
   STATE <= "00101111";

when DATA_W24 =>
```

```
      STATE <= "00110000";
    when DATA_R24 =>
      STATE <= "00110001";

    when DATA_W25 =>
      STATE <= "00110010";
    when DATA_R25 =>
      STATE <= "00110011";

    when WAIT_STATE1 =>  -- start delay counter
      STATE <= "00111110";

    when DELAY_COUNT11 =>
      STATE <= "00111111";

    when DELAY_COUNT12 =>
      STATE <= "00111111";

    when DELAY_COUNT13 =>
      STATE <= "00111111";

    when DELAY_COUNT14 =>
      STATE <= "00111111";

  --Starting Address Test
    when  ADDRESS_W1 =>  -- write pattern
      STATE <= "01000000";
    when ADDRESS_WL1 => -- write pattern to last address
      STATE <= "01000001";
     when ADDRESS_WI1 =>  -- write inverted pattern
      STATE <= "01000010";
    when  ADDRESS_R1 =>  -- read pattern
      STATE <= "01000011";
    when  ADDRESS_RL1 =>  -- read pattern from last address
      STATE <= "11000000";

    when ADDRESS_W2 =>
      STATE <= "01000100";
    when ADDRESS_WL2 =>
      STATE <= "01000101";
    when ADDRESS_WI2 =>
      STATE <= "01000110";
    when ADDRESS_R2 =>
      STATE <= "01000111";
    when ADDRESS_RL2 =>
      STATE <= "11000001";

    when ADDRESS_W3 =>
      STATE <= "01001000";
    when ADDRESS_WL3 =>
      STATE <= "01001001";
    when ADDRESS_WI3 =>
      STATE <= "01001010";
    when ADDRESS_R3 =>
```

```
   STATE <= "01001011";
when ADDRESS_RL3 =>
   STATE <= "11000010";

when ADDRESS_W4 =>
   STATE <= "01001100";
when ADDRESS_WL4 =>
   STATE <= "01001101";
when ADDRESS_WI4 =>
   STATE <= "01001110";
when ADDRESS_R4 =>
   STATE <= "01001111";
when ADDRESS_RL4 =>
   STATE <= "11000011";

when ADDRESS_W5 =>
   STATE <= "01010000";
when ADDRESS_WL5 =>
   STATE <= "01010001";
when ADDRESS_WI5 =>
   STATE <= "01010010";
when ADDRESS_R5 =>
   STATE <= "01010011";
when ADDRESS_RL5 =>
   STATE <= "11000100";

when ADDRESS_W6 =>
   STATE <= "01010100";
when ADDRESS_WL6 =>
   STATE <= "01010101";
when ADDRESS_WI6 =>
   STATE <= "01010110";
when ADDRESS_R6 =>
   STATE <= "01010111";
when ADDRESS_RL6 =>
   STATE <= "11000101";

when ADDRESS_W7 =>
   STATE <= "01011000";
when ADDRESS_WL7 =>
   STATE <= "01011001";
when ADDRESS_WI7 =>
   STATE <= "01011010";
when ADDRESS_R7 =>
   STATE <= "01011011";
when ADDRESS_RL7 =>
   STATE <= "11000110";

when ADDRESS_W8 =>
   STATE <= "01011100";
when ADDRESS_WL8 =>
   STATE <= "01011101";
when ADDRESS_WI8 =>
   STATE <= "01011110";
```

```
when ADDRESS_R8 =>
   STATE <= "01011111";
when ADDRESS_RL8 =>
   STATE <= "11000111";

 when ADDRESS_W9  =>
   STATE <= "01100000";
when ADDRESS_WL9  =>
   STATE <= "01100001";
when ADDRESS_WI9  =>
   STATE <= "01100010";
when ADDRESS_R9  =>
   STATE <= "01100011";
when ADDRESS_RL9 =>
   STATE <= "11001000";

when ADDRESS_W10 =>
   STATE <= "01100100";
when ADDRESS_WL10 =>
   STATE <= "01100101";
when ADDRESS_WI10 =>
   STATE <= "01100110";
when ADDRESS_R10 =>
   STATE <= "01100111";
when ADDRESS_RL10 =>
   STATE <= "11001001";

when ADDRESS_W11 =>
   STATE <= "01101000";
when ADDRESS_WL11 =>
   STATE <= "01101001";
when ADDRESS_WI11 =>
   STATE <= "01101010";
when ADDRESS_R11 =>
   STATE <= "01101011";
when ADDRESS_RL11 =>
   STATE <= "11001010";

when ADDRESS_W12 =>
   STATE <= "01101100";
when ADDRESS_WL12 =>
   STATE <= "01101101";
when ADDRESS_WI12 =>
   STATE <= "01101110";
when ADDRESS_R12 =>
   STATE <= "01101111";
when ADDRESS_RL12 =>
   STATE <= "11001011";

when  ADDRESS_W13 =>
   STATE <= "01110000";
when ADDRESS_WL13 =>
   STATE <= "01110001";
when  ADDRESS_WI13 =>
```

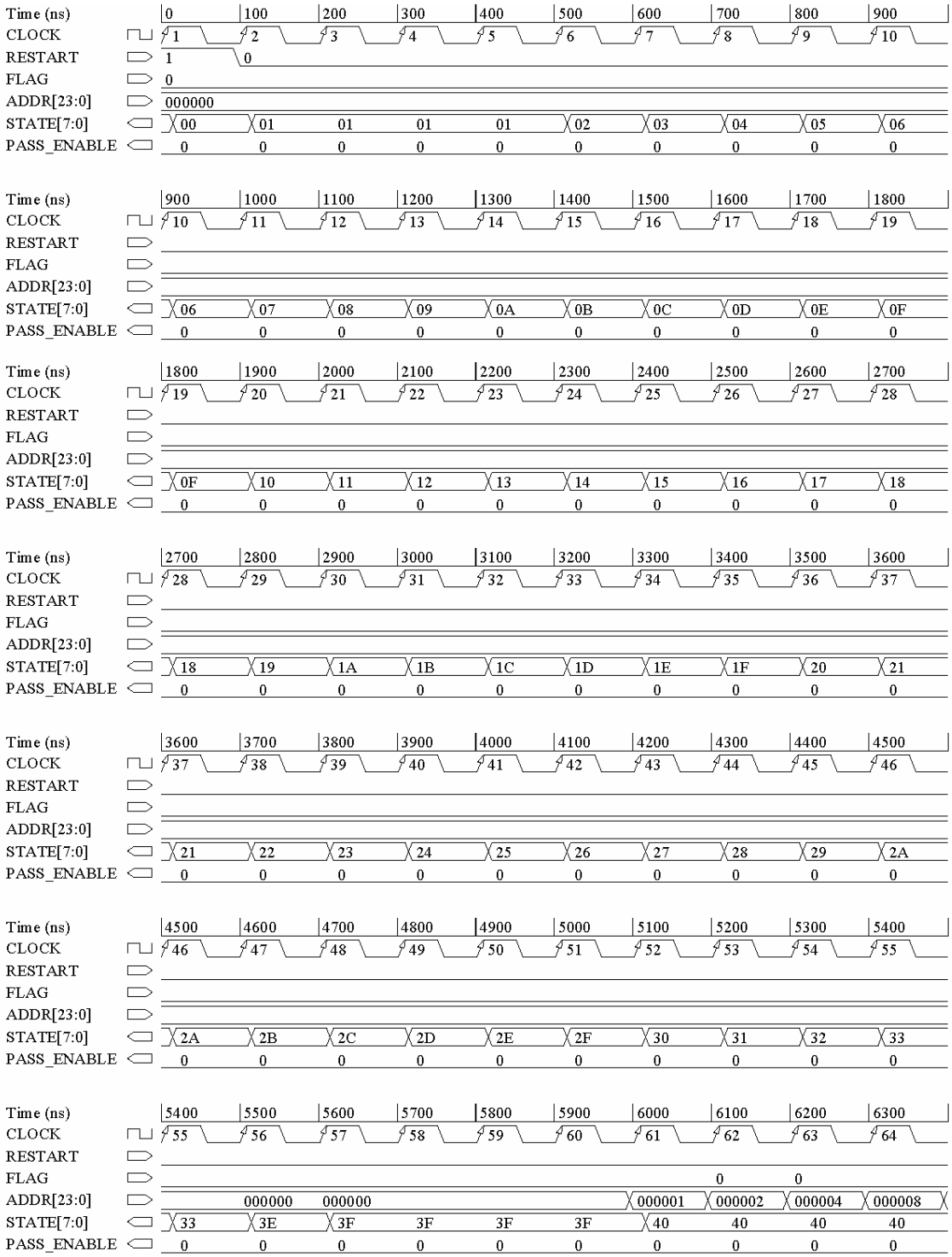
```
   STATE <= "01110010";
when  ADDRESS_R13 =>
   STATE <= "01110011";
when ADDRESS_RL13 =>
   STATE <= "11001100";

when  ADDRESS_W14 =>
   STATE <= "01110100";
when ADDRESS_WL14 =>
   STATE <= "01110101";
when  ADDRESS_WI14 =>
   STATE <= "01110110";
when  ADDRESS_R14 =>
   STATE <= "01110111";
when ADDRESS_RL14 =>
   STATE <= "11001101";

when  ADDRESS_W15 =>
   STATE <= "01111000";
when ADDRESS_WL15 =>
   STATE <= "01111001";
when  ADDRESS_WI15 =>
   STATE <= "01111010";
when  ADDRESS_R15 =>
   STATE <= "01111011";
when ADDRESS_RL15 =>
   STATE <= "11001110";

when  ADDRESS_W16 =>
   STATE <= "01111100";
when ADDRESS_WL16 =>
   STATE <= "01111101";
when  ADDRESS_WI16 =>
   STATE <= "01111110";
when  ADDRESS_R16 =>
   STATE <= "01111111";
when ADDRESS_RL16 =>
   STATE <= "11001111";

when  ADDRESS_W17 =>
   STATE <= "10000000";
when ADDRESS_WL17 =>
   STATE <= "10000001";
when  ADDRESS_WI17 =>
   STATE <= "10000010";
when  ADDRESS_R17 =>
   STATE <= "10000011";
when ADDRESS_RL17 =>
   STATE <= "11010000";

when  ADDRESS_W18 =>
   STATE <= "10000100";
when ADDRESS_WL18 =>
   STATE <= "10000101";
```

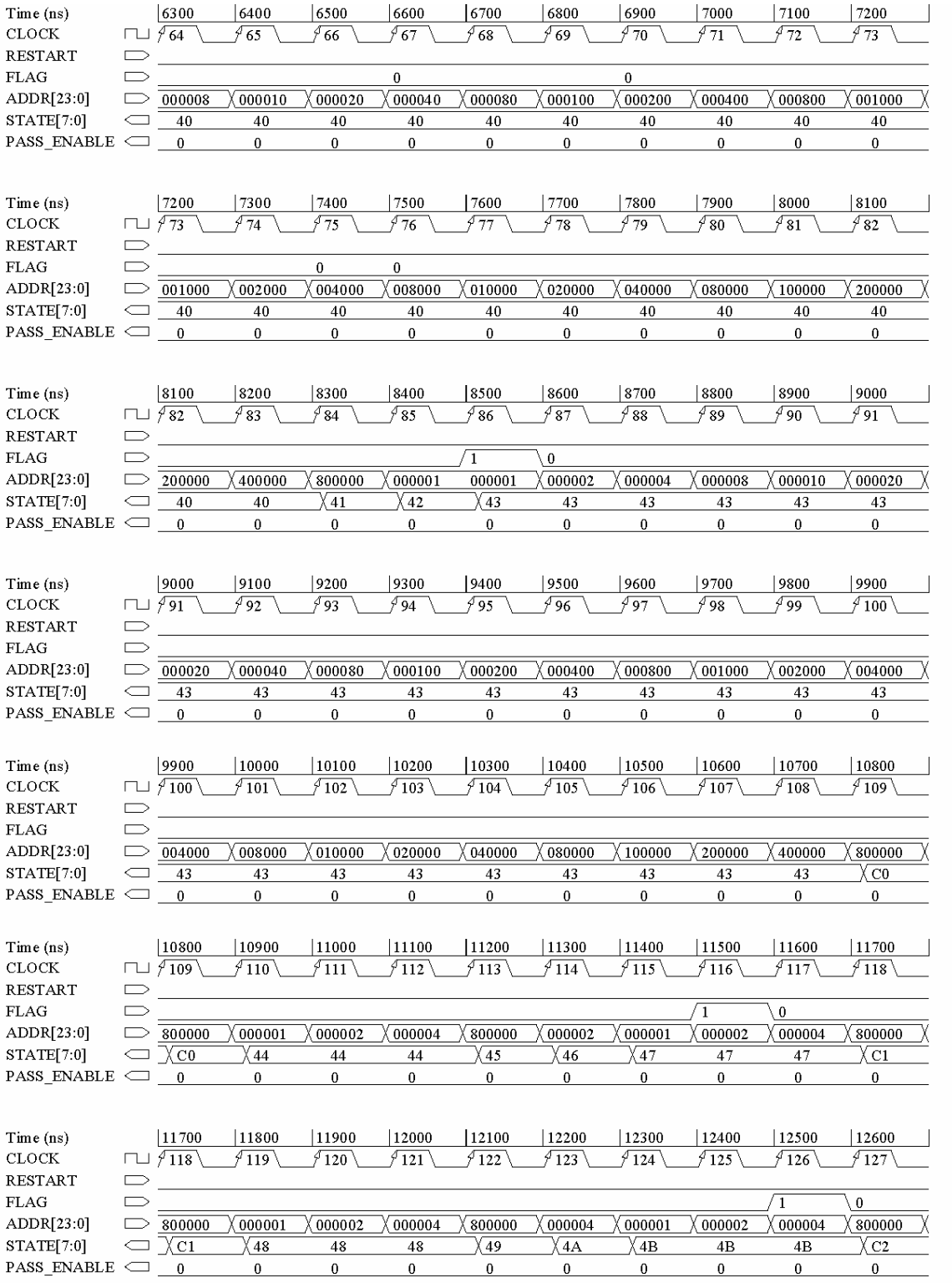
```
when  ADDRESS_WI18 =>
   STATE <= "10000110";
when  ADDRESS_R18 =>
   STATE <= "10000111";
when ADDRESS_RL18 =>
   STATE <= "11010001";

when  ADDRESS_W19 =>
   STATE <= "10001000";
when ADDRESS_WL19 =>
   STATE <= "10001001";
when  ADDRESS_WI19 =>
   STATE <= "10001010";
when  ADDRESS_R19 =>
   STATE <= "10001011";
when ADDRESS_RL19 =>
   STATE <= "11010010";

when  ADDRESS_W20 =>
   STATE <= "10001100";
when ADDRESS_WL20 =>
   STATE <= "10001101";
when  ADDRESS_WI20 =>
   STATE <= "10001110";
when  ADDRESS_R20 =>
   STATE <= "10001111";
when ADDRESS_RL20 =>
   STATE <= "11010011";

when  ADDRESS_W21 =>
   STATE <= "10010000";
when ADDRESS_WL21 =>
   STATE <= "10010001";
when  ADDRESS_WI21 =>
   STATE <= "10010010";
when  ADDRESS_R21 =>
   STATE <= "10010011";
when ADDRESS_RL21 =>
   STATE <= "11010100";

when  ADDRESS_W22 =>
   STATE <= "10010100";
when ADDRESS_WL22 =>
   STATE <= "10010101";
when  ADDRESS_WI22 =>
   STATE <= "10010110";
when  ADDRESS_R22 =>
   STATE <= "10010111";
when ADDRESS_RL22 =>
   STATE <= "11010101";

when  ADDRESS_W23 =>
   STATE <= "10011000";
when ADDRESS_WL23 =>
```

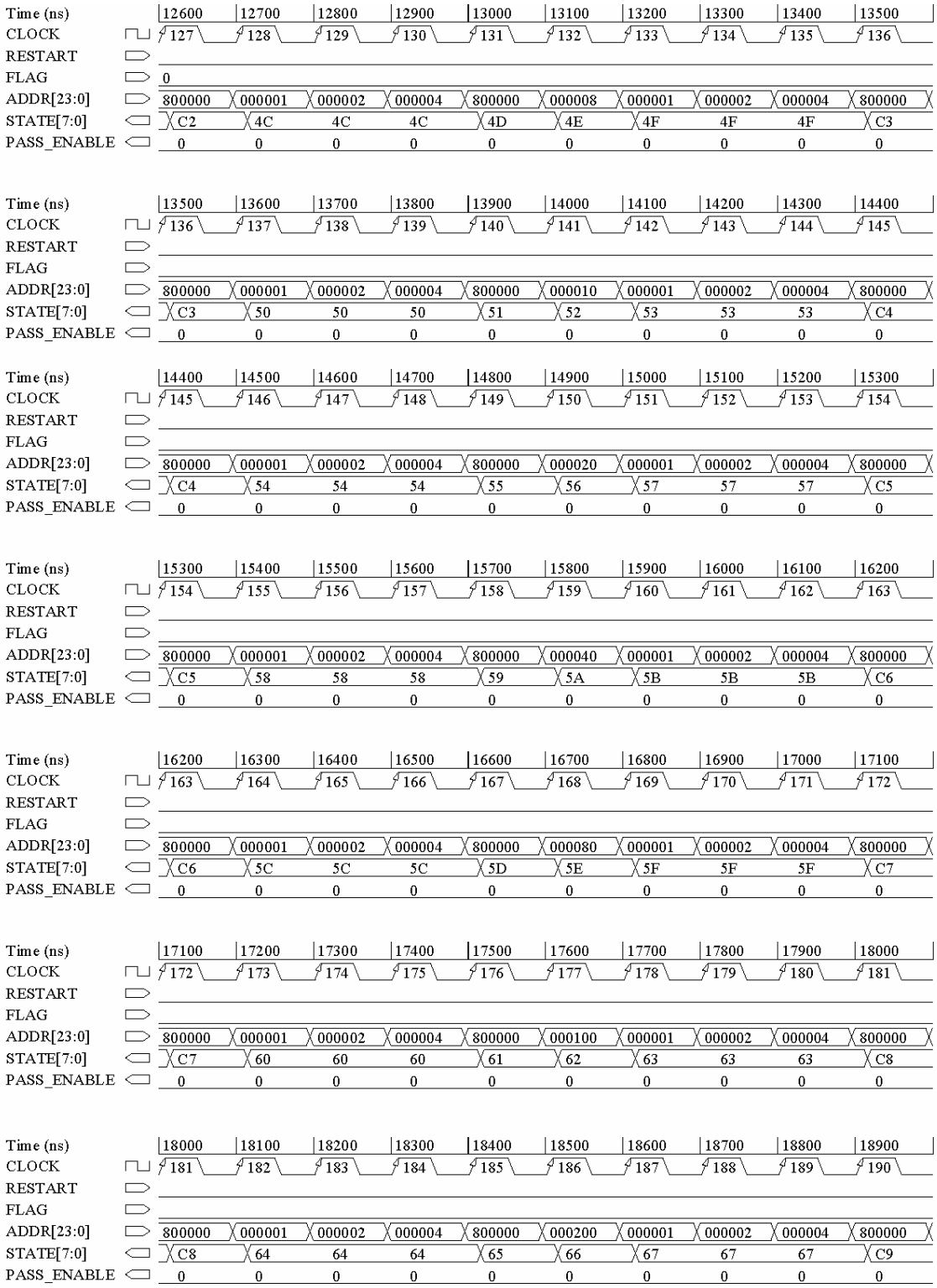
```
      STATE <= "10011001";
    when  ADDRESS_WI23 =>
      STATE <= "10011010";
    when  ADDRESS_R23 =>
      STATE <= "10011011";
    when ADDRESS_RL23 =>
      STATE <= "11010110";

    when  ADDRESS_W24 =>
      STATE <= "10011100";
    when ADDRESS_WL24 =>
      STATE <= "10011101";
    when  ADDRESS_WI24 =>
      STATE <= "10011110";
    when  ADDRESS_R24 =>
      STATE <= "10011111";
    when ADDRESS_RL24 =>
      STATE <= "11010111";

  --Starting Memory Test
    when MEMORY_WU =>
      STATE <= "11111000";

    when MEMORY_WLU =>
      STATE <= "11111000";

    when MEMORY_RU =>
      STATE <= "11111001";

    when MEMORY_RLU =>
      STATE <= "11111001";

    when MEMORY_WD =>
      STATE <= "11111010";

    when MEMORY_WLD =>
      STATE <= "11111010";

    when MEMORY_RD =>
      STATE <= "11111011";

    when MEMORY_RLD =>
      STATE <= "11111011";

    when PASS =>
      PASS_ENABLE <= '1';   -- clock pass counter
      STATE <= "11111100";

    when FREEZE =>
      STATE <= "11111111";

    when others =>
      null;
```

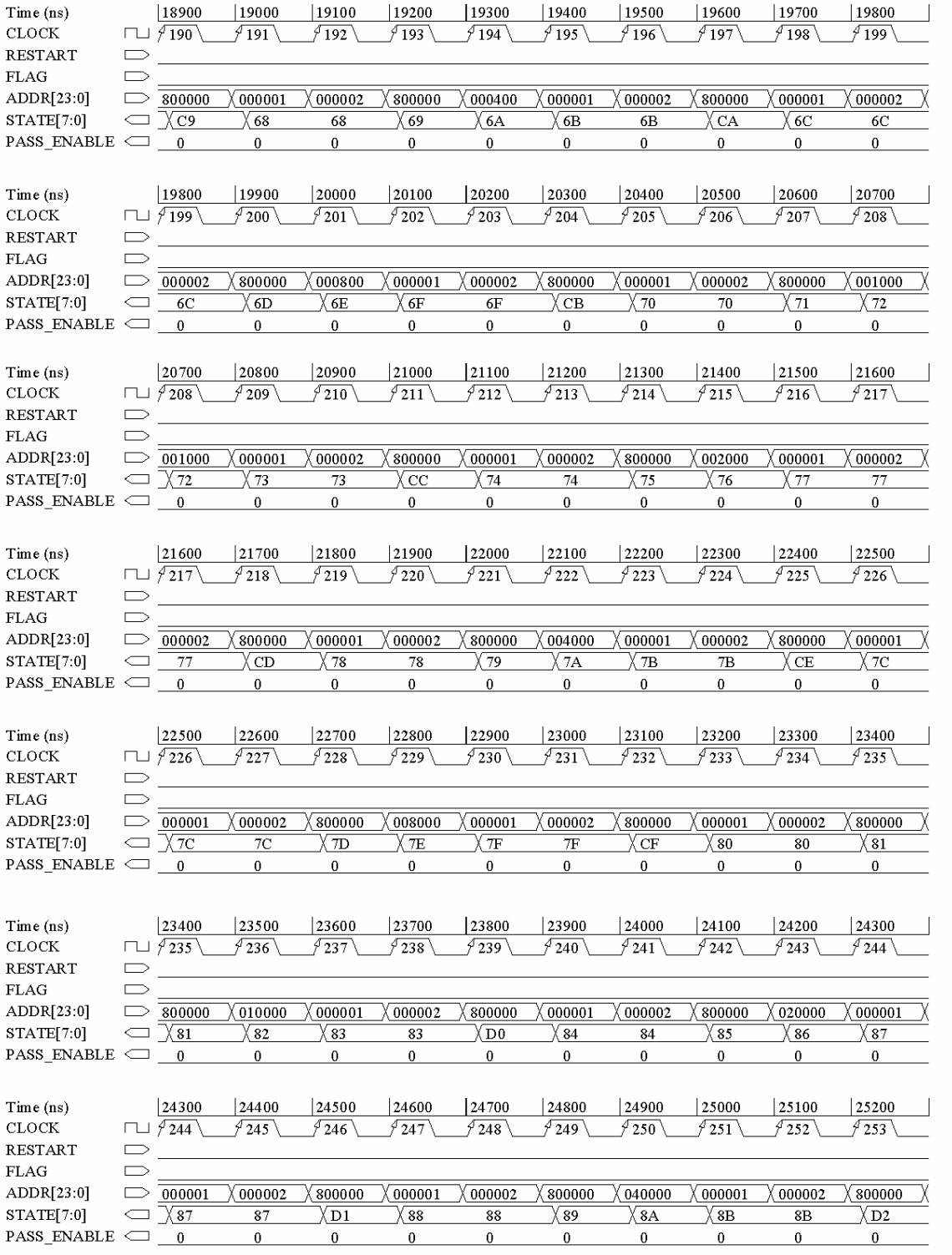
```
    end case;

   end process outConProc;

end STATE_MACHINE_arch;
```

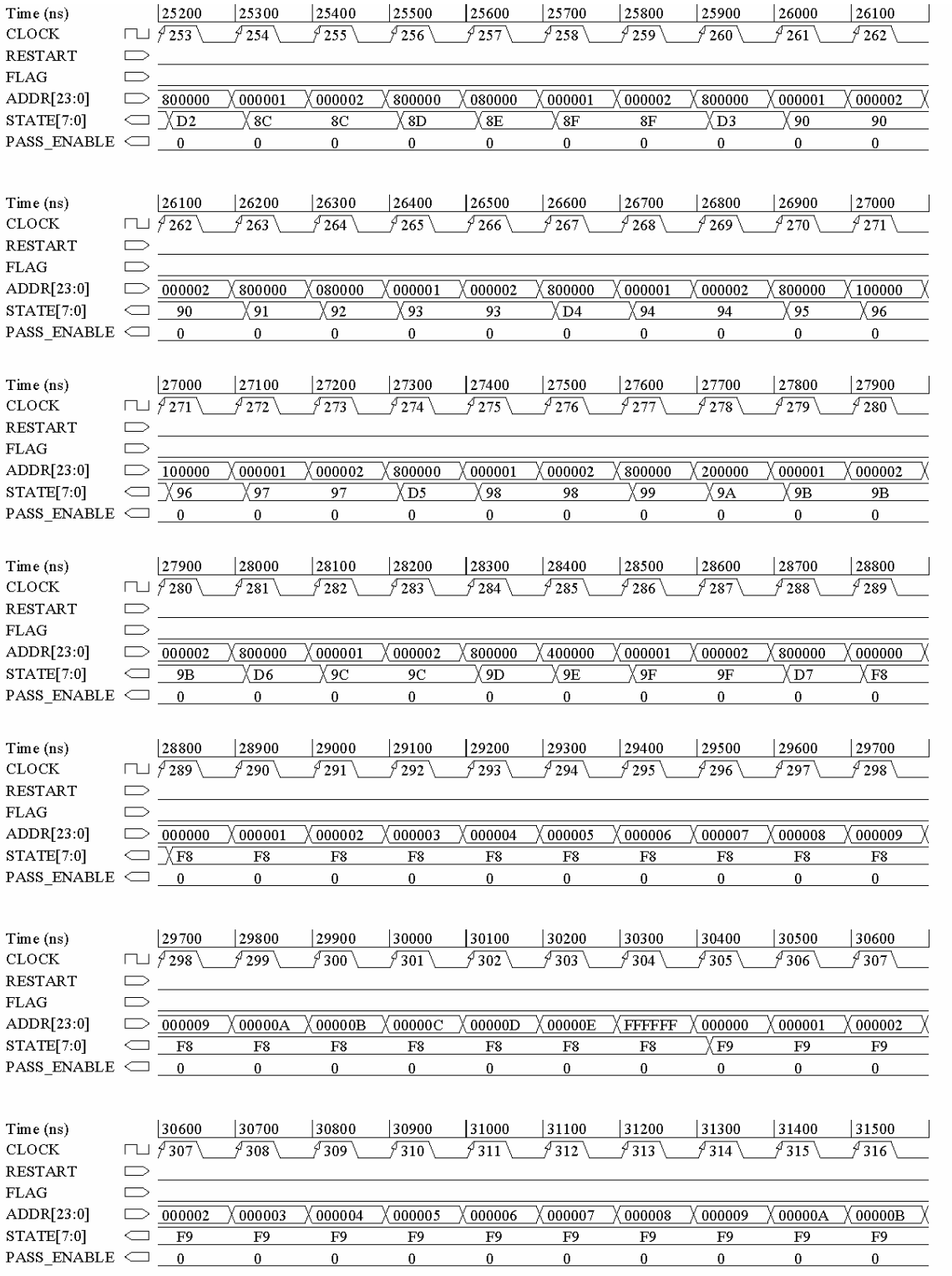
2. Test-Bench Waveform

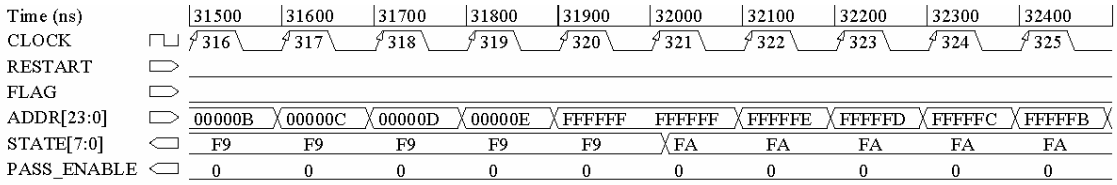


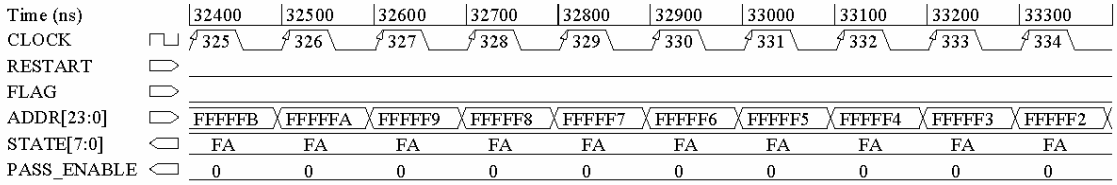

Time (ns) 6300 6400 6500 6600 6700 6800 6900 7000 7100 7200
CLOCK 64 65 66 67 68 69 70 71 72 73
RESTART
FLAG 0 0
ADDR[23:0] 000008 000010 000020 000040 000080 000100 000200 000400 000800 001000
STATE[7:0] 40 40 40 40 40 40 40 40 40 40
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 7200 7300 7400 7500 7600 7700 7800 7900 8000 8100
CLOCK 73 74 75 76 77 78 79 80 81 82
RESTART
FLAG 0 0
ADDR[23:0] 001000 002000 004000 008000 010000 020000 040000 080000 100000 200000
STATE[7:0] 40 40 40 40 40 40 40 40 40 40
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 8100 8200 8300 8400 8500 8600 8700 8800 8900 9000
CLOCK 82 83 84 85 86 87 88 89 90 91
RESTART
FLAG 1 0
ADDR[23:0] 200000 400000 800000 000001 000001 000002 000004 000008 000010 000020
STATE[7:0] 40 40 41 42 43 43 43 43 43 43
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 9000 9100 9200 9300 9400 9500 9600 9700 9800 9900
CLOCK 91 92 93 94 95 96 97 98 99 100
RESTART
FLAG
ADDR[23:0] 000020 000040 000080 000100 000200 000400 000800 001000 002000 004000
STATE[7:0] 43 43 43 43 43 43 43 43 43 43
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 9900 10000 10100 10200 10300 10400 10500 10600 10700 10800
CLOCK 100 101 102 103 104 105 106 107 108 109
RESTART
FLAG
ADDR[23:0] 004000 008000 010000 020000 040000 080000 100000 200000 400000 800000
STATE[7:0] 43 43 43 43 43 43 43 43 43 C0
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 10800 10900 11000 11100 11200 11300 11400 11500 11600 11700
CLOCK 109 110 111 112 113 114 115 116 117 118
RESTART
FLAG 1 0
ADDR[23:0] 800000 000001 000002 000004 800000 000002 000001 000002 000004 800000
STATE[7:0] C0 44 44 44 45 46 47 47 47 C1
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 11700 11800 11900 12000 12100 12200 12300 12400 12500 12600
CLOCK 118 119 120 121 122 123 124 125 126 127
RESTART
FLAG 1 0
ADDR[23:0] 800000 000001 000002 000004 800000 000004 000001 000002 000004 800000
STATE[7:0] C1 48 48 48 49 4A 4B 4B 4B C2
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0

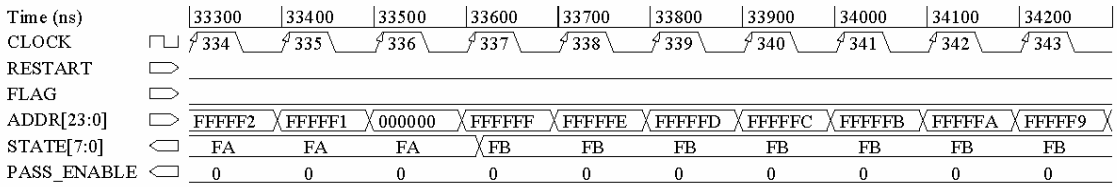

Time (ns) 12600 12700 12800 12900 13000 13100 13200 13300 13400 13500
CLOCK 127 128 129 130 131 132 133 134 135 136
RESTART
FLAG 0
ADDR[23:0] 800000 000001 000002 000004 800000 000008 000001 000002 000004 800000
STATE[7:0] C2 4C 4C 4C 4D 4E 4F 4F 4F C3
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 13500 13600 13700 13800 13900 14000 14100 14200 14300 14400
CLOCK 136 137 138 139 140 141 142 143 144 145
RESTART
FLAG
ADDR[23:0] 800000 000001 000002 000004 800000 000010 000001 000002 000004 800000
STATE[7:0] C3 50 50 50 51 52 53 53 53 C4
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 14400 14500 14600 14700 14800 14900 15000 15100 15200 15300
CLOCK 145 146 147 148 149 150 151 152 153 154
RESTART
FLAG
ADDR[23:0] 800000 000001 000002 000004 800000 000020 000001 000002 000004 800000
STATE[7:0] C4 54 54 54 55 56 57 57 57 C5
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 15300 15400 15500 15600 15700 15800 15900 16000 16100 16200
CLOCK 154 155 156 157 158 159 160 161 162 163
RESTART
FLAG
ADDR[23:0] 800000 000001 000002 000004 800000 000040 000001 000002 000004 800000
STATE[7:0] C5 58 58 58 59 5A 5B 5B 5B C6
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 16200 16300 16400 16500 16600 16700 16800 16900 17000 17100
CLOCK 163 164 165 166 167 168 169 170 171 172
RESTART
FLAG
ADDR[23:0] 800000 000001 000002 000004 800000 000080 000001 000002 000004 800000
STATE[7:0] C6 5C 5C 5C 5D 5E 5F 5F 5F C7
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 17100 17200 17300 17400 17500 17600 17700 17800 17900 18000
CLOCK 172 173 174 175 176 177 178 179 180 181
RESTART
FLAG
ADDR[23:0] 800000 000001 000002 000004 800000 000100 000001 000002 000004 800000
STATE[7:0] C7 60 60 60 61 62 63 63 63 C8
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 18000 18100 18200 18300 18400 18500 18600 18700 18800 18900
CLOCK 181 182 183 184 185 186 187 188 189 190
RESTART
FLAG
ADDR[23:0] 800000 000001 000002 000004 800000 000200 000001 000002 000004 800000
STATE[7:0] C8 64 64 64 65 66 67 67 67 C9
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0

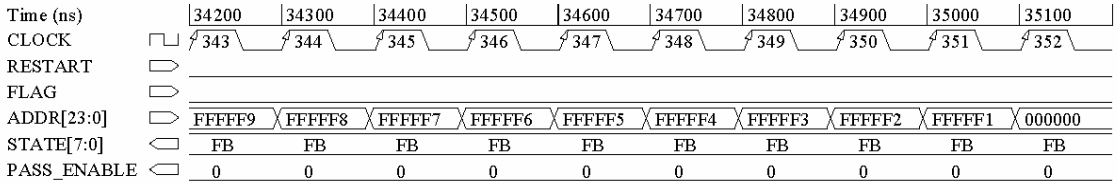

Time (ns) 18900 19000 19100 19200 19300 19400 19500 19600 19700 19800
CLOCK 190 191 192 193 194 195 196 197 198 199
RESTART
FLAG
ADDR[23:0] 800000 000001 000002 800000 000400 000001 000002 800000 000001 000002
STATE[7:0] C9 68 68 69 6A 6B 6B CA 6C 6C
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 19800 19900 20000 20100 20200 20300 20400 20500 20600 20700
CLOCK 199 200 201 202 203 204 205 206 207 208
RESTART
FLAG
ADDR[23:0] 000002 800000 000800 000001 000002 800000 000001 000002 800000 001000
STATE[7:0] 6C 6D 6E 6F 6F CB 70 70 71 72
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 20700 20800 20900 21000 21100 21200 21300 21400 21500 21600
CLOCK 208 209 210 211 212 213 214 215 216 217
RESTART
FLAG
ADDR[23:0] 001000 000001 000002 800000 000001 000002 800000 002000 000001 000002
STATE[7:0] 72 73 73 CC 74 74 75 76 77 77
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 21600 21700 21800 21900 22000 22100 22200 22300 22400 22500
CLOCK 217 218 219 220 221 222 223 224 225 226
RESTART
FLAG
ADDR[23:0] 000002 800000 000001 000002 800000 004000 000001 000002 800000 000001
STATE[7:0] 77 CD 78 78 79 7A 7B 7B CE 7C
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 22500 22600 22700 22800 22900 23000 23100 23200 23300 23400
CLOCK 226 227 228 229 230 231 232 233 234 235
RESTART
FLAG
ADDR[23:0] 000001 000002 800000 008000 000001 000002 800000 000001 000002 800000
STATE[7:0] 7C 7C 7D 7E 7F 7F CF 80 80 81
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 23400 23500 23600 23700 23800 23900 24000 24100 24200 24300
CLOCK 235 236 237 238 239 240 241 242 243 244
RESTART
FLAG
ADDR[23:0] 800000 010000 000001 000002 800000 000001 000002 800000 020000 000001
STATE[7:0] 81 82 83 83 D0 84 84 85 86 87
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 24300 24400 24500 24600 24700 24800 24900 25000 25100 25200
CLOCK 244 245 246 247 248 249 250 251 252 253
RESTART
FLAG
ADDR[23:0] 000001 000002 800000 000001 000002 800000 040000 000001 000002 800000
STATE[7:0] 87 87 D1 88 88 89 8A 8B 8B D2
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0

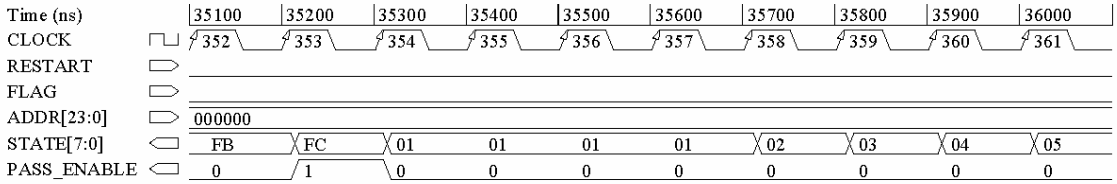

Time (ns) 25200 25300 25400 25500 25600 25700 25800 25900 26000 26100
CLOCK 253 254 255 256 257 258 259 260 261 262
RESTART
FLAG
ADDR[23:0] 800000 000001 000002 800000 080000 000001 000002 800000 000001 000002
STATE[7:0] D2 8C 8C 8D 8E 8F 8F D3 90 90
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 26100 26200 26300 26400 26500 26600 26700 26800 26900 27000
CLOCK 262 263 264 265 266 267 268 269 270 271
RESTART
FLAG
ADDR[23:0] 000002 800000 080000 000001 000002 800000 000001 000002 800000 100000
STATE[7:0] 90 91 92 93 93 D4 94 94 95 96
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 27000 27100 27200 27300 27400 27500 27600 27700 27800 27900
CLOCK 271 272 273 274 275 276 277 278 279 280
RESTART
FLAG
ADDR[23:0] 100000 000001 000002 800000 000001 000002 800000 200000 000001 000002
STATE[7:0] 96 97 97 D5 98 98 99 9A 9B 9B
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 27900 28000 28100 28200 28300 28400 28500 28600 28700 28800
CLOCK 280 281 282 283 284 285 286 287 288 289
RESTART
FLAG
ADDR[23:0] 000002 800000 000001 000002 800000 400000 000001 000002 800000 000000
STATE[7:0] 9B D6 9C 9C 9D 9E 9F 9F D7 F8
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 28800 28900 29000 29100 29200 29300 29400 29500 29600 29700
CLOCK 289 290 291 292 293 294 295 296 297 298
RESTART
FLAG
ADDR[23:0] 000000 000001 000002 000003 000004 000005 000006 000007 000008 000009
STATE[7:0] F8 F8 F8 F8 F8 F8 F8 F8 F8 F8
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 29700 29800 29900 30000 30100 30200 30300 30400 30500 30600
CLOCK 298 299 300 301 302 303 304 305 306 307
RESTART
FLAG
ADDR[23:0] 000009 00000A 00000B 00000C 00000D 00000E FFFFFF 000000 000001 000002
STATE[7:0] F8 F8 F8 F8 F8 F8 F8 F9 F9 F9
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 30600 30700 30800 30900 31000 31100 31200 31300 31400 31500
CLOCK 307 308 309 310 311 312 313 314 315 316
RESTART
FLAG
ADDR[23:0] 000002 000003 000004 000005 000006 000007 000008 000009 00000A 00000B
STATE[7:0] F9 F9 F9 F9 F9 F9 F9 F9 F9 F9
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0

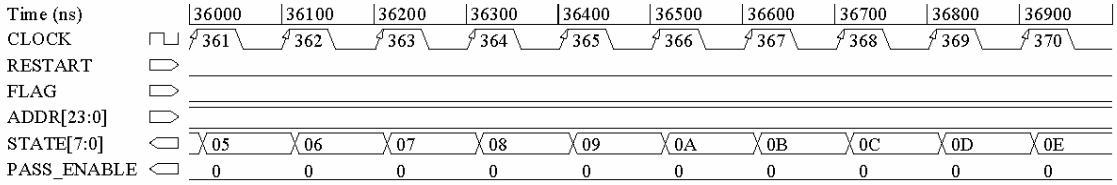

Time (ns) 31500 31600 31700 31800 31900 32000 32100 32200 32300 32400
CLOCK 316 317 318 319 320 321 322 323 324 325
RESTART
FLAG
ADDR[23:0] 00000B 00000C 00000D 00000E FFFFFF FFFFFF FFFFFE FFFFFD FFFFFC FFFFFB
STATE[7:0] F9 F9 F9 F9 F9 FA FA FA FA FA
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0

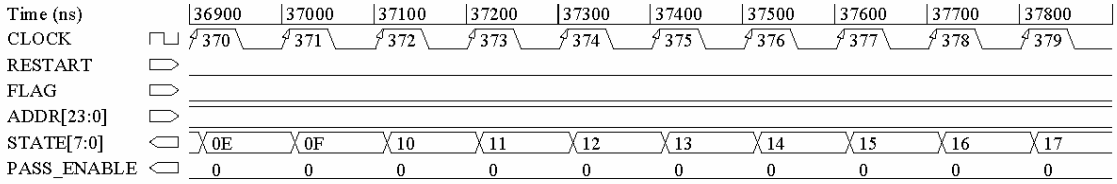

Time (ns) 32400 32500 32600 32700 32800 32900 33000 33100 33200 33300
CLOCK 325 326 327 328 329 330 331 332 333 334
RESTART
FLAG
ADDR[23:0] FFFFFB FFFFFA FFFFF9 FFFFF8 FFFFF7 FFFFF6 FFFFF5 FFFFF4 FFFFF3 FFFFF2
STATE[7:0] FA FA FA FA FA FA FA FA FA FA
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0

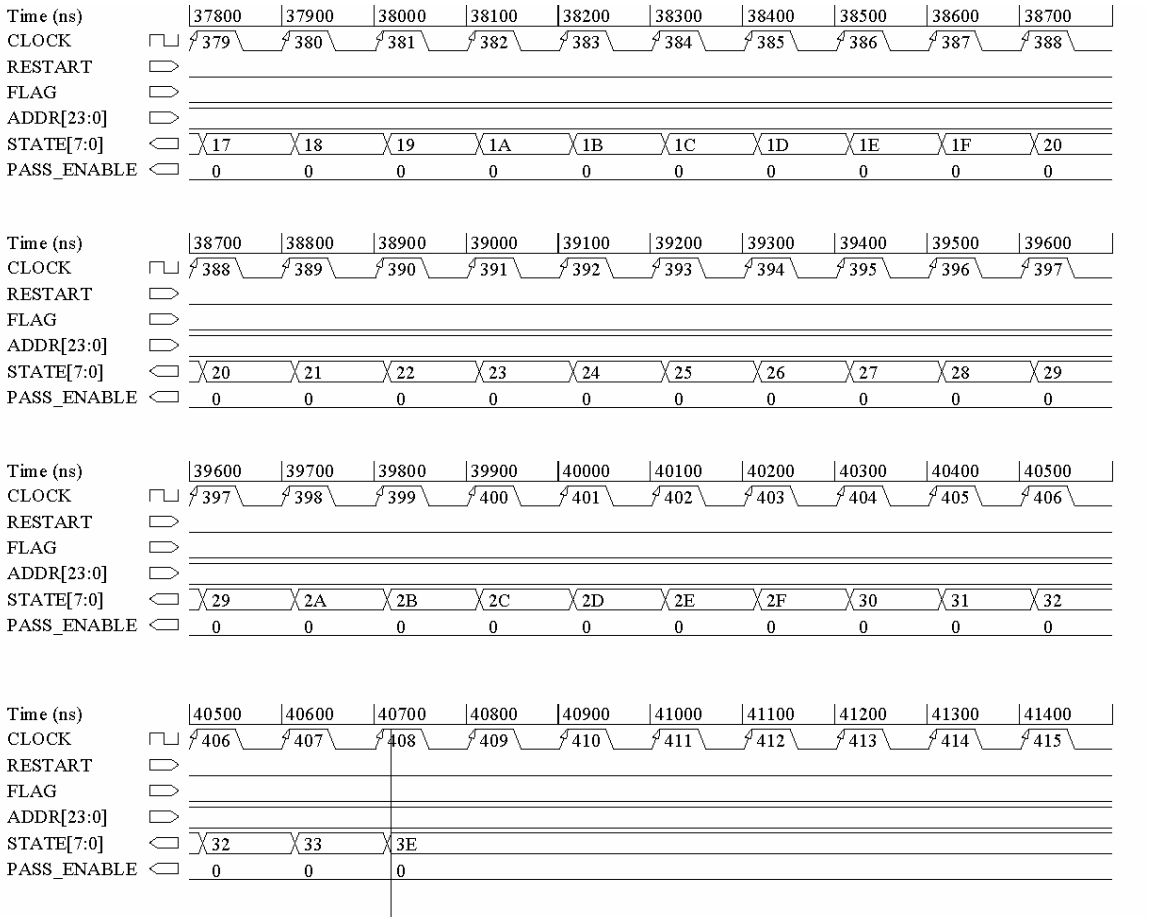

Time (ns) 33300 33400 33500 33600 33700 33800 33900 34000 34100 34200
CLOCK 334 335 336 337 338 339 340 341 342 343
RESTART
FLAG
ADDR[23:0] FFFFF2 FFFFF1 000000 FFFFFF FFFFFE FFFFFD FFFFFC FFFFFB FFFFFA FFFFF9
STATE[7:0] FA FA FA FB FB FB FB FB FB FB
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0


Time (ns) 34200 34300 34400 34500 34600 34700 34800 34900 35000 35100
CLOCK 343 344 345 346 347 348 349 350 351 352
RESTART
FLAG
ADDR[23:0] FFFFF9 FFFFF8 FFFFF7 FFFFF6 FFFFF5 FFFFF4 FFFFF3 FFFFF2 FFFFF1 000000
STATE[7:0] FB FB FB FB FB FB FB FB FB FB
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0


Time (ns) 35100 35200 35300 35400 35500 35600 35700 35800 35900 36000
CLOCK 352 353 354 355 356 357 358 359 360 361
RESTART
FLAG
ADDR[23:0] 000000
STATE[7:0] FB FC 01 01 01 01 02 03 04 05
PASS_ENABLE 0 1 0 0 0 0 0 0 0 0

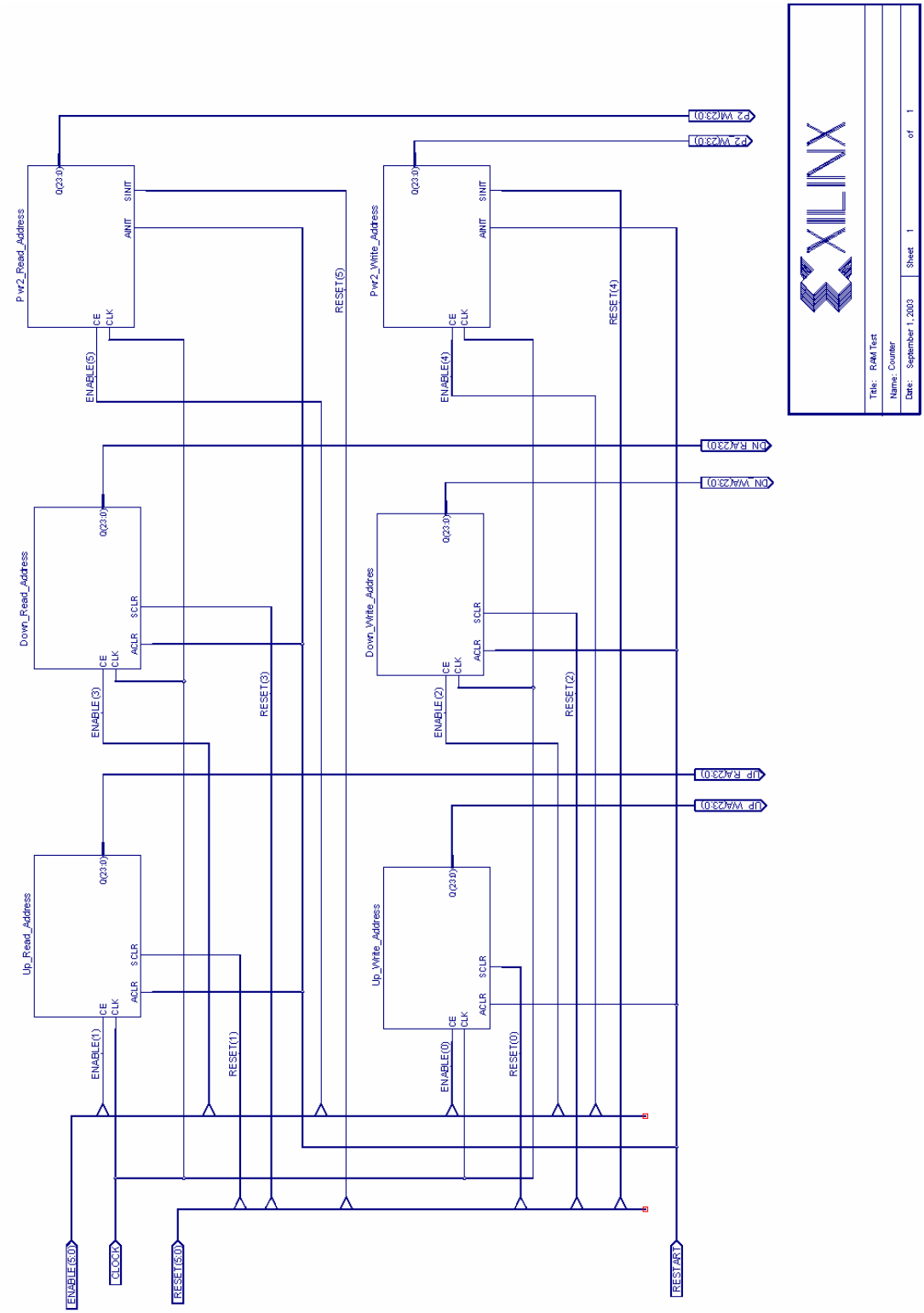

Time (ns) 36000 36100 36200 36300 36400 36500 36600 36700 36800 36900
CLOCK 361 362 363 364 365 366 367 368 369 370
RESTART
FLAG
ADDR[23:0]
STATE[7:0] 05 06 07 08 09 0A 0B 0C 0D 0E
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0


Time (ns) 36900 37000 37100 37200 37300 37400 37500 37600 37700 37800
CLOCK 370 371 372 373 374 375 376 377 378 379
RESTART
FLAG
ADDR[23:0]
STATE[7:0] 0E 0F 10 11 12 13 14 15 16 17
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0

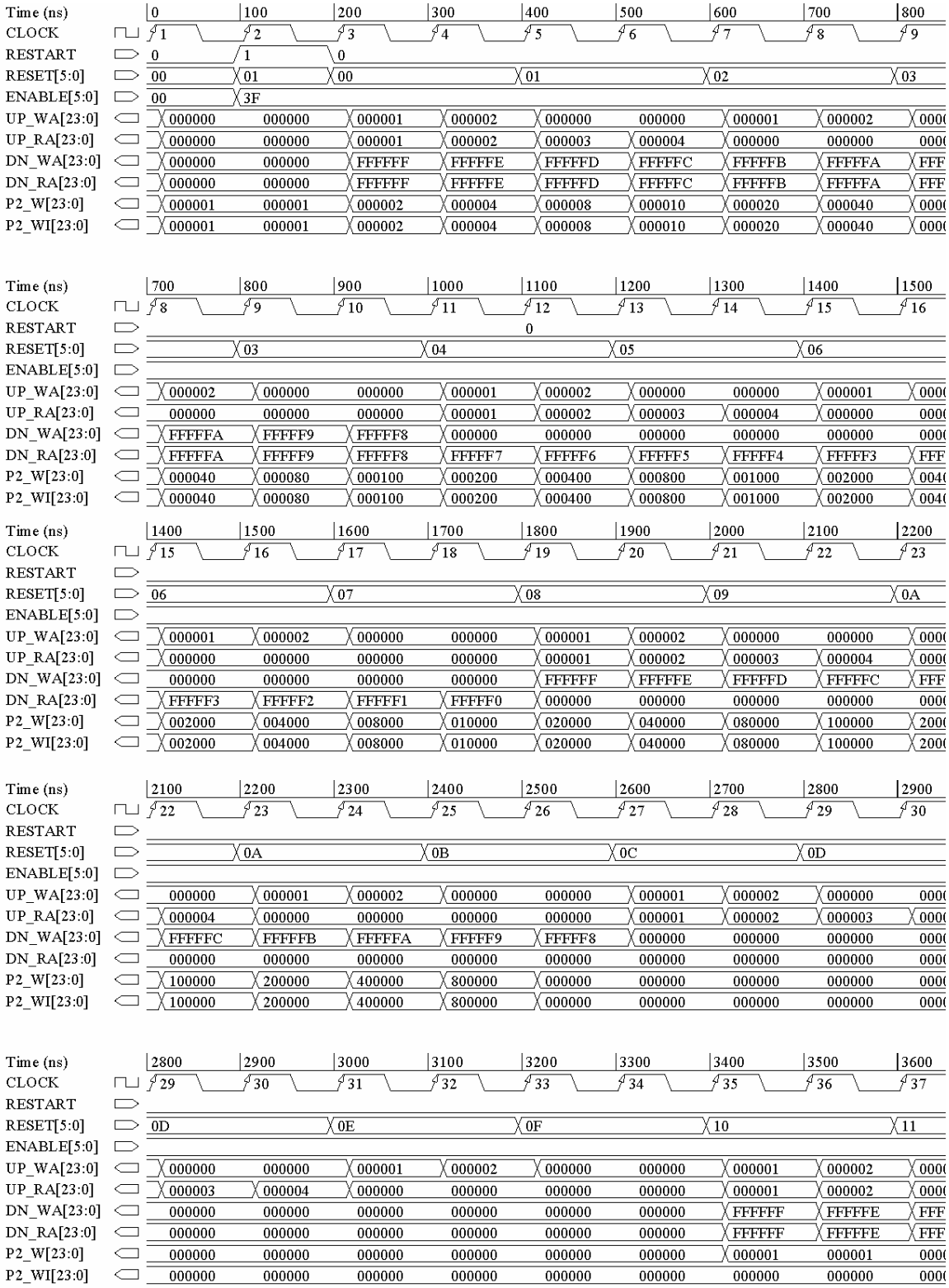

Time (ns) 37800 37900 38000 38100 38200 38300 38400 38500 38600 38700
CLOCK 379 380 381 382 383 384 385 386 387 388
RESTART
FLAG
ADDR[23:0]
STATE[7:0] 17 18 19 1A 1B 1C 1D 1E 1F 20
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 38700 38800 38900 39000 39100 39200 39300 39400 39500 39600
CLOCK 388 389 390 391 392 393 394 395 396 397
RESTART
FLAG
ADDR[23:0]
STATE[7:0] 20 21 22 23 24 25 26 27 28 29
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 39600 39700 39800 39900 40000 40100 40200 40300 40400 40500
CLOCK 397 398 399 400 401 402 403 404 405 406
RESTART
FLAG
ADDR[23:0]
STATE[7:0] 29 2A 2B 2C 2D 2E 2F 30 31 32
PASS_ENABLE 0 0 0 0 0 0 0 0 0 0
Time (ns) 40500 40600 40700 40800 40900 41000 41100 41200 41300 41400
CLOCK 406 407 408 409 410 411 412 413 414 415
RESTART
FLAG
ADDR[23:0]
STATE[7:0] 32 33 3E
PASS_ENABLE 0 0 0

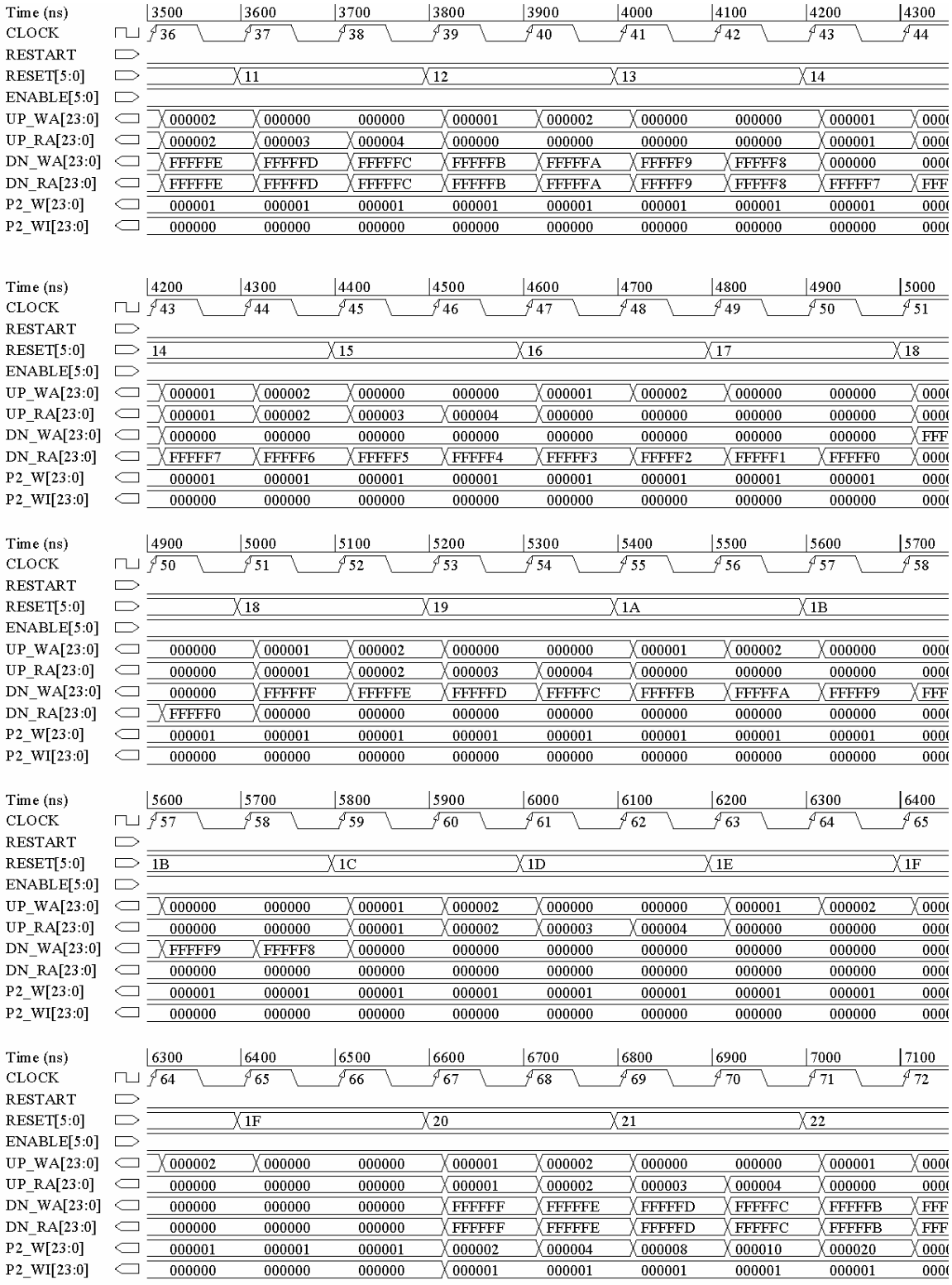
E. ADDRESS COUNTER MODULE

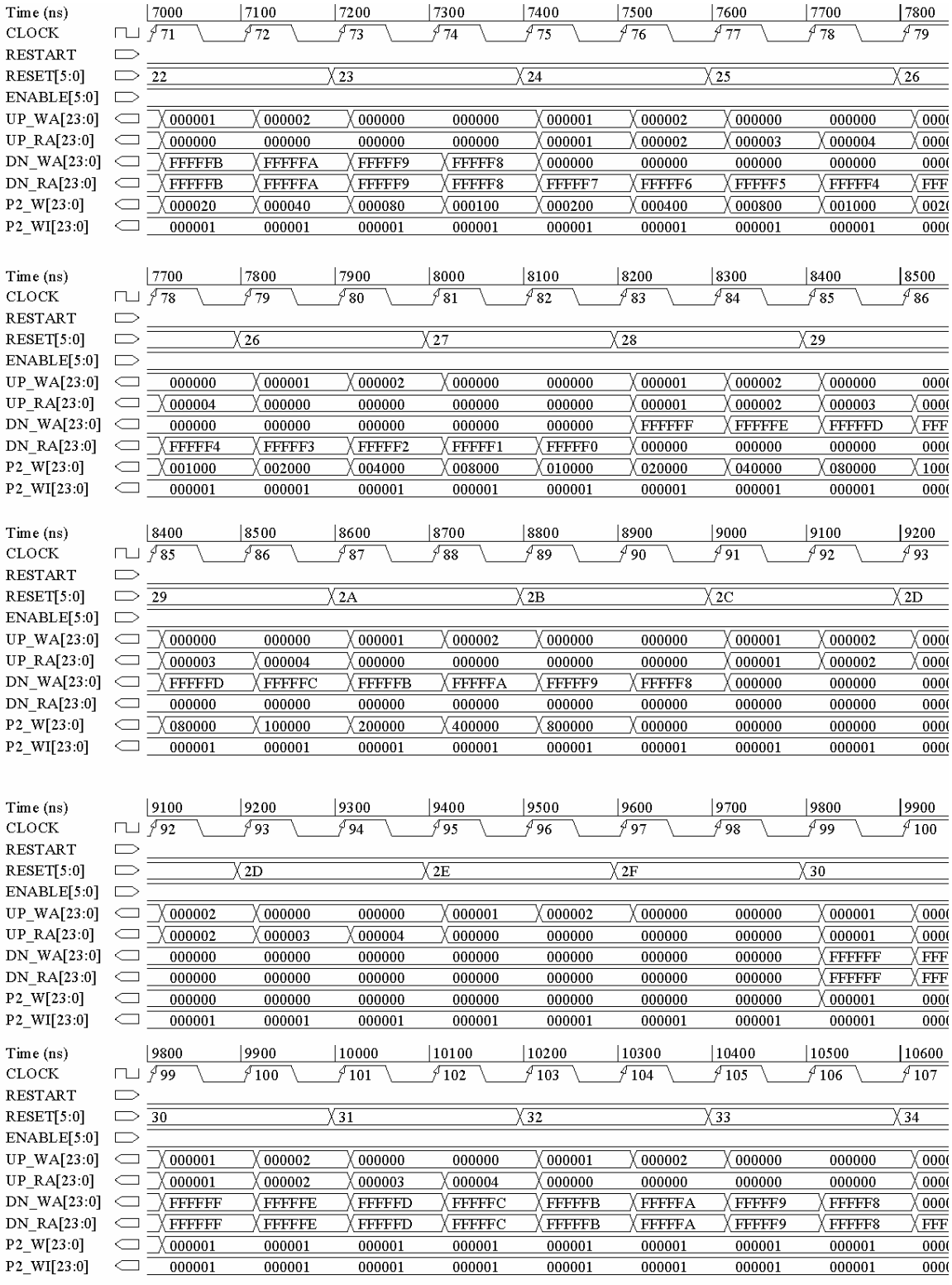
1. Schematic Diagram

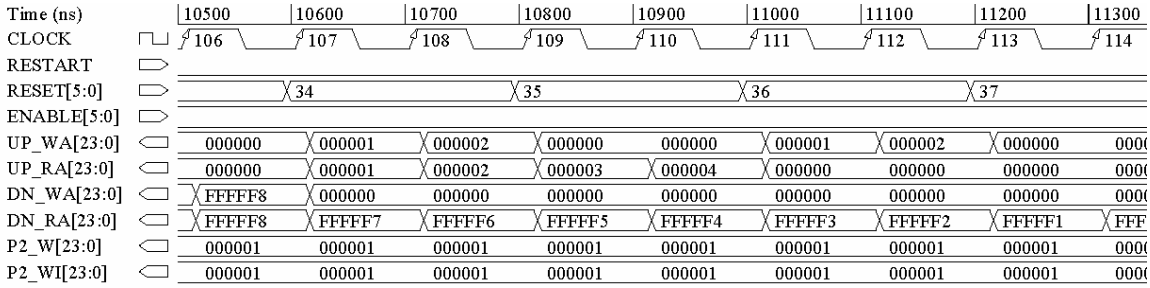


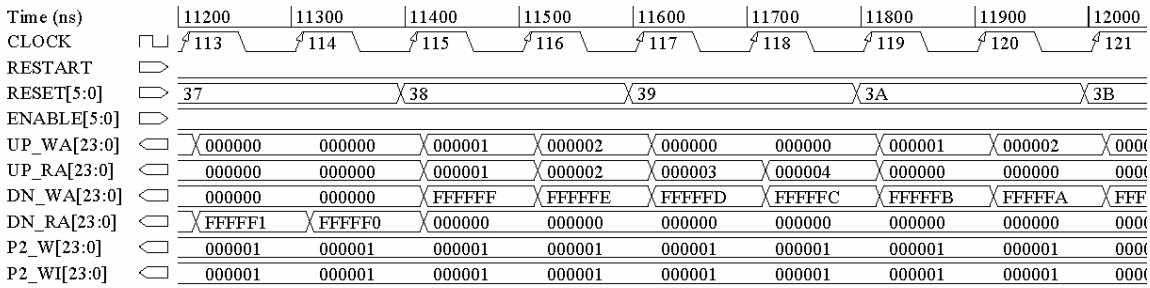
2. Test-Bench Waveform

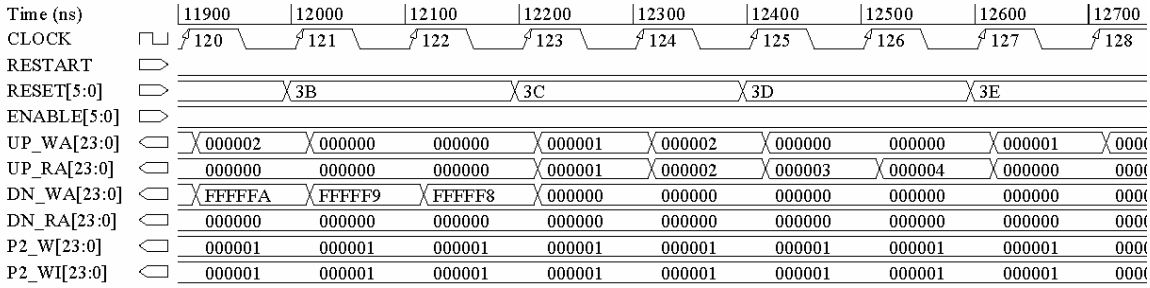


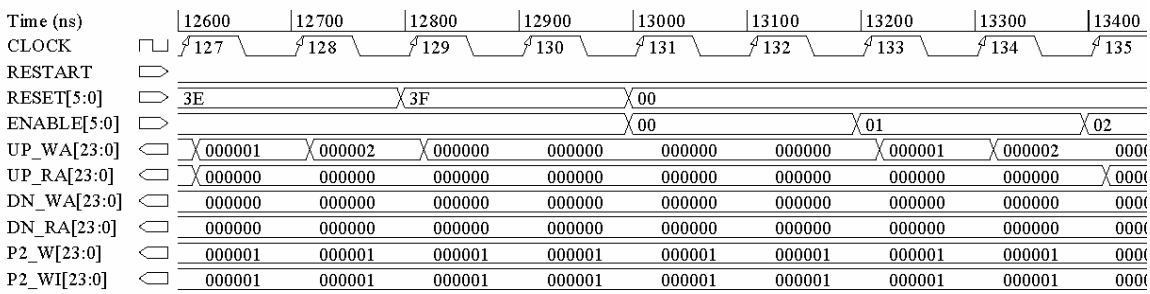

Time (ns) 3500 3600 3700 3800 3900 4000 4100 4200 4300
CLOCK 36 37 38 39 40 41 42 43 44
RESTART
RESET[5:0] 11 12 13 14
ENABLE[5:0]
UP_WA[23:0] 000002 000000 000000 000001 000002 000000 000000 000001 000(
UP_RA[23:0] 000002 000003 000004 000000 000000 000000 000000 000001 000(
DN_WA[23:0] FFFFFE FFFFFD FFFFFC FFFFFB FFFFFA FFFFF9 FFFFF8 000000 000(
DN_RA[23:0] FFFFFE FFFFFD FFFFFC FFFFFB FFFFFA FFFFF9 FFFFF8 FFFFF7 FFF
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000(
Time (ns) 4200 4300 4400 4500 4600 4700 4800 4900 5000
CLOCK 43 44 45 46 47 48 49 50 51
RESTART
RESET[5:0] 14 15 16 17 18
ENABLE[5:0]
UP_WA[23:0] 000001 000002 000000 000000 000001 000002 000000 000000 000(
UP_RA[23:0] 000001 000002 000003 000004 000000 000000 000000 000000 000(
DN_WA[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 FFF
DN_RA[23:0] FFFFF7 FFFFF6 FFFFF5 FFFFF4 FFFFF3 FFFFF2 FFFFF1 FFFFF0 000(
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000(
Time (ns) 4900 5000 5100 5200 5300 5400 5500 5600 5700
CLOCK 50 51 52 53 54 55 56 57 58
RESTART
RESET[5:0] 18 19 1A 1B
ENABLE[5:0]
UP_WA[23:0] 000000 000001 000002 000000 000000 000001 000002 000000 000(
UP_RA[23:0] 000000 000001 000002 000003 000004 000000 000000 000000 000(
DN_WA[23:0] 000000 FFFFFF FFFFFE FFFFFD FFFFFC FFFFFB FFFFFA FFFFF9 FFF
DN_RA[23:0] FFFFF0 000000 000000 000000 000000 000000 000000 000000 000(
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000(
Time (ns) 5600 5700 5800 5900 6000 6100 6200 6300 6400
CLOCK 57 58 59 60 61 62 63 64 65
RESTART
RESET[5:0] 1B 1C 1D 1E 1F
ENABLE[5:0]
UP_WA[23:0] 000000 000000 000001 000002 000000 000000 000001 000002 000(
UP_RA[23:0] 000000 000000 000001 000002 000003 000004 000000 000000 000(
DN_WA[23:0] FFFFF9 FFFFF8 000000 000000 000000 000000 000000 000000 000(
DN_RA[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000(
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000(
Time (ns) 6300 6400 6500 6600 6700 6800 6900 7000 7100
CLOCK 64 65 66 67 68 69 70 71 72
RESTART
RESET[5:0] 1F 20 21 22
ENABLE[5:0]
UP_WA[23:0] 000002 000000 000000 000001 000002 000000 000000 000001 000(
UP_RA[23:0] 000000 000000 000000 000001 000002 000003 000004 000000 000(
DN_WA[23:0] 000000 000000 000000 FFFFFF FFFFFE FFFFFD FFFFFC FFFFFB FFF
DN_RA[23:0] 000000 000000 000000 FFFFFF FFFFFE FFFFFD FFFFFC FFFFFB FFF
P2_W[23:0] 000001 000001 000001 000002 000004 000008 000010 000020 000(
P2_WI[23:0] 000000 000000 000000 000001 000001 000001 000001 000001 000(

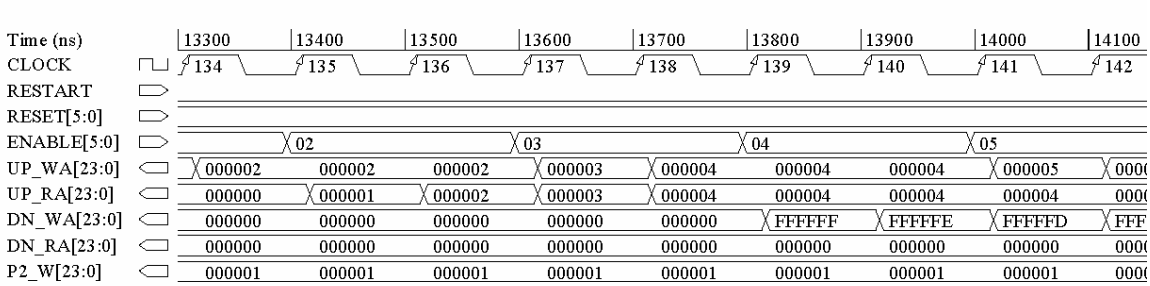

Time (ns) 7000 7100 7200 7300 7400 7500 7600 7700 7800
CLOCK 71 72 73 74 75 76 77 78 79
RESTART
RESET[5:0] 22 23 24 25 26
ENABLE[5:0]
UP_WA[23:0] 000001 000002 000000 000000 000001 000002 000000 000000 000
UP_RA[23:0] 000000 000000 000000 000000 000001 000002 000003 000004 000
DN_WA[23:0] FFFFFB FFFFFA FFFFF9 FFFFF8 000000 000000 000000 000000 000
DN_RA[23:0] FFFFFB FFFFFA FFFFF9 FFFFF8 FFFFF7 FFFFF6 FFFFF5 FFFFF4 FFF
P2_W[23:0] 000020 000040 000080 000100 000200 000400 000800 001000 002
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000
Time (ns) 7700 7800 7900 8000 8100 8200 8300 8400 8500
CLOCK 78 79 80 81 82 83 84 85 86
RESTART
RESET[5:0] 26 27 28 29
ENABLE[5:0]
UP_WA[23:0] 000000 000001 000002 000000 000000 000001 000002 000000 000
UP_RA[23:0] 000004 000000 000000 000000 000000 000001 000002 000003 000
DN_WA[23:0] 000000 000000 000000 000000 000000 FFFFFF FFFFFE FFFFFD FFF
DN_RA[23:0] FFFFF4 FFFFF3 FFFFF2 FFFFF1 FFFFF0 000000 000000 000000 000
P2_W[23:0] 001000 002000 004000 008000 010000 020000 040000 080000 100
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000
Time (ns) 8400 8500 8600 8700 8800 8900 9000 9100 9200
CLOCK 85 86 87 88 89 90 91 92 93
RESTART
RESET[5:0] 29 2A 2B 2C 2D
ENABLE[5:0]
UP_WA[23:0] 000000 000000 000001 000002 000000 000000 000001 000002 000
UP_RA[23:0] 000003 000004 000000 000000 000000 000000 000001 000002 000
DN_WA[23:0] FFFFFD FFFFFC FFFFFB FFFFFA FFFFF9 FFFFF8 000000 000000 000
DN_RA[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
P2_W[23:0] 080000 100000 200000 400000 800000 000000 000000 000000 000
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000
Time (ns) 9100 9200 9300 9400 9500 9600 9700 9800 9900
CLOCK 92 93 94 95 96 97 98 99 100
RESTART
RESET[5:0] 2D 2E 2F 30
ENABLE[5:0]
UP_WA[23:0] 000002 000000 000000 000001 000002 000000 000000 000001 000
UP_RA[23:0] 000002 000003 000004 000000 000000 000000 000000 000001 000
DN_WA[23:0] 000000 000000 000000 000000 000000 000000 000000 FFFFFF FFF
DN_RA[23:0] 000000 000000 000000 000000 000000 000000 000000 FFFFFF FFF
P2_W[23:0] 000000 000000 000000 000000 000000 000000 000000 000001 000
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000
Time (ns) 9800 9900 10000 10100 10200 10300 10400 10500 10600
CLOCK 99 100 101 102 103 104 105 106 107
RESTART
RESET[5:0] 30 31 32 33 34
ENABLE[5:0]
UP_WA[23:0] 000001 000002 000000 000000 000001 000002 000000 000000 000
UP_RA[23:0] 000001 000002 000003 000004 000000 000000 000000 000000 000
DN_WA[23:0] FFFFFF FFFFFE FFFFFD FFFFFC FFFFFB FFFFFA FFFFF9 FFFFF8 000
DN_RA[23:0] FFFFFF FFFFFE FFFFFD FFFFFC FFFFFB FFFFFA FFFFF9 FFFFF8 FFF
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000

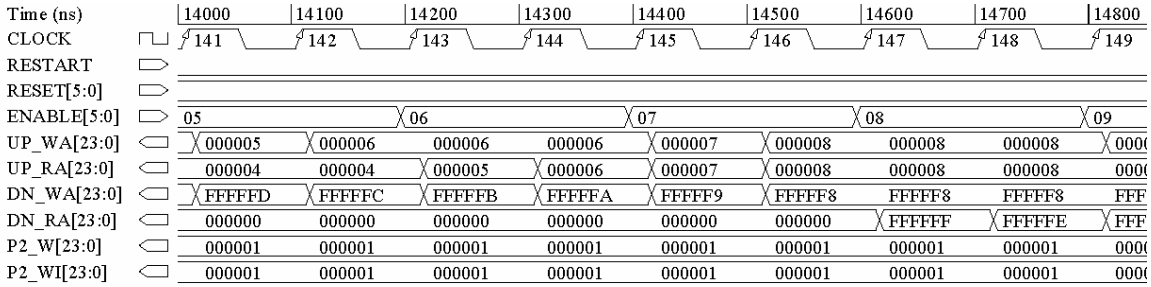

Time (ns) 10500 10600 10700 10800 10900 11000 11100 11200 11300
CLOCK 106 107 108 109 110 111 112 113 114
RESTART
RESET[5:0] 34 35 36 37
ENABLE[5:0]
UP_WA[23:0] 000000 000001 000002 000000 000000 000001 000002 000000 000(
UP_RA[23:0] 000000 000001 000002 000003 000004 000000 000000 000000 000(
DN_WA[23:0] FFFFF8 000000 000000 000000 000000 000000 000000 000000 000(
DN_RA[23:0] FFFFF8 FFFFF7 FFFFF6 FFFFF5 FFFFF4 FFFFF3 FFFFF2 FFFFF1 FFF
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(

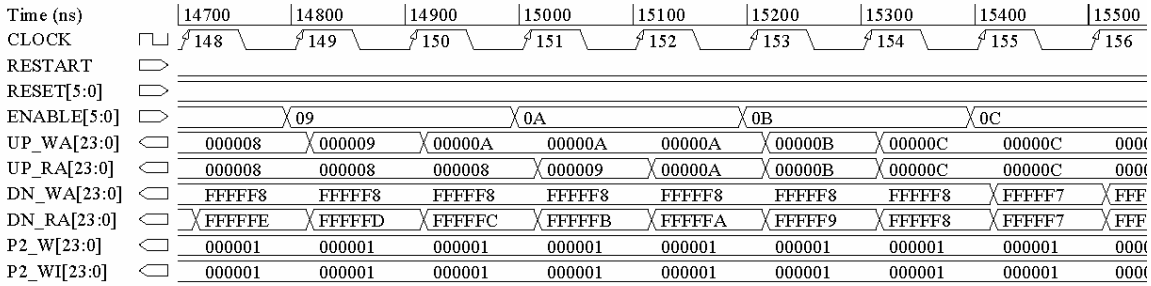

Time (ns) 11200 11300 11400 11500 11600 11700 11800 11900 12000
CLOCK 113 114 115 116 117 118 119 120 121
RESTART
RESET[5:0] 37 38 39 3A 3B
ENABLE[5:0]
UP_WA[23:0] 000000 000000 000001 000002 000000 000000 000001 000002 000(
UP_RA[23:0] 000000 000000 000001 000002 000003 000004 000000 000000 000(
DN_WA[23:0] 000000 000000 FFFFFF FFFFFE FFFFFD FFFFFC FFFFFB FFFFFA FFF
DN_RA[23:0] FFFFF1 FFFFF0 000000 000000 000000 000000 000000 000000 000(
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(

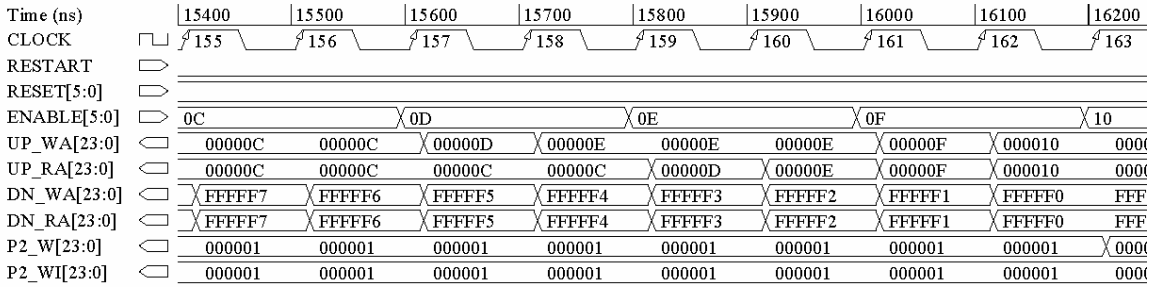

Time (ns) 11900 12000 12100 12200 12300 12400 12500 12600 12700
CLOCK 120 121 122 123 124 125 126 127 128
RESTART
RESET[5:0] 3B 3C 3D 3E
ENABLE[5:0]
UP_WA[23:0] 000002 000000 000000 000001 000002 000000 000000 000001 000(
UP_RA[23:0] 000000 000000 000000 000001 000002 000003 000004 000000 000(
DN_WA[23:0] FFFFFA FFFFF9 FFFFF8 000000 000000 000000 000000 000000 000(
DN_RA[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000(
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(

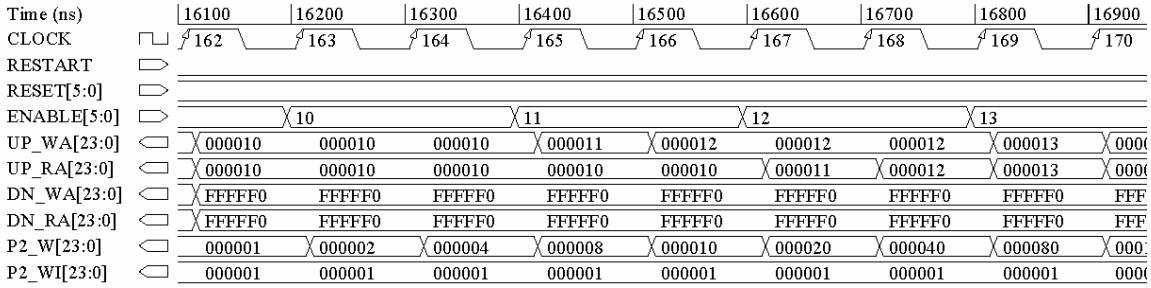

Time (ns) 12600 12700 12800 12900 13000 13100 13200 13300 13400
CLOCK 127 128 129 130 131 132 133 134 135
RESTART
RESET[5:0] 3E 3F 00
ENABLE[5:0] 00 01 02
UP_WA[23:0] 000001 000002 000000 000000 000000 000000 000001 000002 000(
UP_RA[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000(
DN_WA[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000(
DN_RA[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000(
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(

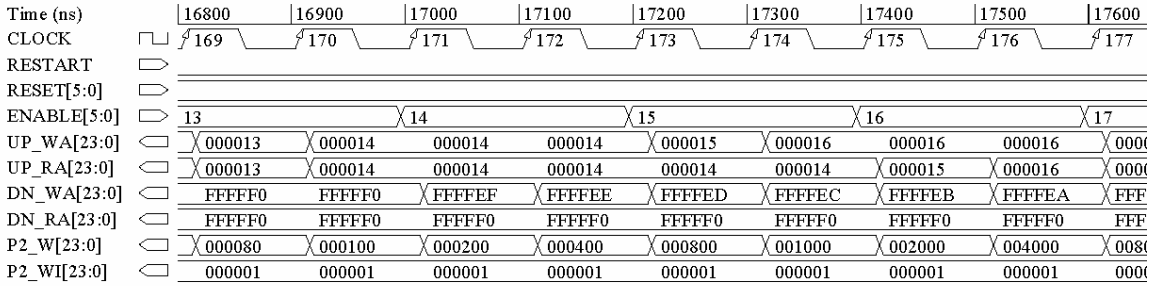

Time (ns) 13300 13400 13500 13600 13700 13800 13900 14000 14100
CLOCK 134 135 136 137 138 139 140 141 142
RESTART
RESET[5:0]
ENABLE[5:0] 02 03 04 05
UP_WA[23:0] 000002 000002 000002 000003 000004 000004 000004 000005 000(
UP_RA[23:0] 000000 000001 000002 000003 000004 000004 000004 000004 000(
DN_WA[23:0] 000000 000000 000000 000000 000000 FFFFFF FFFFFE FFFFFD FFF
DN_RA[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000(
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(

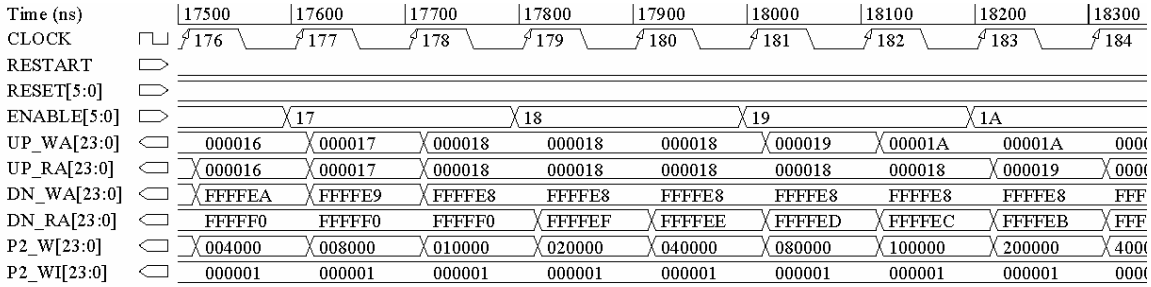

Time (ns) 14000 14100 14200 14300 14400 14500 14600 14700 14800
CLOCK 141 142 143 144 145 146 147 148 149
RESTART
RESET[5:0]
ENABLE[5:0] 05 06 07 08 09
UP_WA[23:0] 000005 000006 000006 000006 000007 000008 000008 000008 000(
UP_RA[23:0] 000004 000004 000005 000006 000007 000008 000008 000008 000(
DN_WA[23:0] FFFFFD FFFFFC FFFFFB FFFFFA FFFFF9 FFFFF8 FFFFF8 FFFFF8 FFF
DN_RA[23:0] 000000 000000 000000 000000 000000 000000 FFFFFF FFFFFE FFF
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(

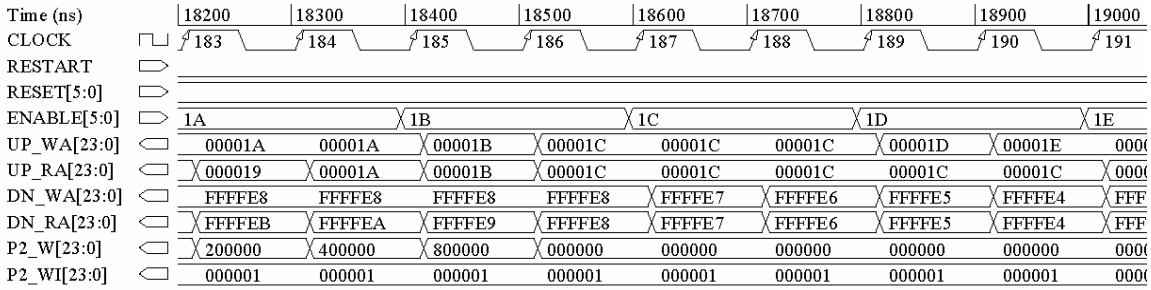

Time (ns) 14700 14800 14900 15000 15100 15200 15300 15400 15500
CLOCK 148 149 150 151 152 153 154 155 156
RESTART
RESET[5:0]
ENABLE[5:0] 09 0A 0B 0C
UP_WA[23:0] 000008 000009 00000A 00000A 00000A 00000B 00000C 00000C 000(
UP_RA[23:0] 000008 000008 000008 000009 00000A 00000B 00000C 00000C 000(
DN_WA[23:0] FFFFF8 FFFFF8 FFFFF8 FFFFF8 FFFFF8 FFFFF8 FFFFF8 FFFFF7 FFF
DN_RA[23:0] FFFFFE FFFFFD FFFFFC FFFFFB FFFFFA FFFFF9 FFFFF8 FFFFF7 FFF
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(

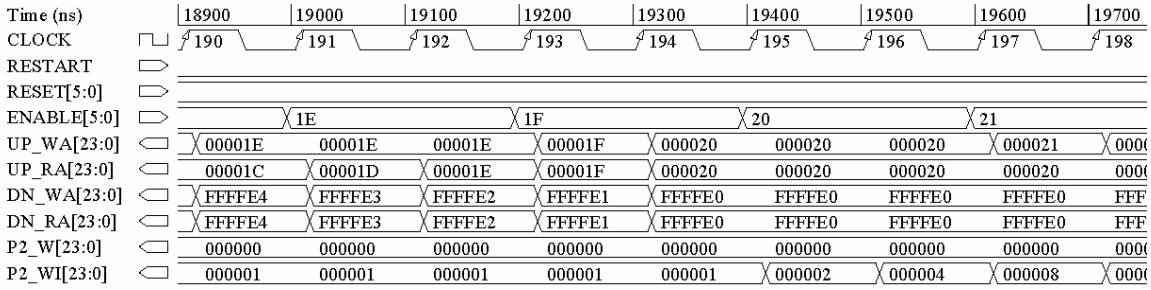

Time (ns) 15400 15500 15600 15700 15800 15900 16000 16100 16200
CLOCK 155 156 157 158 159 160 161 162 163
RESTART
RESET[5:0]
ENABLE[5:0] 0C 0D 0E 0F 10
UP_WA[23:0] 00000C 00000C 00000D 00000E 00000E 00000E 00000F 000010 000(
UP_RA[23:0] 00000C 00000C 00000C 00000C 00000D 00000E 00000F 000010 000(
DN_WA[23:0] FFFFF7 FFFFF6 FFFFF5 FFFFF4 FFFFF3 FFFFF2 FFFFF1 FFFFF0 FFF
DN_RA[23:0] FFFFF7 FFFFF6 FFFFF5 FFFFF4 FFFFF3 FFFFF2 FFFFF1 FFFFF0 FFF
P2_W[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(

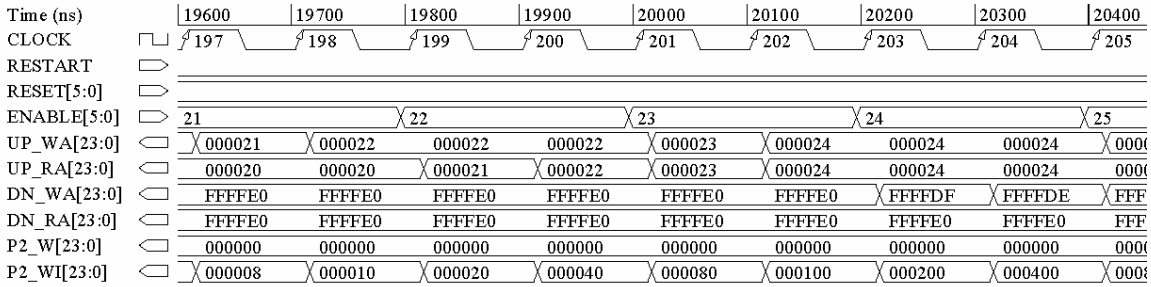

Time (ns) 16100 16200 16300 16400 16500 16600 16700 16800 16900
CLOCK 162 163 164 165 166 167 168 169 170
RESTART
RESET[5:0]
ENABLE[5:0] 10 11 12 13
UP_WA[23:0] 000010 000010 000010 000011 000012 000012 000012 000013 000(
UP_RA[23:0] 000010 000010 000010 000010 000010 000011 000012 000013 000(
DN_WA[23:0] FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFF
DN_RA[23:0] FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFF
P2_W[23:0] 000001 000002 000004 000008 000010 000020 000040 000080 000
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(

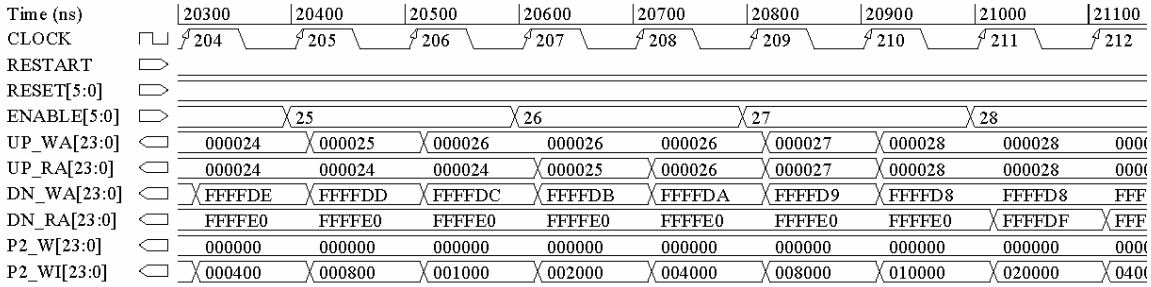

Time (ns) 16800 16900 17000 17100 17200 17300 17400 17500 17600
CLOCK 169 170 171 172 173 174 175 176 177
RESTART
RESET[5:0]
ENABLE[5:0] 13 14 15 16 17
UP_WA[23:0] 000013 000014 000014 000014 000015 000016 000016 000016 000(
UP_RA[23:0] 000013 000014 000014 000014 000014 000014 000015 000016 000(
DN_WA[23:0] FFFFF0 FFFFF0 FFFFEF FFFFEE FFFFED FFFFEC FFFFEB FFFFEA FFF
DN_RA[23:0] FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFFFF0 FFF
P2_W[23:0] 000080 000100 000200 000400 000800 001000 002000 004000 008(
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000(

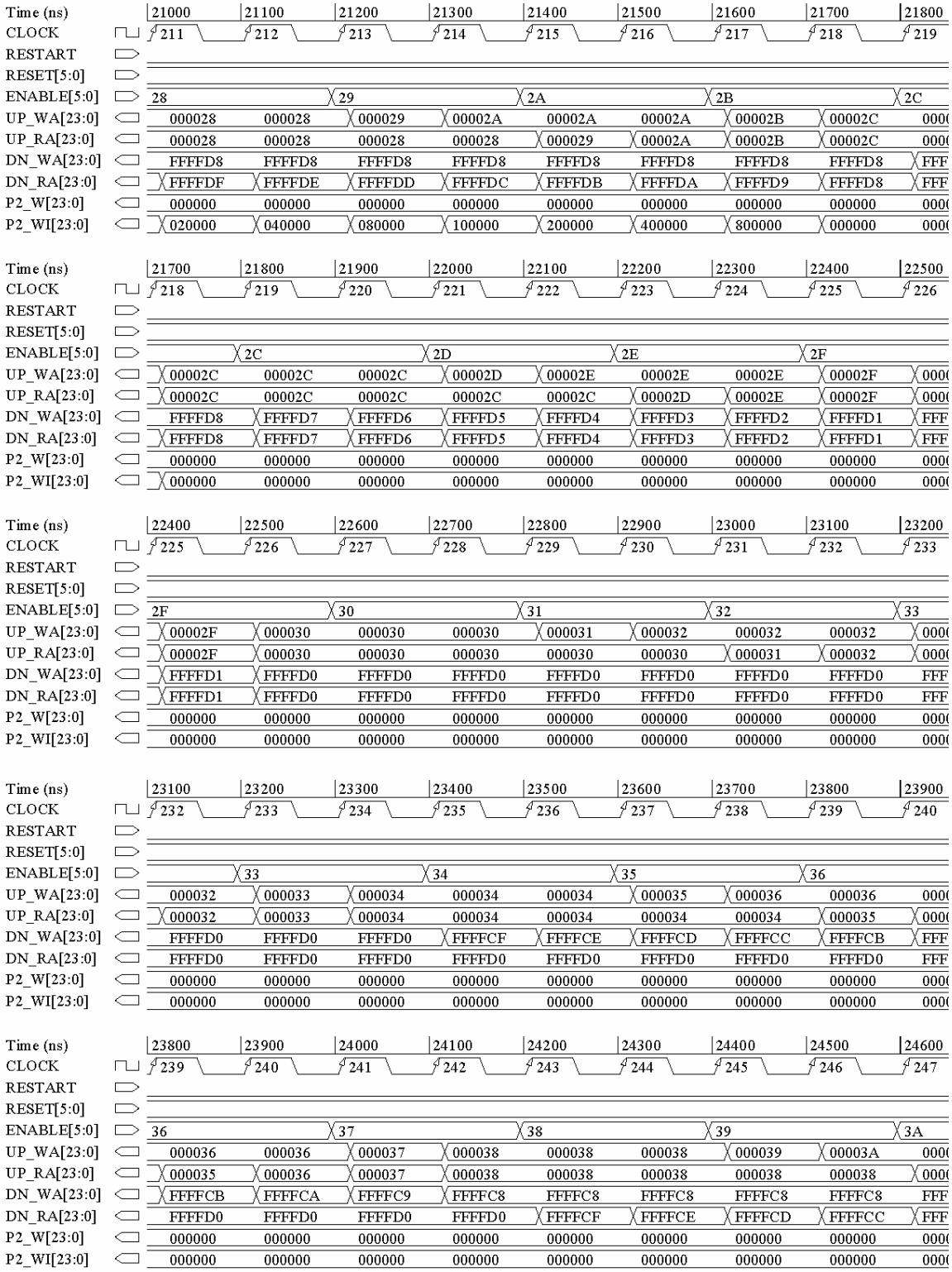

Time (ns) 17500 17600 17700 17800 17900 18000 18100 18200 18300
CLOCK 176 177 178 179 180 181 182 183 184
RESTART
RESET[5:0]
ENABLE[5:0] 17 18 19 1A
UP_WA[23:0] 000016 000017 000018 000018 000018 000019 00001A 00001A 000
UP_RA[23:0] 000016 000017 000018 000018 000018 000018 000018 000019 000
DN_WA[23:0] FFFFEA FFFFE9 FFFFE8 FFFFE8 FFFFE8 FFFFE8 FFFFE8 FFFFE8 FFF
DN_RA[23:0] FFFFF0 FFFFF0 FFFFF0 FFFFEF FFFFEE FFFFED FFFFEC FFFFEB FFF
P2_W[23:0] 004000 008000 010000 020000 040000 080000 100000 200000 400
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000

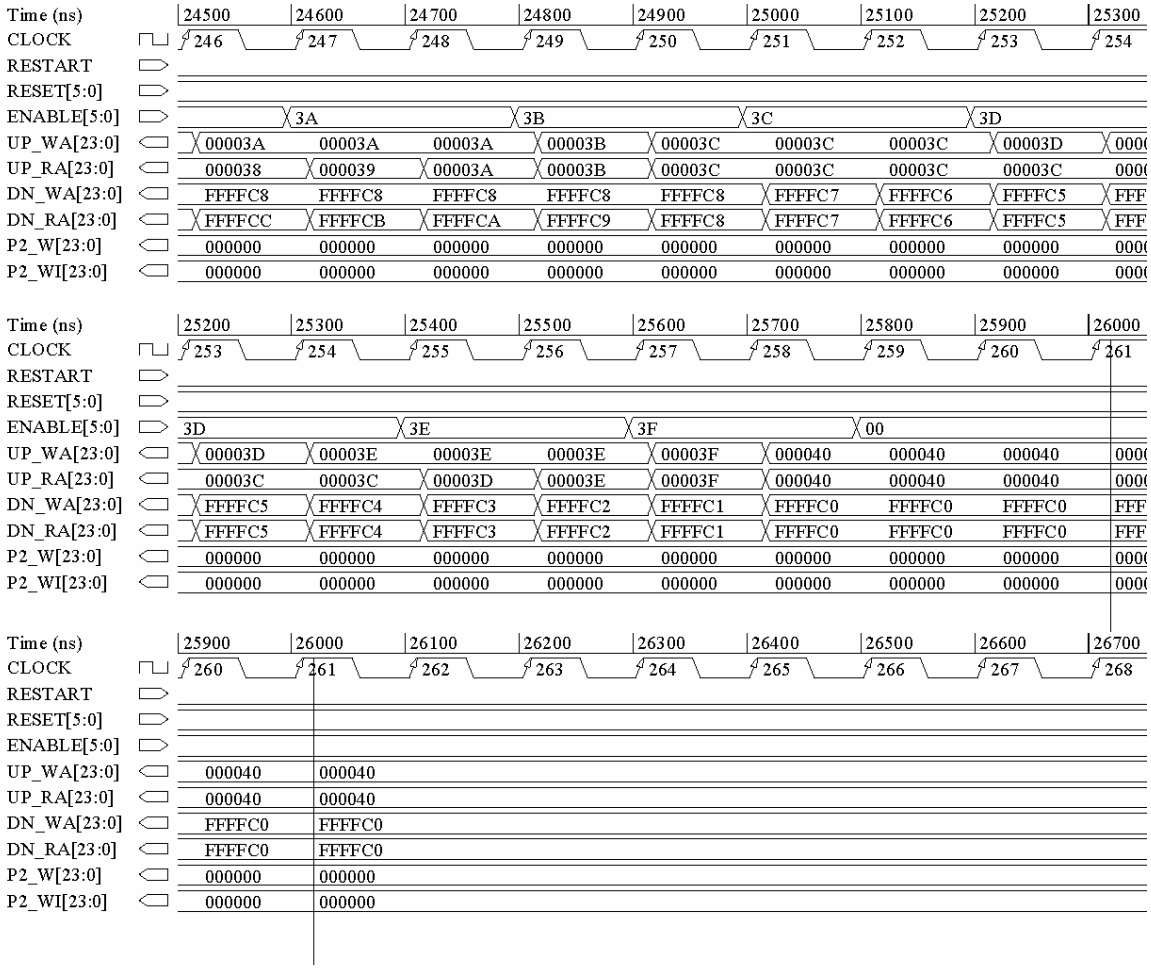

Time (ns) 18200 18300 18400 18500 18600 18700 18800 18900 19000
CLOCK 183 184 185 186 187 188 189 190 191
RESTART
RESET[5:0]
ENABLE[5:0] 1A 1B 1C 1D 1E
UP_WA[23:0] 00001A 00001A 00001B 00001C 00001C 00001C 00001D 00001E 000
UP_RA[23:0] 000019 00001A 00001B 00001C 00001C 00001C 00001C 00001C 000
DN_WA[23:0] FFFFE8 FFFFE8 FFFFE8 FFFFE8 FFFFE7 FFFFE6 FFFFE5 FFFFE4 FFF
DN_RA[23:0] FFFFEB FFFFEA FFFFE9 FFFFE8 FFFFE7 FFFFE6 FFFFE5 FFFFE4 FFF
P2_W[23:0] 200000 400000 800000 000000 000000 000000 000000 000000 000
P2_WI[23:0] 000001 000001 000001 000001 000001 000001 000001 000001 000


Time (ns) 18900 19000 19100 19200 19300 19400 19500 19600 19700
CLOCK 190 191 192 193 194 195 196 197 198
RESTART
RESET[5:0]
ENABLE[5:0] 1E 1F 20 21
UP_WA[23:0] 00001E 00001E 00001E 00001F 000020 000020 000020 000021 000
UP_RA[23:0] 00001C 00001D 00001E 00001F 000020 000020 000020 000020 000
DN_WA[23:0] FFFFE4 FFFFE3 FFFFE2 FFFFE1 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFF
DN_RA[23:0] FFFFE4 FFFFE3 FFFFE2 FFFFE1 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFF
P2_W[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
P2_WI[23:0] 000001 000001 000001 000001 000001 000002 000004 000008 000


Time (ns) 19600 19700 19800 19900 20000 20100 20200 20300 20400
CLOCK 197 198 199 200 201 202 203 204 205
RESTART
RESET[5:0]
ENABLE[5:0] 21 22 23 24 25
UP_WA[23:0] 000021 000022 000022 000022 000023 000024 000024 000024 000
UP_RA[23:0] 000020 000020 000021 000022 000023 000024 000024 000024 000
DN_WA[23:0] FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFDF FFFFDE FFF
DN_RA[23:0] FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFF
P2_W[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
P2_WI[23:0] 000008 000010 000020 000040 000080 000100 000200 000400 000


Time (ns) 20300 20400 20500 20600 20700 20800 20900 21000 21100
CLOCK 204 205 206 207 208 209 210 211 212
RESTART
RESET[5:0]
ENABLE[5:0] 25 26 27 28
UP_WA[23:0] 000024 000025 000026 000026 000026 000027 000028 000028 000
UP_RA[23:0] 000024 000024 000024 000025 000026 000027 000028 000028 000
DN_WA[23:0] FFFFDE FFFFDD FFFFDC FFFFDB FFFFDA FFFFD9 FFFFD8 FFFFD8 FFF
DN_RA[23:0] FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFE0 FFFFDF FFF
P2_W[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
P2_WI[23:0] 000400 000800 001000 002000 004000 008000 010000 020000 040


Time (ns) 21000 21100 21200 21300 21400 21500 21600 21700 21800
CLOCK 211 212 213 214 215 216 217 218 219
RESTART
RESET[5:0]
ENABLE[5:0] 28 29 2A 2B 2C
UP_WA[23:0] 000028 000028 000029 00002A 00002A 00002A 00002B 00002C 000
UP_RA[23:0] 000028 000028 000028 000028 000029 00002A 00002B 00002C 000
DN_WA[23:0] FFFFD8 FFFFD8 FFFFD8 FFFFD8 FFFFD8 FFFFD8 FFFFD8 FFFFD8 FFF
DN_RA[23:0] FFFFDF FFFFDE FFFFDD FFFFDC FFFFDB FFFFDA FFFFD9 FFFFD8 FFF
P2_W[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
P2_WI[23:0] 020000 040000 080000 100000 200000 400000 800000 000000 000
Time (ns) 21700 21800 21900 22000 22100 22200 22300 22400 22500
CLOCK 218 219 220 221 222 223 224 225 226
RESTART
RESET[5:0]
ENABLE[5:0] 2C 2D 2E 2F
UP_WA[23:0] 00002C 00002C 00002C 00002D 00002E 00002E 00002E 00002F 000
UP_RA[23:0] 00002C 00002C 00002C 00002C 00002C 00002D 00002E 00002F 000
DN_WA[23:0] FFFFD8 FFFFD7 FFFFD6 FFFFD5 FFFFD4 FFFFD3 FFFFD2 FFFFD1 FFF
DN_RA[23:0] FFFFD8 FFFFD7 FFFFD6 FFFFD5 FFFFD4 FFFFD3 FFFFD2 FFFFD1 FFF
P2_W[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
P2_WI[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
Time (ns) 22400 22500 22600 22700 22800 22900 23000 23100 23200
CLOCK 225 226 227 228 229 230 231 232 233
RESTART
RESET[5:0]
ENABLE[5:0] 2F 30 31 32 33
UP_WA[23:0] 00002F 000030 000030 000030 000031 000032 000032 000032 000
UP_RA[23:0] 00002F 000030 000030 000030 000030 000030 000031 000032 000
DN_WA[23:0] FFFFD1 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFF
DN_RA[23:0] FFFFD1 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFF
P2_W[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
P2_WI[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
Time (ns) 23100 23200 23300 23400 23500 23600 23700 23800 23900
CLOCK 232 233 234 235 236 237 238 239 240
RESTART
RESET[5:0]
ENABLE[5:0] 33 34 35 36
UP_WA[23:0] 000032 000033 000034 000034 000034 000035 000036 000036 000
UP_RA[23:0] 000032 000033 000034 000034 000034 000034 000034 000035 000
DN_WA[23:0] FFFFD0 FFFFD0 FFFFD0 FFFFCF FFFFCE FFFFCD FFFFCC FFFFCB FFF
DN_RA[23:0] FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFF
P2_W[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
P2_WI[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
Time (ns) 23800 23900 24000 24100 24200 24300 24400 24500 24600
CLOCK 239 240 241 242 243 244 245 246 247
RESTART
RESET[5:0]
ENABLE[5:0] 36 37 38 39 3A
UP_WA[23:0] 000036 000036 000037 000038 000038 000038 000039 00003A 000
UP_RA[23:0] 000035 000036 000037 000038 000038 000038 000038 000038 000
DN_WA[23:0] FFFFCB FFFFCA FFFFC9 FFFFC8 FFFFC8 FFFFC8 FFFFC8 FFFFC8 FFF
DN_RA[23:0] FFFFD0 FFFFD0 FFFFD0 FFFFD0 FFFFCF FFFFCE FFFFCD FFFFCC FFF
P2_W[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000
P2_WI[23:0] 000000 000000 000000 000000 000000 000000 000000 000000 000


Time (ns)
24500
24600
24700
24800
24900
25000
25100
25200
25300
CLOCK
246
247
248
249
250
251
252
253
254
RESTART
RESET[5:0]
ENABLE[5:0]
3A
3B
3C
3D
UP_WA[23:0]
00003A
00003A
00003A
00003B
00003C
00003C
00003C
00003D
UP_RA[23:0]
000038
000039
00003A
00003B
00003C
00003C
00003C
00003C
DN_WA[23:0]
FFFFC8
FFFFC8
FFFFC8
FFFFC8
FFFFC8
FFFFC7
FFFFC6
FFFFC5
DN_RA[23:0]
FFFFCC
FFFFCB
FFFFCA
FFFFC9
FFFFC8
FFFFC7
FFFFC6
FFFFC5
P2_W[23:0]
000000
000000
000000
000000
000000
000000
000000
000000
P2_WI[23:0]
000000
000000
000000
000000
000000
000000
000000
000000
Time (ns)
25200
25300
25400
25500
25600
25700
25800
25900
26000
CLOCK
253
254
255
256
257
258
259
260
261
RESTART
RESET[5:0]
ENABLE[5:0]
3D
3E
3F
00
UP_WA[23:0]
00003D
00003E
00003E
00003E
00003F
000040
000040
000040
UP_RA[23:0]
00003C
00003C
00003D
00003E
00003F
000040
000040
000040
DN_WA[23:0]
FFFFC5
FFFFC4
FFFFC3
FFFFC2
FFFFC1
FFFFC0
FFFFC0
FFFFC0
DN_RA[23:0]
FFFFC5
FFFFC4
FFFFC3
FFFFC2
FFFFC1
FFFFC0
FFFFC0
FFFFC0
P2_W[23:0]
000000
000000
000000
000000
000000
000000
000000
000000
P2_WI[23:0]
000000
000000
000000
000000
000000
000000
000000
000000
Time (ns)
25900
26000
26100
26200
26300
26400
26500
26600
26700
CLOCK
260
261
262
263
264
265
266
267
268
RESTART
RESET[5:0]
ENABLE[5:0]
UP_WA[23:0]
000040
000040
UP_RA[23:0]
000040
000040
DN_WA[23:0]
FFFFC0
FFFFC0
DN_RA[23:0]
FFFFC0
FFFFC0
P2_W[23:0]
000000
000000
P2_WI[23:0]
000000
000000

F. COUNTER-CONTROL MODULE

1. VHDL Code

```
----------------------------------------------------------------------
-- filename: cntr_cntl.vhd
-- written by: Charles Hulme
--
-- This controls the different counters and when they are
-- enabled/disabled     and when they are reset.
--
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity cntr_cntl is
    Port (
                  STATE: in std_logic_vector (7 downto 0);
                  RESET: out STD_LOGIC_vector (5 downto 0);
                  ENABLE: out STD_LOGIC_vector (5 downto 0)
     );
end cntr_cntl;

architecture Behavioral of cntr_cntl is

begin

fcn: process (STATE)

begin

      --Data Test uses a fixed address

      --Start Address Test
      if    (STATE = "01000000" or STATE = "01000100"
            or STATE = "01001000" or STATE = "01001100"
            or STATE = "01010000" or STATE = "01010100"
            or STATE = "01011000" or STATE = "01011100"
            or STATE = "01100000" or STATE = "01100100"
            or STATE = "01101000" or STATE = "01101100"
           or STATE = "01110000" or STATE = "01110100"
           or STATE = "01111000" or STATE = "01111100"
           or STATE = "10000000" or STATE = "10000100"
           or STATE = "10001000" or STATE = "10001100"
           or STATE = "10010000" or STATE = "10010100"
           or STATE = "10011000" or STATE = "10011100") then
            RESET  <= "000000";
            ENABLE <= "010000";

      elsif (STATE = "01000001" or STATE = "01000101"
            or STATE = "01001001" or STATE = "01001101"
            or STATE = "01010001" or STATE = "01010101"
           or STATE = "01011001" or STATE = "01011101"
```

```
     or STATE = "01100001" or STATE = "01100101"
     or STATE = "01101001" or STATE = "01101101"
     or STATE = "01110001" or STATE = "01110101"
     or STATE = "01111001" or STATE = "01111101"
     or STATE = "10000001" or STATE = "10000101"
     or STATE = "10001001" or STATE = "10001101"
     or STATE = "10010001" or STATE = "10010101"
     or STATE = "10011001" or STATE = "10011101") then
      RESET  <= "000000";
      ENABLE <= "010000";

elsif (STATE = "01000010" or STATE = "01000110"
      or STATE = "01001010" or STATE = "01001110"
      or STATE = "01010010" or STATE = "01010110"
     or STATE = "01011010" or STATE = "01011110"
     or STATE = "01100010" or STATE = "01100110"
     or STATE = "01101010" or STATE = "01101110"
     or STATE = "01110010" or STATE = "01110110"
     or STATE = "01111010" or STATE = "01111110"
     or STATE = "10000010" or STATE = "10000110"
     or STATE = "10001010" or STATE = "10001110"
     or STATE = "10010010" or STATE = "10010110"
     or STATE = "10011010" or STATE = "10011110") then
      RESET  <= "010000";
      ENABLE <= "100000";

elsif (STATE = "01000011" or STATE = "01000111"
      or STATE = "01001011" or STATE = "01001111"
      or STATE = "01010011" or STATE = "01010111"
     or STATE = "01011011" or STATE = "01011111"
     or STATE = "01100011" or STATE = "01100111"
     or STATE = "01101011" or STATE = "01101111"
     or STATE = "01110011" or STATE = "01110111"
     or STATE = "01111011" or STATE = "01111111"
     or STATE = "10000011" or STATE = "10000111"
     or STATE = "10001011" or STATE = "10001111"
     or STATE = "10010011" or STATE = "10010111"
     or STATE = "10011011" or STATE = "10011111") then
      RESET  <= "000000";
      ENABLE <= "010000";

elsif (STATE = "11000000" or STATE = "11000001"
      or STATE = "11000010" or STATE = "11000011"
      or STATE = "11000100" or STATE = "11000101"
     or STATE = "11000110" or STATE = "11000111"
     or STATE = "11001000" or STATE = "11001001"
     or STATE = "11001010" or STATE = "11001011"
     or STATE = "11001100" or STATE = "11001101"
     or STATE = "11001110" or STATE = "11001111"
     or STATE = "11010000" or STATE = "11010001"
     or STATE = "11010010" or STATE = "11010011"
     or STATE = "11010100" or STATE = "11010101"
     or STATE = "11010110" or STATE = "11010111") then
      RESET  <= "010000";
```

```
            ENABLE <= "010000";

      --Start Memory Test
      elsif STATE = "11111000" then
            RESET  <= "110000";
            ENABLE <= "000001";

      elsif STATE = "11111001" then
            RESET  <= "000001";
            ENABLE <= "000010";

      elsif STATE = "11111010" then
            RESET  <= "000000";
            ENABLE <= "000100";

      elsif STATE = "11111011" then
            RESET  <= "000000";
            ENABLE <= "001000";

      else
            RESET  <= "111111";
            ENABLE <= "000000";

      end if;

end process;

end Behavioral;
```

G. COUNTER-DECODE MODULE

1. VHDL Code

```
----------------------------------------------------------------------
----
-- filename: counter_decode.vhd
-- written by: Charles Hulme
--
-- This module tells you which address counter is selected based
on
-- current state.
--
----------------------------------------------------------------------
----
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter_decode is
    Port (
            state: in std_logic_vector (7 downto 0);
            address1: in std_logic_vector (23 downto 0);  --UP_WA
            address2: in std_logic_vector (23 downto 0);  --UP_RA
            address3: in std_logic_vector (23 downto 0);  --DN_WA
            address4: in std_logic_vector (23 downto 0);  --DN_RA
            address5: in std_logic_vector (23 downto 0);  --P2_W
            address6: in std_logic_vector (23 downto 0);  --P2_WI
            RTWF: out std_logic;  --Read True,'1', and Write
False,'0'
            ADDR: out STD_LOGIC_VECTOR (23 downto 0)
     );
end counter_decode;

architecture Behavioral of counter_decode is

begin

  fcn: process
(state,address1,address2,address3,address4,address5,address6)

  begin

      --Starting Data Test
      if    (state = "00000010" or state = "00000100"
            or state = "00000110" or state = "00001000"
            or state = "00001010" or state = "00001100"
          or state = "00001110" or state = "00010000"
          or state = "00010010" or state = "00010100"
          or state = "00010110" or state = "00011000"
          or state = "00011010" or state = "00011100"
          or state = "00011110" or state = "00100000"
          or state = "00100010" or state = "00100100"
          or state = "00100110" or state = "00101000"
```

```
          or state = "00101010" or state = "00101100"
          or state = "00101110" or state = "00110000"
          or state = "00110010") then
             ADDR <= "00000000000000000000000000";  --any address
for data test
             RTWF <= '0';  --write pattern to memory

      elsif (state = "00000011" or state = "00000101"
            or state = "00000111" or state = "00001001"
            or state = "00001011" or state = "00001101"
          or state = "00001111" or state = "00010001"
          or state = "00010011" or state = "00010101"
          or state = "00010111" or state = "00011001"
          or state = "00011011" or state = "00011101"
          or state = "00011111" or state = "00100001"
          or state = "00100011" or state = "00100101"
          or state = "00100111" or state = "00101001"
          or state = "00101011" or state = "00101101"
          or state = "00101111" or state = "00110001"
          or state = "00110011") then
             ADDR <= "00000000000000000000000000"; --same address
as above
             RTWF <= '1'; --read back pattern from memory

    --Starting Address Test
      elsif (state = "01000000" or state = "01000100"
            or state = "01001000" or state = "01001100"
            or state = "10100000" or state = "01010100"
          or state = "01011000" or state = "01011100"
          or state = "01100000" or state = "01100100"
          or state = "01101000" or state = "01101100"
          or state = "01110000" or state = "01110100"
          or state = "01111000" or state = "01111100"
          or state = "10000000" or state = "10000100"
          or state = "10001000" or state = "10001100"
          or state = "10010000" or state = "10010100"
          or state = "10011000" or state = "10011100") then
             ADDR <= address5; --Power-of-2 counter for address
test
             RTWF <= '0'; --write pattern to memory

--note there is no statements for the _WL cases, the null state-
ment below
--for the else statement will keep the same address and RTWF
value

      elsif (state = "01000010" or state = "01000110"
            or state = "01001010" or state = "01001110"
            or state = "01010010" or state = "01010110"
          or state = "01011010" or state = "01011110"
          or state = "01100010" or state = "01100110"
          or state = "01101010" or state = "01101110"
          or state = "01110010" or state = "01110110"
          or state = "01111010" or state = "01111110"
```

```
          or state = "10000010" or state = "10000110"
          or state = "10001010" or state = "10001110"
          or state = "10010010" or state = "10010110"
          or state = "10011010" or state = "10011110") then
             ADDR <= address6; --Power-of-2 counter for address
test
             RTWF <= '0'; --write inverted pattern to memory

      elsif (state = "01000011" or state = "01000111"
            or state = "01001011" or state = "01001111"
            or state = "01010011" or state = "01010111"
          or state = "01011011" or state = "01011111"
          or state = "01100011" or state = "01100111"
          or state = "01101011" or state = "01101111"
          or state = "01110011" or state = "01110111"
          or state = "01111011" or state = "01111111"
          or state = "10000011" or state = "10000111"
          or state = "10001011" or state = "10001111"
          or state = "10010011" or state = "10010111"
          or state = "10011011" or state = "10011111") then
             ADDR <= address5; --Reset Power of 2s counter for
address test
             RTWF <= '1'; --read pattern from memory

--note there is no statements for the _RL cases, the null state-
ment below
--for the else statement will keep the same address and RTWF
value

      --Starting Memory Test
      elsif state = "11111000" then
             ADDR <= address1; --Up counter for memory test
             RTWF <= '0'; --write pattern to memory

      elsif state = "11111001"      then
             ADDR <= address2; --Up counter for memory test
             RTWF <= '1'; --read pattern from memory

      elsif state = "11111010" then
             ADDR <= address3; --Down counter for memory test
             RTWF <= '0'; --write pattern to memory

      elsif state = "11111011"      then
             ADDR <= address4; --Down counter for memory test
             RTWF <= '1'; --read pattern from memory

      else
            null;

      end if;

  end process;

end Behavioral;
```

H. COMPARE-ENABLE MODULE

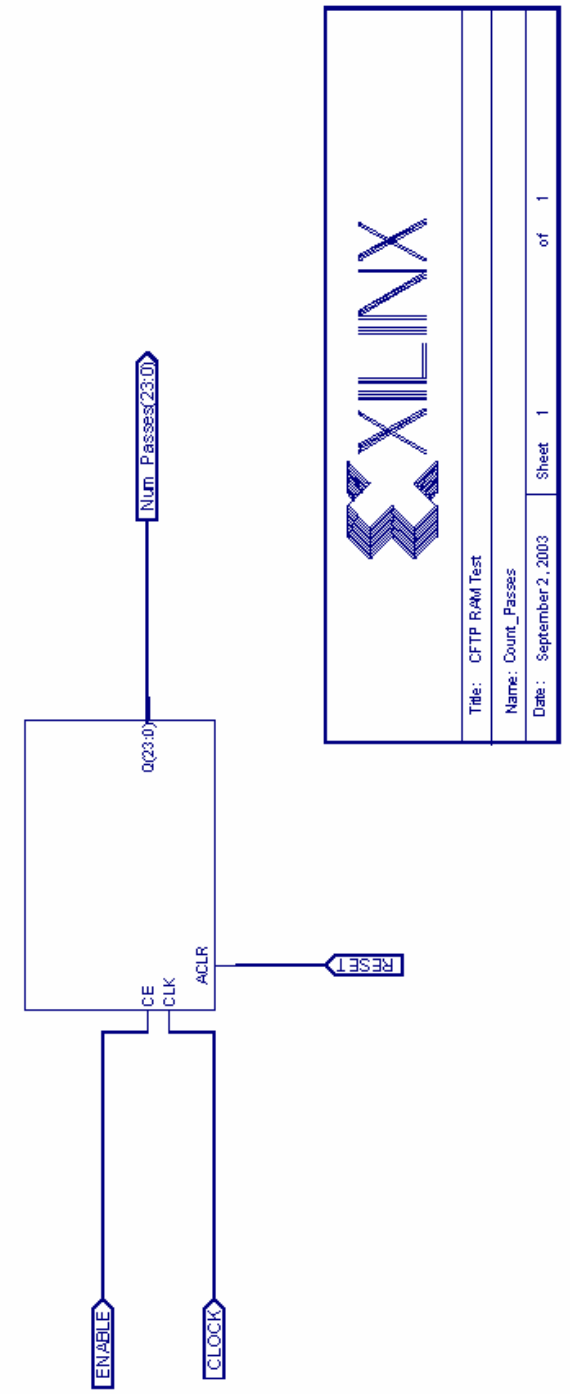
1. VHDL Code

```
----------------------------------------------------------------------
-- filename: comp_en.vhd
-- written by: Charles Hulme
--
-- This module dictates when the comparator can test.  Typically,
-- this occurs any time a read operation is occuring.
--
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity comp_en is
    Port (
                  state: in std_logic_vector (7 downto 0);
                  comp_enable: out STD_LOGIC
     );
end comp_en;

architecture Behavioral of comp_en is

begin

  fcn: process (state)

  begin

      --Staring Data Test
         if (state = "00000011" or state = "00000101"
            or state = "00000111" or state = "00001001"
            or state = "00001011" or state = "00001101"
           or state = "00001111" or state = "00010001"
           or state = "00010011" or state = "00010101"
           or state = "00010111" or state = "00011001"
           or state = "00011011" or state = "00011101"
           or state = "00011111" or state = "00100001"
           or state = "00100011" or state = "00100101"
           or state = "00100111" or state = "00101001"
           or state = "00101011" or state = "00101101"
           or state = "00101111" or state = "00110001"
           or state = "00110011") then
                  comp_enable<='1';

      --Starting Address Test
      elsif (state = "01000011" or state = "01000111"
            or state = "01001011" or state = "01001111"
            or state = "01010011" or state = "01010111"
            or state = "01011011" or state = "01011111"
            or state = "01100011" or state = "01100111"
            or state = "01101011" or state = "01101111"
```

```
            or state = "01110011" or state = "01110111"
            or state = "01111011" or state = "01111111"
            or state = "10000011" or state = "10000111"
            or state = "10001011" or state = "10001111"
            or state = "10010011" or state = "10010111"
            or state = "10011011" or state = "10011111") then
                  comp_enable<='1';

      elsif (state = "11000000" or state = "11000001"
            or state = "11000010" or state = "11000011"
            or state = "11000100" or state = "11000101"
            or state = "11000110" or state = "11000111"
            or state = "110001000" or state = "11001001"
            or state = "11001010" or state = "11001011"
            or state = "11001100" or state = "11001101"
            or state = "11001110" or state = "11001111"
            or state = "11010000" or state = "11010001"
            or state = "11010010" or state = "11010011"
            or state = "11010100" or state = "11010101"
            or state = "11010110" or state = "11010111") then
                  comp_enable<='1';

      --Starting Memory Test
      elsif state = "11111001"      then
            comp_enable<='1';

      elsif state = "11111011"      then
            comp_enable<='1';

      else
            comp_enable<='0';

      end if;

  end process;

end Behavioral;
```

I. PASS-COUNTER MODULE

1. Schematic Diagram



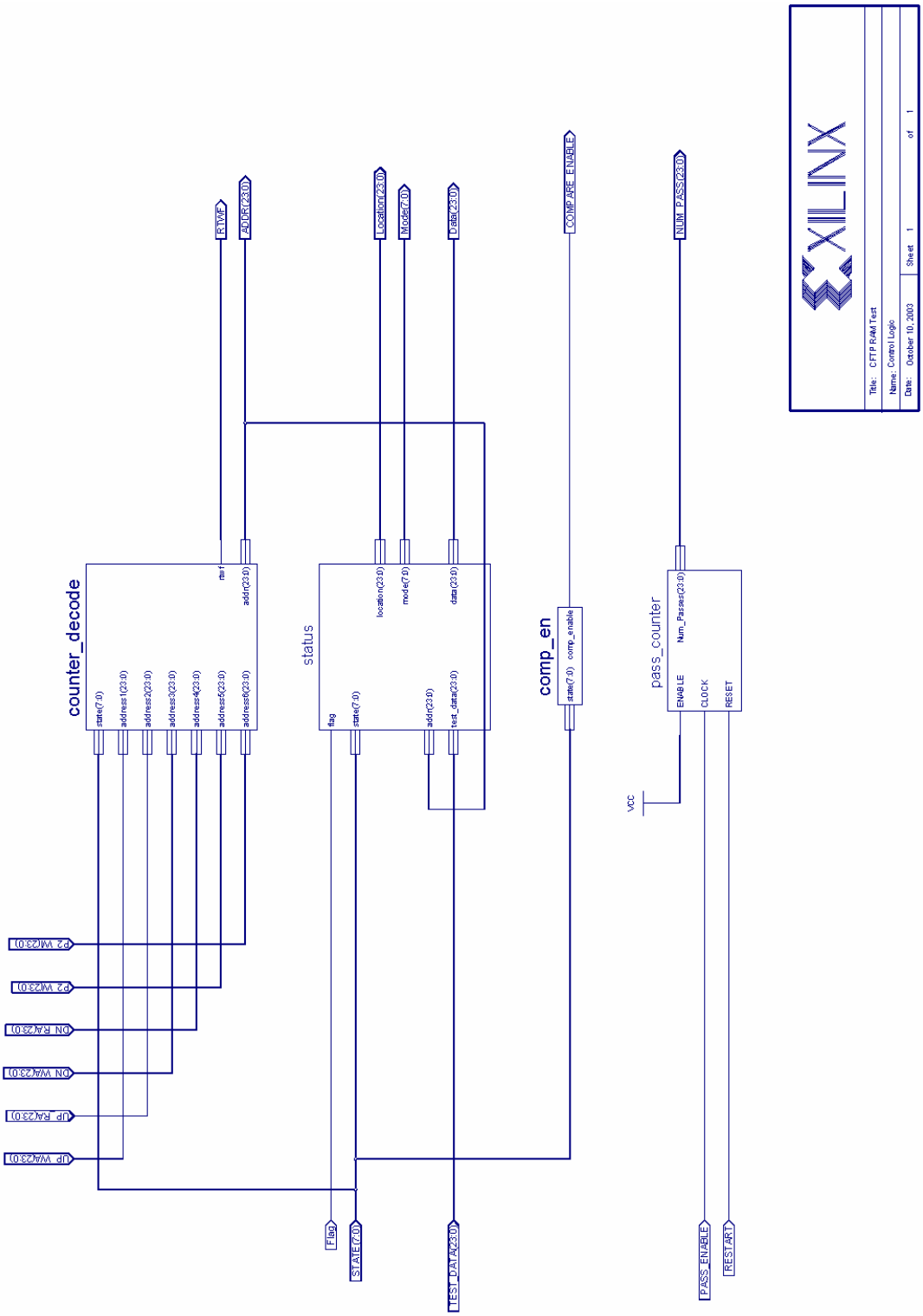
J. STATUS MODULE

1. VHDL Code

```
----------------------------------------------------------------------
-- filename: lat_stat.vhd
-- written by: Charles Hulme
--
-- If a test fails, the current state and memory location are latched
--
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lat_stat is
    Port (
                  state: in std_logic_vector (7 downto 0);
                  flag: in std_logic;
                  addr: in std_logic_vector (23 downto 0);
                  test_data: in std_logic_vector (23 downto 0);
                  location: out STD_LOGIC_vector (23 downto 0);
                  data: out STD_LOGIC_vector (23 downto 0);
                  mode: out STD_LOGIC_vector (7 downto 0)
     );
end lat_stat;

architecture Behavioral of lat_stat is
signal state_latch: std_logic_vector (7 downto 0);
signal addr_latch: std_logic_vector (23 downto 0);
signal data_latch: std_logic_vector (23 downto 0);

begin

fcn: process (flag,addr,state,test_data)

begin
state_latch <= state;
addr_latch <= addr;
data_latch <= test_data;

      if flag = '1' then
            location<=addr_latch;
            data<=data_latch;
            mode<=state_latch;
      else
            null;
      end if;

end process;

end Behavioral;
```

K. TOP LEVEL CONTROL LOGIC MODULE

1. Schematic Diagram



APPENDIX B: COMPLETE SCHEMATICS, VHDL CODES AND TEST-BENCH WAVEFORMS FOR EPROM/PROM TEST

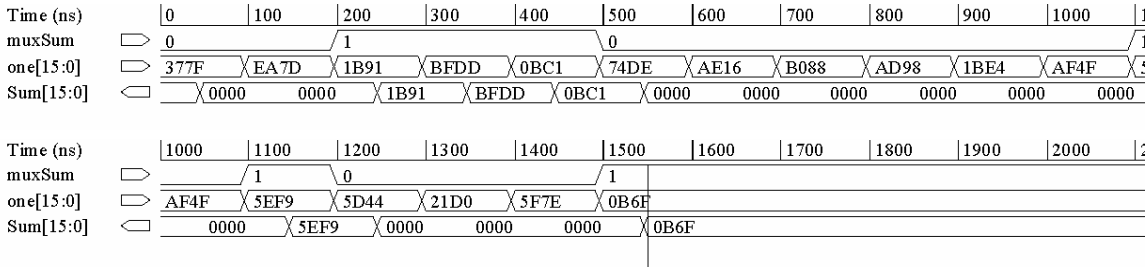
Appendix B contains the schematic diagrams, VHDL code, and Test-Bench waveforms for the complete EPROM/PROM test design. The appendix is organized by module, so each section contains a module schematic/VHDL code and a Test-Bench waveform.

A. MULTIPLEXER MODULE

1. VHDL Code

```
----------------------------------------------------------------------
-- filename: multiplexer.vhd
-- written by: Charles Hulme
--
--  This program simulates a multiplexer and outputs all zeros if
--  muxSum = 0 or the input value of Sum if muxSum = 1
--
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity multiplexer is
    Port (muxSum : in std_logic;
            one : in std_logic_vector (15 downto 0);
            Sum : out std_logic_vector (15 downto 0)
    );
end multiplexer;

architecture Behavioral of multiplexer is

begin

process (muxSum, one)
begin
      if muxSum = '0' then
            Sum <= "0000000000000000";
      else
            Sum <= one;
      end if;
end process;

end Behavioral;
```

2. Test-Bench Waveform

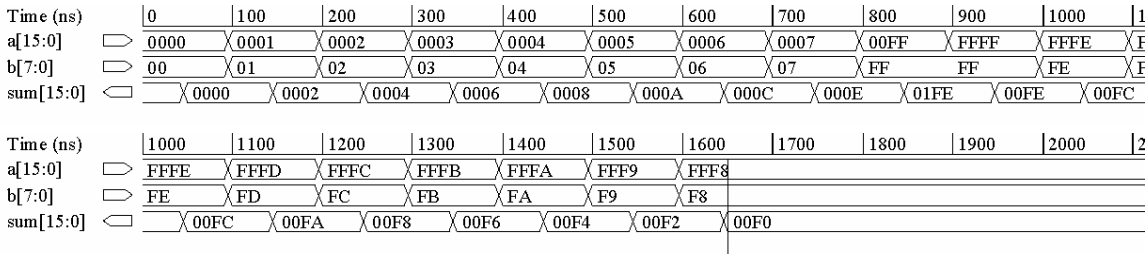


B. ADDER MODULE

1. VHDL Code

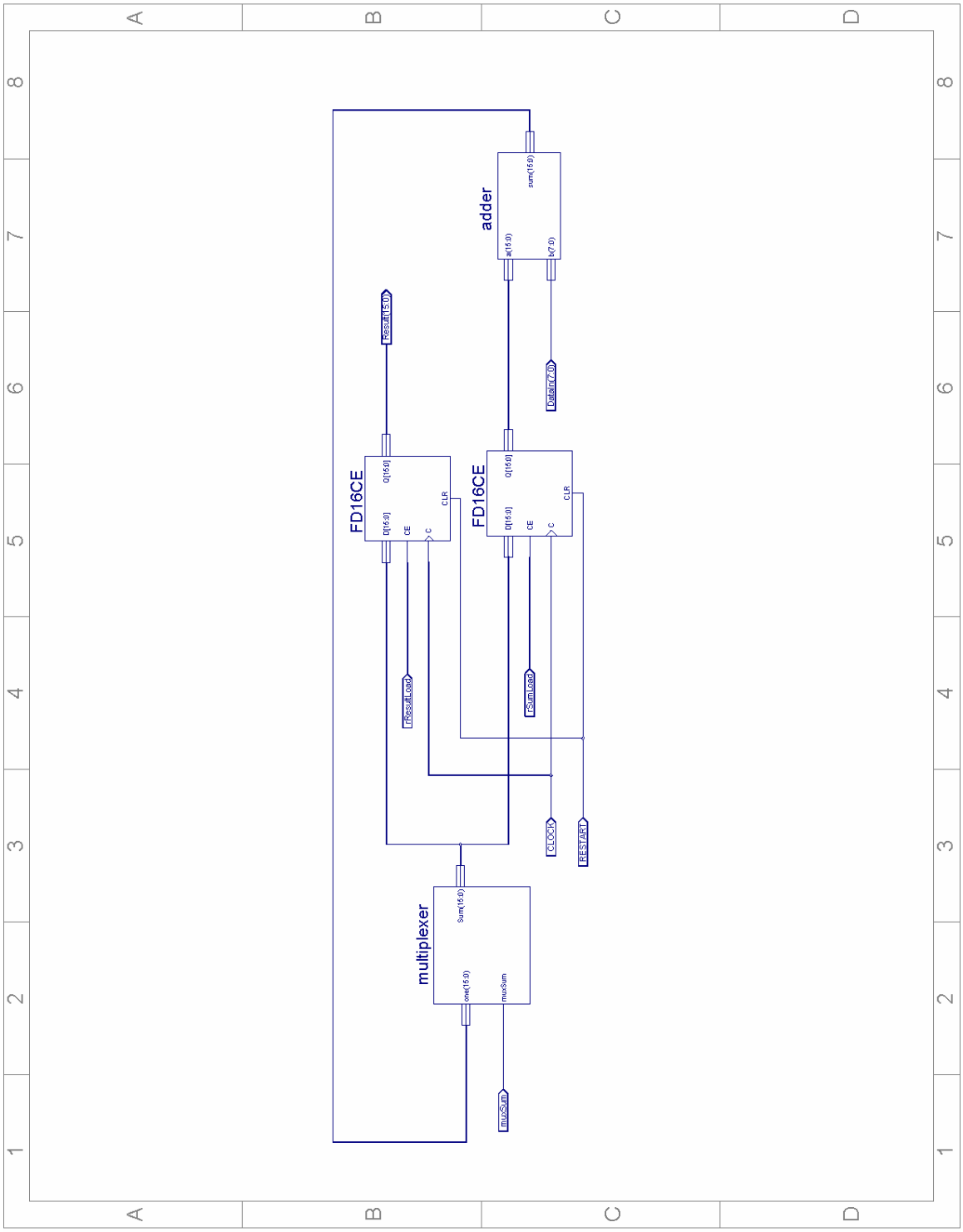
```
----------------------------------------------------------
-- filename: adder.vhd
-- written by: Charles Hulme
--
--  This program adds an 8 bit number with a 16 bit number
--  by concatinating 8 zeros on the front of the 8 bit
--  number.
--
----------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity adder is
    Port (  a: in std_logic_vector (15 downto 0);
             b: in std_logic_vector (7 downto 0);
            sum: out std_logic_vector (15 downto 0)
            );
end adder;

architecture Behavioral of adder is

begin

sum <= a + ("00000000"&b);

end Behavioral;
```

2. Test-Bench Waveform

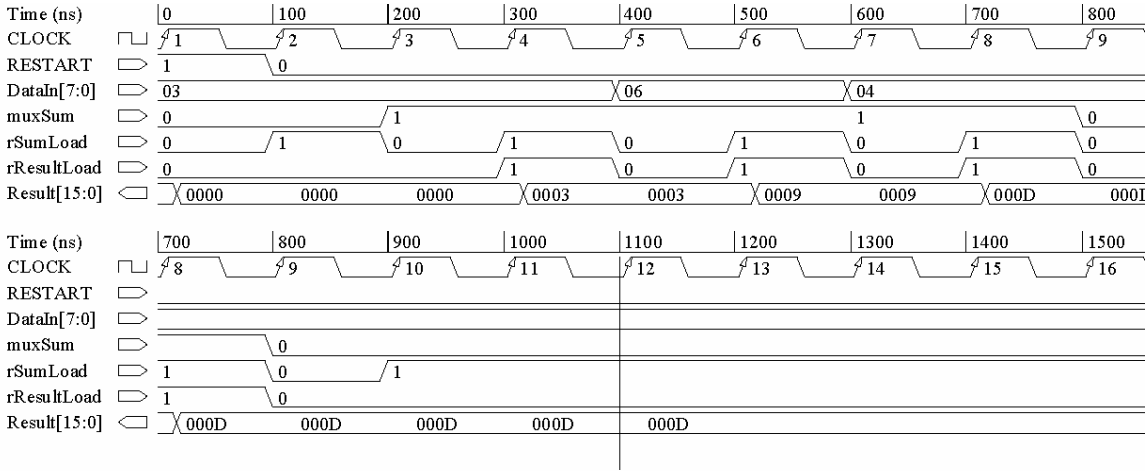


C. DATA MODULE

1. Schematic Diagram



2. Test-Bench Waveform



D. CONTROL MODULE

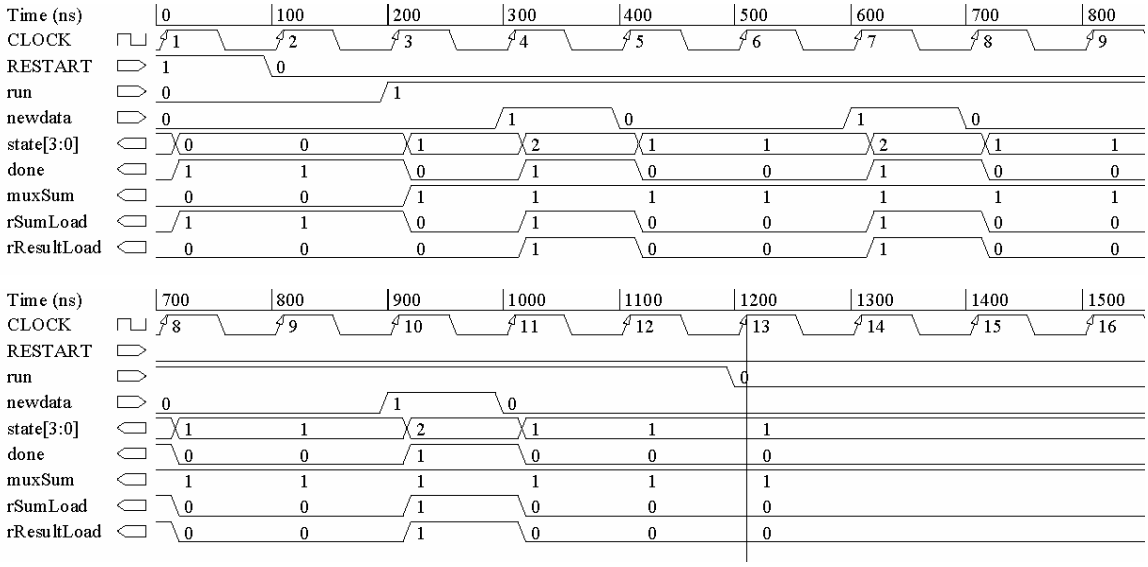
1. VHDL Code

```
------------------------------------------------------------
-- filename: control.vhd
-- written by: Charles Hulme
--
--  This state machine controls the transition of states
--  based on the inputs run and newdata.
--
------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control is
   Port ( CLOCK : in std_logic;
                  RESTART : in std_logic;
                  run : in std_logic;
          newdata : in std_logic;
          state : out std_logic_vector(3 downto 0);
                  done : out std_logic;
          muxSum : out std_logic;
          rSumLoad : out std_logic;
          rResultLoad : out std_logic);
end control;

architecture Behavioral of control is

type FSM_type is (state0,state1,state2);

signal Curr_State, Next_State : FSM_Type;

begin

-- Process that implements the Next State Logic
nxtStProc: process(Curr_State,RESTART,run,newdata)

  begin

   if RESTART = '1' then
     Next_State <= state0;
   else
     case Curr_State is
```

```
        when state0 =>
             if run = '1' then
          Next_State <= state1;
             elsif run = '0' then
               Next_State <= state0;
                    end if;

        when state1 =>
             if newdata = '1' then
          Next_State <= state2;
             elsif newdata = '0' then
               Next_State <= state1;
        elsif run = '0' then
               Next_state <= state0;
             end if;

        when state2 =>
             if run = '1' then
          Next_State <= state1;
             elsif run = '0' then
               Next_state <= state0;
                    end if;

           end case;

end if;

end process nxtStProc;

  --Process to register current state
  curStProc: process (CLOCK,RESTART)
  begin
    if (RESTART = '1') then
       Curr_State <= state0;
    elsif (CLOCK'event and CLOCK ='1')then
       Curr_State <= Next_State;
    end if;
  end process curStProc;

  --Process to generate outputs
  outConProc: process(Curr_State)
  begin
```

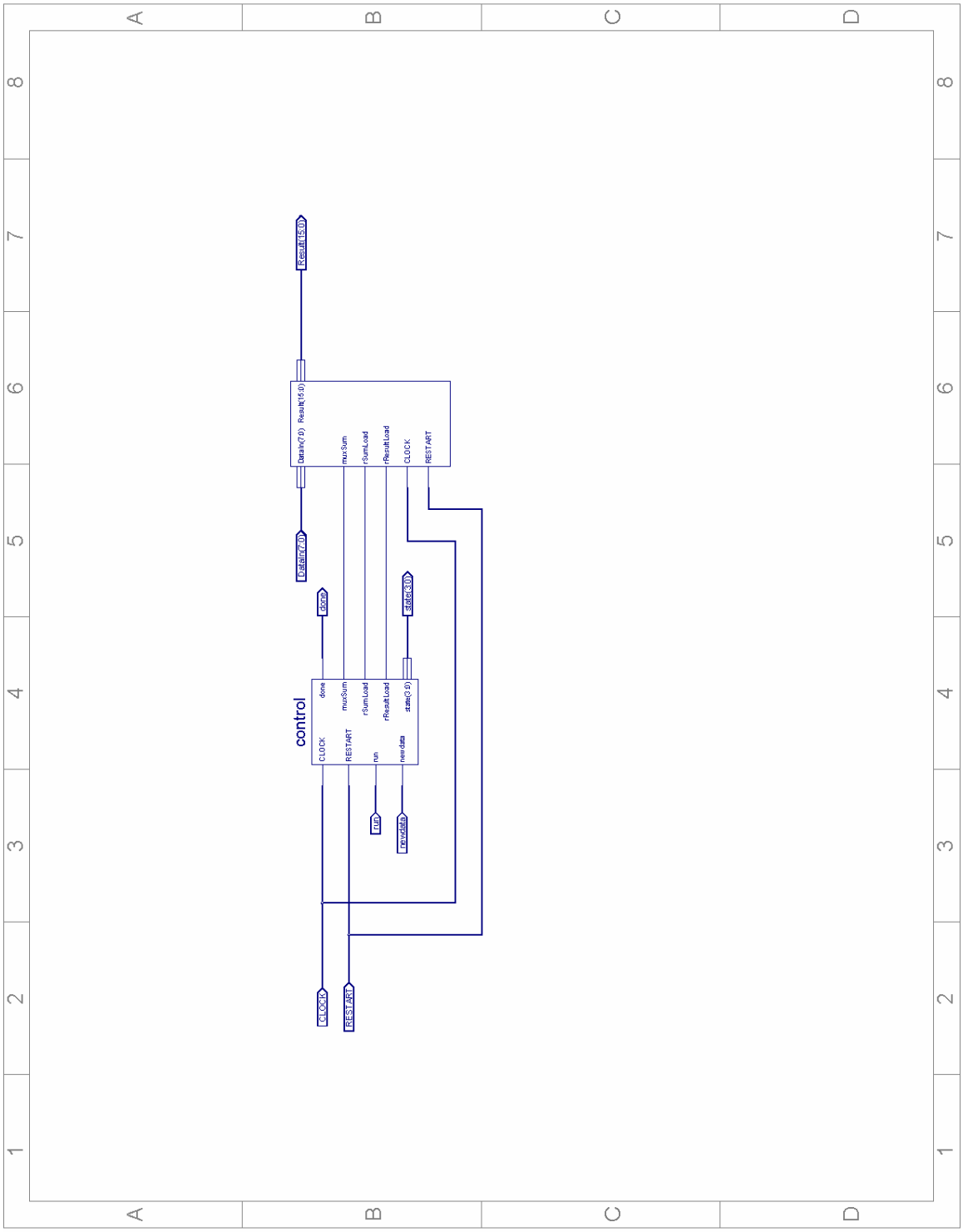
```
    -- Set certain outputs depending on which state currently in
    case Curr_State is

      when state0 =>
            done <= '1';
            muxSum <= '0';
            rSumLoad <= '1';
            rResultLoad <= '0';
            state <= "0000";

      when state1 =>
            done <= '0';
            muxSum <= '1';
            rSumLoad <= '0';
            rResultLoad <= '0';
            state <= "0001";

      when state2 =>
            done <= '1';
            muxSum <= '1';
            rSumLoad <= '1';
            rResultLoad <= '1';
            state <= "0010";

    end case;

   end process outConProc;

end Behavioral;
```

2. Test-Bench Waveform

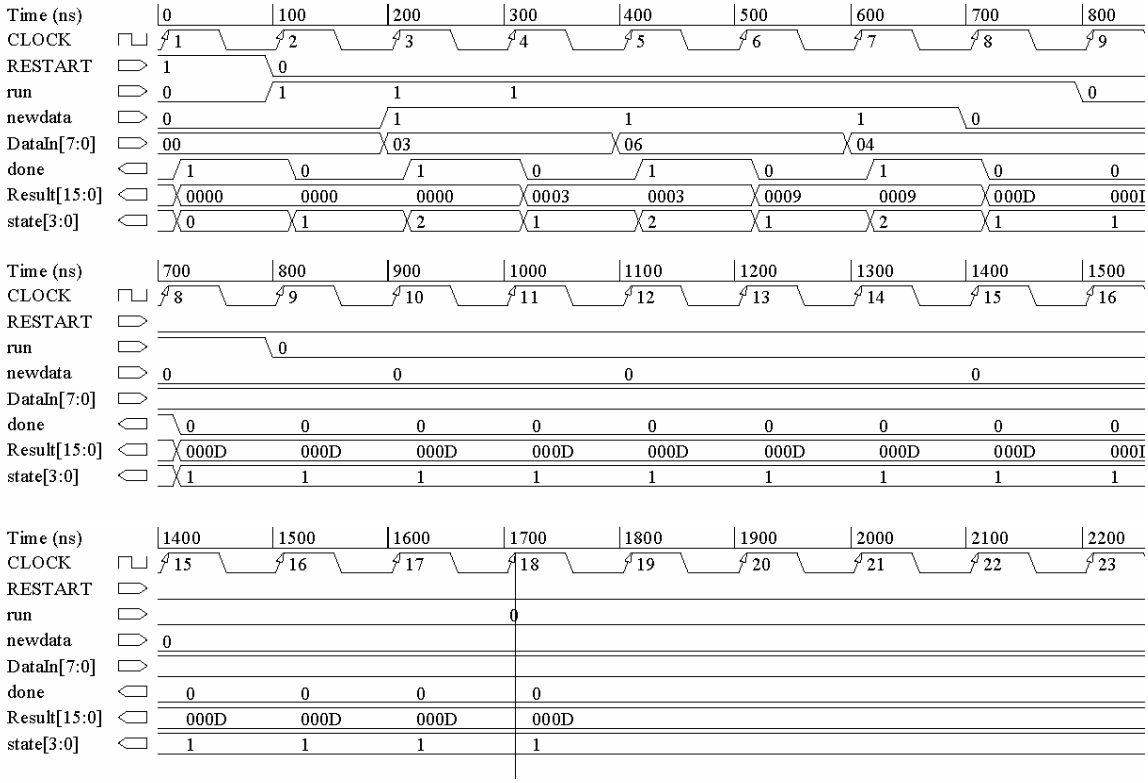


E. SYSTEM MODULE

1. Schematic Diagram

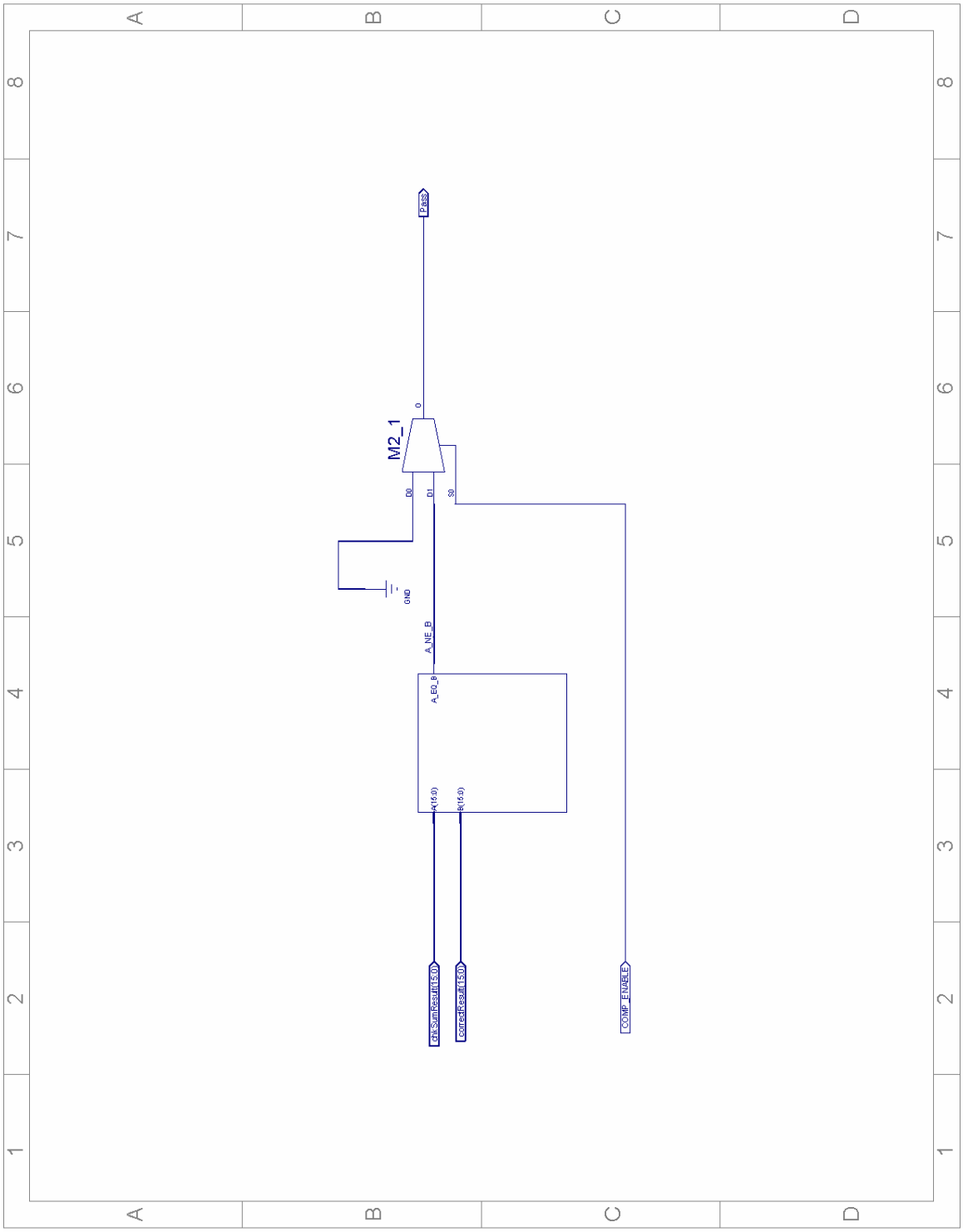


2. Test-Bench Waveform

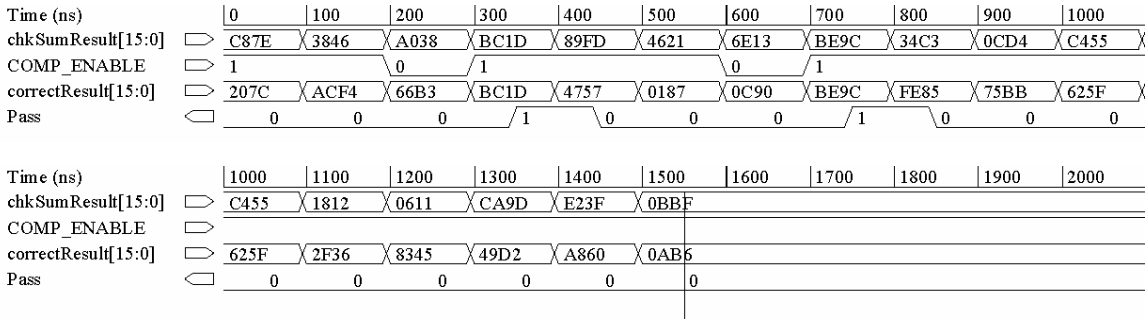


F. COMPARATOR MODULE

1. Schematic Diagram

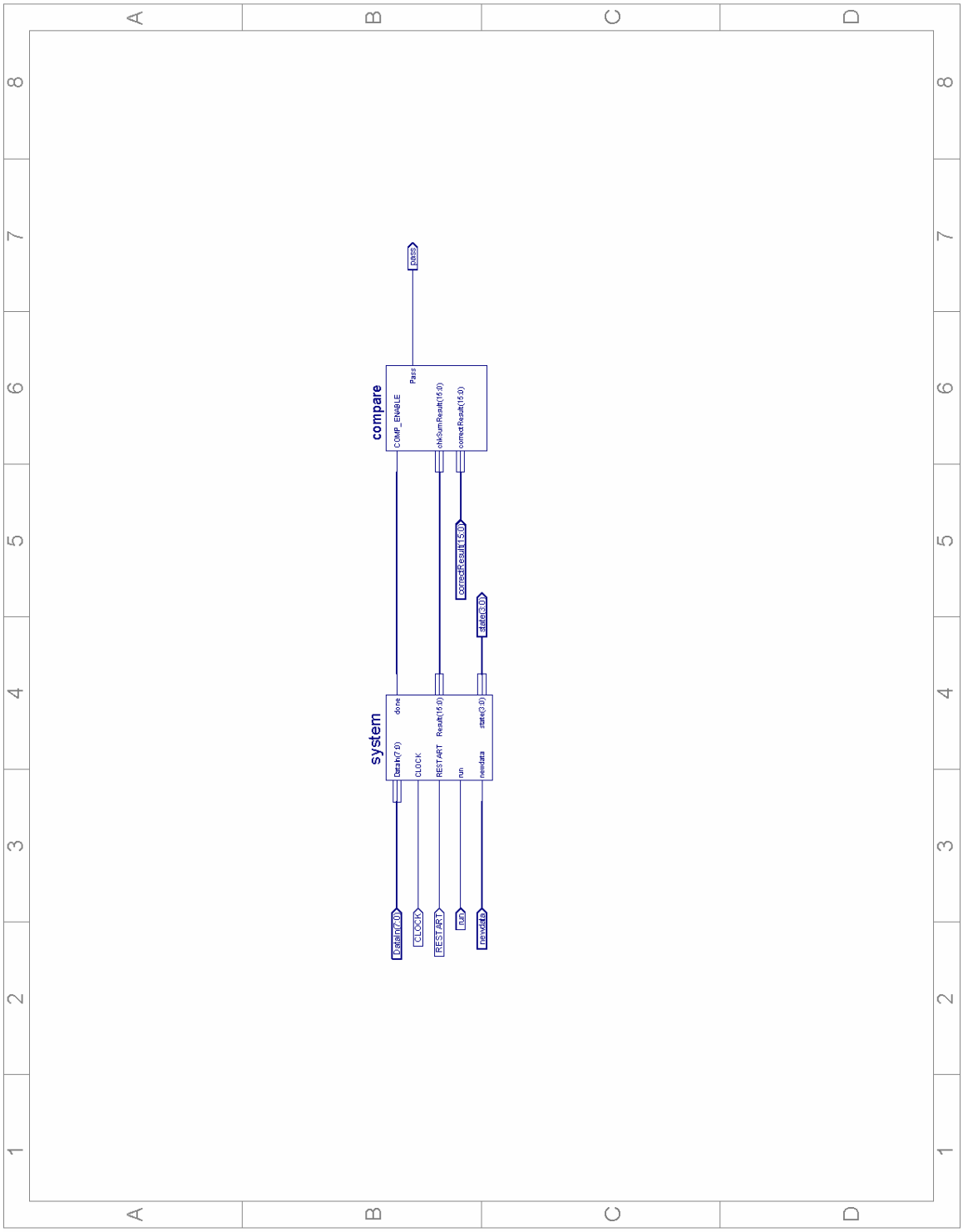


2. Test-Bench Waveform

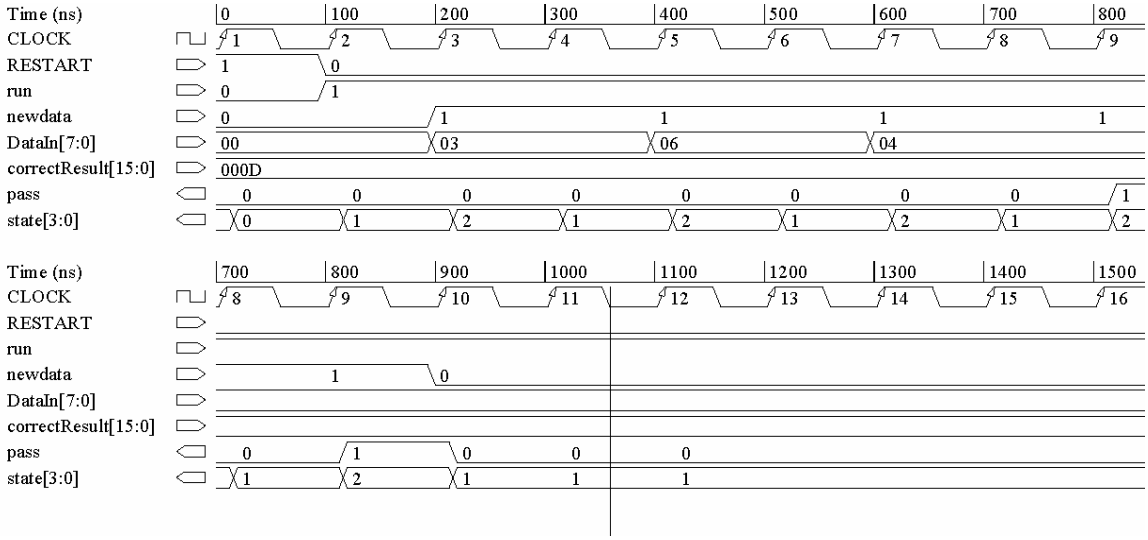


G. TOP LEVEL MODULE

1. Schematic Diagram



2. Test-Bench Waveform



THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Bursch, Daniel W., Notes for SS3011 (Space Technology and Applications), Naval Postgraduate School, 2003 (unpublished).

2. Lashomb, Peter A., "Triple Modular Redundant (TMR) Microprocessor System for Field-Programmable Gate Array (FPGA) Implementation," Master's Thesis, Naval Postgraduate School, Monterey, California, March 2002.

3. Butler, Jon T., Notes for EC4810 (Fault Tolerant Computing), Naval Postgraduate School, 2003 (unpublished).

4. Ebert, Dean, "Design and Development of a Configurable Fault Tolerant Processor (CFTP) for Space Applications," Master's Thesis, Naval Postgraduate School, Monterey, California, June 2003.

5. Configurable Fault Tolerant Processor Space Test Program Application for Spaceflight, DD FORM 1721, July 2003.

6. Configurable Fault Tolerant Processor Space Test Program Application for Spaceflight, DD FORM 1721, August 2002.

7. Saini, Milan, "Platform FPGAs Take on ASIC SOCs," *Xcell Journal Online*, Issue 42, March 1, 2003, http://www.xilinx.com/publications/products/v2pro/xc_pdf/xc_socs.pdf, December 2003.

8. Davis, Shelly, "The Virtex Family - a Powerful ASIC Alternative," *Xcell Journal Online*, Issue 33, December 10, 1999, http://www.xilinx.com/xcell/xl33/xl33_35.pdf, December 2003.

9. "Orbital Express EC3230 Project Brief," Presented by Captain Charles Hulme, USMC, and 1stLt Rong Yuan, TWAF, March 2003.

10. "Orbital Express SERB Brief," Presented by Major James Shoemaker, USAF, Ph.D., November 2002.

11. Payne, John C., "Fault Tolerant Computing Testbed: A Tool for the Analysis of Hardware and Software Fault Handling Techniques", Master's Thesis, Naval Postgraduate School, Monterey, California, December 1998.

12. Summers, David, "Implementation of a Fault Tolerant Computing Testbed: A tool for the Analysis of Hardware and Software Fault Handling Techniques," Master's Thesis, Naval Postgraduate School, Monterey, California, June 2000.

13. Groening, S. E. and Whitehouse, K.D., "Application of Fault-Tolerant Computing for Spacecraft Using Commercial-Off-The-Shelf Microprocessors," Master's Thesis, Naval Postgraduate School, Monterey, California, June 2000.

14. Hofheinz D., "Completion and Testing of a TMR Computing Test Bed and Recommendations for a Flight-Ready Follow-On Design," Master's Thesis, Naval Postgraduate School, Monterey, California, December 2000.

15. Johnson, Steven, "Implementation of a Configurable Fault Tolerant Processor (CFTP)," Master's Thesis, Naval Postgraduate School, Monterey, California, March 2003.

16. Clark, Kenneth A., "The Effect of Single Event Transients on Complex Digital Systems," Doctoral Dissertation, Naval Postgraduate School, Monterey, California, June 2002.

17. "CFTP For Space Based Applications Small Satellite Conference Brief," Presented by Captain Charles Hulme, USMC, August 2003.

18. "Xilinx QPro Virtex 2.5V Radiation Hardened FPGAs," Xilinx Data sheet DS028, San Jose, California, November 2001.

19. "Xilinx Packaging and Thermal Characteristics: Thermally Enhanced Packaging," http://www.xilinx.com/publications/products/packaging/enhanced.htm, October 2003.

20. "Intel Advanced Boot Block Flash Memory (C3)," Intel Data sheet 290645-016, Santa Clara, California, May 2003.

22. "XC18V00 Series of In-System Programmable Configuration PROMS," Xilinx Data Sheet DS026 (v4.0), San Jose, California, June 2003.

23. "QPro Series Configuration PROMs (XQ) including Radiation-Hardened Series (XQR)," Xilinx Data Sheet DS062 (v3.1), San Jose, California, November 2001.

21. "Virtex 2.5V Field Programmable Gate Arrays," Xilinx Data sheet DS003-1, San Jose, California, April 2001.

24. "Elpida HM5225165B-75/A6/B6 HM5225805B-75/A6/B6 HM5225405B-75/A6/B6 256M LVTTL interface SDRAM" Data Sheet E0082H10 (1st edition), Tokyo, Japan, January 2001.

25. Email from Lt. Richard Caldwell, USAF of the DoD Space Test Program to Captain Charles Hulme, USMC, of Naval Postgraduate School, 5 December 2002.

26. Stroud, Charles E., *A Designer's Guide to Built-In Self-Test*, Kluwer Academic Publishers, Massachusetts, 2002.

27. MIL-STD-1540C, *Test Requirements for Launch, Upper-Stage, and Space Vehicles*, 15 September 1994.

28. Barr, Michael, *Programming Embedded Systems in C and C++*, First Edition, O'Reilly and Associates, Inc., 1999.

29. M.D. Ercegovac, *Introduction to Digital Systems*, John Wiley & Sons, 1999.

30. M.D. Ercegovac, *Digital Systems and Hardware/Firmware Algorithms*, John Wiley & Sons, 1985.

31. "XC17V00 Series Configuration PROMS," Xilinx Data Sheet DS073 (v1.1), San Jose, California, April 2002.

32. Wakerly, J. F., *Digital Design, Principles and Practice*, Third Edition, Prentice Hall, New Jersey, 2001.

33. "Configuration and Readback of Virtex FPGAs using (JTAG) Boundary Scan," Xilinx Data Sheet DS139 (v1.4), San Jose, California, April 2002.

34. "IEEE Standard Access Test Port and Boundary-Scan Architecture," Institute of Electrical and Electronics Engineers, New York, New York, July 2001.

35. C. Jordan and W. P. Marnane, "Incoming Inspection of FPGAs," Proc. European Test Conf., pp. 371–377, April 1993.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Director, Training and Education
   MCCDC, Code C46
   Quantico, Virginia

5. Director, Marine Corps Research Center,
   MCCDC, Code C40RC
   Quantico, Virginia

6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California

7. Professor Herschel H. Loomis
   Naval Postgraduate School
   Monterey, California

8. Professor Alan A. Ross
   Naval Postgraduate School
   Monterey, California

9. LCDR Joe Reason, USN
   National Reconnaissance Office
   Chantilly, Virginia

10. CPT Brian Bailey, USAF
    National Reconnaissance Office
    Chantilly, Virginia

11. Captain Charles Hulme, USMC
    United States Naval Academy
    Annapolis, Maryland