NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DESIGN, IMPLEMENTATION, AND TESTING OF A SOFTWARE INTERFACE BETWEEN THE AN/SPS–65(V)1 RADAR AND THE SRC–6E RECONFIGURABLE COMPUTER

by

Thomas G. Guthrie

March 2005

Thesis Advisor:	Douglas J. Fouts
Second Reader:	Jon T. Butler

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** March 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Design, Implementation and Testing of a Software Interface between the AN/SPS–65(V)1 Radar and the SRC–6E Reconfigurable Computer			**5. FUNDING NUMBERS**
6. AUTHOR(S) Capt Thomas G. Guthrie			
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** NSA, 9800 Savage Rd, Fort George C. Mead, MD 20755
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)

This thesis outlines the development, programming, and testing a logical interface between a radar system, the AN/SPS–65(V)1, and a general-purpose reconfigurable computing platform, the SRC Computer, Inc. model, the SRC–6E. To confirm the proper operation of the interface and associated subcomponents, software was developed to perform basic radar signal processing. The interface, as proven by the signal processing results, accurately reflects radar imagery generated by the radar system when compared to maps of the surrounding area. The research accomplished here will allow follow on research to evaluate the potential benefits reconfigurable computing platforms offer for radar signal processing.

14. SUBJECT TERMS C programming, Field programmable logic device (FPGA), hardware description language (HDL), programmable logic device (PLD), radar, reconfigurable computing, signal processing, VHDL			**15. NUMBER OF PAGES** 83
			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

DESIGN, IMPLEMENTATION, AND TESTING OF A SOFTWARE INTERFACE BETWEEN THE AN/SPS–65(V)1 RADAR AND THE SRC–6E RECONFIGURABLE COMPUTER

Thomas G. Guthrie
Captain, United States Marine Corps
B.S., University of Washington, 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2005

Author: Thomas G. Guthrie

Approved by: Douglas J. Fouts
Thesis Advisor

Jon T. Butler
Second Reader

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

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis outlines the development, programming, and testing a logical interface between a radar system, the AN/SPS–65(V)1, and a general-purpose reconfigurable computing platform, the SRC Computer, Inc. model, the SRC–6E. To confirm the proper operation of the interface and associated subcomponents, software was developed to perform basic radar signal processing. The interface, as proven by the signal processing results, accurately reflects radar imagery generated by the radar system when compared to maps of the surrounding area. The research accomplished here will allow follow-on research to evaluate the potential benefits reconfigurable computing platforms offer for radar signal processing.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to acknowledge the efforts of various members of SRC Computers, Inc. for their help with aspects of the coding and design of the work that went into this thesis.

I would like to thank Tim King who served as a valuable sounding board while running with me through the flats and hills of Monterey.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

This thesis is a part of a larger project, which is to compare radar signal processing on a general-purpose computer with processing on more traditional radar systems. If the processing speed is significantly greater on a reconfigurable computing system, this could have significant military implications. Namely, it might be possible to replace several, purpose-built radar processing systems with one general-purpose radar processing system thereby saving in space and weight on military platforms.

The reconfigurable computer used contains field programmable gate arrays (FPGA), or Programmable Logic Device, embedded in the system architecture. FPGAs are devices which can be programmed to implement a wide variety of logic circuits. FPGAs exhibit the benefits of customized hardware processing speeds coupled with the ability to be reconfigured for other applications in a manner similar to loading a new program. Given this speed and flexibility, general-purpose reconfigurable computers marry standard general-purpose computer systems with FPGAs. The intent of these systems is to harness the benefits of FPGAs in standard computing platforms. FPGAs have been shown to provide significant performance speedups in signal processing tasks over traditional methods [1].

With the potential benefits of the FPGA-based processing in mind, the overall project goal was to:

1. Develop, build, and test a physical interface required to pass data from a radar system to a general-purpose reconfigurable computing device.

2. Develop, configure, and test the logical implementation required on the reconfigurable computing device to connect to the interface. This step also includes sampling and storing the data transferred across the interface.

3. Process the radar signal data for display and compare processing speeds to traditional methods.

This thesis developed software to provide a reconfigurable computing system with data from a radar system. Also, limited radar signal processing on that data to proved the proper operation of the interface. The thesis work culminated in a successful

test of the processed radar image of the Monterey Bay area. When compared to cartographic information of Monterey Bay, the processed signal strongly correlated to map data.

In addition to the successful testing of the logical interface, this thesis explored a number of processing methodologies and coding complexities encountered with the reconfigurable computing platform. These lessons learned and the suggested future work should help streamline future work within this project.

I. INTRODUCTION

A. PROJECT OBJECTIVE

This thesis is a part of a larger project, which is to compare radar signal processing on a general-purpose computer with processing on more traditional radar systems. If the processing speed is significantly greater on a reconfigurable computing system, this could have significant military implications. Namely, it might be possible to replace several, purpose-built radar processing systems with one general-purpose radar processing system thereby saving in space and weight on military platforms.

The reconfigurable computer used contains field programmable gate arrays (FPGA), or Programmable Logic Device, embedded in the system architecture. FPGAs are hardware-based devices which can be programmed to implement a wide variety of logic circuits. FPGAs exhibit the benefits of customized hardware processing speeds coupled with the ability to be reconfigured for other applications in a manner similar to loading a new program. Given this speed and flexibility, general-purpose reconfigurable computers marry standard general-purpose computer systems with FPGAs. The intent is to harness the FPGAs benefits in standard computing platforms. FPGAs have been shown to provide significant performance speedups in signal processing tasks over traditional methods [1].

With the potential benefits of the FPGA based processing in mind, the overall project goals were to:

1. Develop, build, and test a physical interface required to pass data from a radar system to a general-purpose reconfigurable computing device.

2. Develop, configure, and test the logical implementation required on the reconfigurable computing device to connect to the interface. This step also includes sampling and storing the data transferred across the interface.

3. Process the radar signal data for display and compare processing speeds to traditional methods.

B. THESIS OBJECTIVE

The objective of this thesis was to develop, implement and test the software interface between a radar system and a reconfigurable computing system. To test the interface it was also necessary to perform limited radar signal processing, specifically building a range map from returned radar signals. The objective of this thesis fulfills the overall project goals two and a portion of goal three. Project goal one was accomplished by the Master's work described in [2].

C. THESIS ORGANIZATION

In order to accomplish the thesis goals, it was necessary to explore the radar system, the AN/SPS–65(V)1, and the interface used to connect to the reconfigurable computing platform. This is described in Chapter II. Chapter III describes the hardware and software environment of the reconfigurable platform used, the SRC–6E.

Moving from the physical devices used, Chapter IV is a discussion concerning logical interface designed to capture data presented by the physical interface to internal components of the SRC–6E. Within the SRC–6E environment, this logical connection is called a macro.

Chapters V and VI are a treatment on basic radar signal generation via the AN/SPS–65(V)1 and how the signal can be processed within the SRC–6E, respectively. Given this base of knowledge, software to perform radar range image processing was written, tested, and evaluated as detailed in Chapter VII. The concluding remarks, Chapter VIII, highlights objectives accomplished by this thesis, some of the limitations of the system presented, and suggest the potential direction of future research.

The general flow of this document is a description of the system built and configured starting with the radar system used, the physical interface built, and the SRC–6E hardware and software configuration. The next chapter describes the general characteristics of the AN/SPS–65(V)1 and the analog-to-digital (A/D) converters used to connect the SRC–6E to the AN/SPS–65(V)1.

II. THE AN/SPS–65(V)1 RADAR SYSTEM AND AN OVERVIEW OF THE A/D CONVERTER INTERFACE

A. INTRODUCTION

As mentioned in Chapter I, the AN/SPS–65(V)1 was used as the radar signal source. This chapter explores the characteristics of the AN/SPS–65(V)1 necessary for understanding subsequent elements of this thesis, particularly the radar signal processing choices made. Also, an overview of the A/D converter functionality and interface signals provided to the SRC–6E are outlined in this chapter.

B. AN/SPS–65(V)1 RADAR SYSTEM FUNDAMENTALS

1. Radar Purpose and Basic Functionality

The AN/SPS–65(V)1 was designed to provide radar detection of moving, low flying targets via Doppler processing [3]. Doppler processing requires an analysis of the phase component of a returned radar signal.

The phase shift of the returned signal, in reference to the transmitted pulse, can be used to determine the speed and direction of a target [4]. The time which the signal returns with respect to the transmission of a radar pulse is used to determine the range of the target. The phase and distance information is represented by two signals within the AN/SPS–65(V)1 on the I and Q channels. Each of these signals is imposed on a 30-MHz carrier wave [3]. Also, a reference signal allows the radar system to resolve the approximate direction of the target in reference to the antenna position.

2. Radar Signals Provided to the A/D Converters

The following signals were provided to the A/D converters, with one exception as noted below.

a. I and Q Signals

As outlined above, the I and Q signals provide the phase shift and range information of a target. The range of a target can be extracted solely from the information presented in either channel.

b. Pulse Repetition Frequency (PRF)

The transmission of a pulse through the antenna occurs at a specific rate. This rate is called PRF and is represented as an electrical signal within the AN/SPS–65(V)1. The PRF signal triggers the process by which the transmission radar pulse occurs. The PRF trigger signal is periodic and can be manually or automatically adjusted during operation of the AN/SPS–65(V)1 [3]. In this analysis, the PRF rate was 3,064 pulses per second.

c. Automatic Gain Control (AGC)

The AGC signal indicates the relative strength of the received signal. The AGC keeps the detected signal within a certain amplitude range to protect the receiving equipment. The relative strength of amplification or attenuation is reported to the radar signal processor via the AGC signal. This signal was provided to the A/D converters and to the SRC–6E. The AGC signal, however, was not used in the processing of the radar signal for this thesis, as the signal strength remained constant at all times during experimentation.

d. Directional Information

The AN/SPS–65(V)1 also provides a reference signal which is used to determine the direction of the antenna. This signal, however, was not used in the design of the A/D converters. As a result, there is no automated manner to determine the relative direction of the antenna at any given time. Given the objective of this thesis, this was not a significant issue, but did limit the automated radar signal processing for this thesis to essentially range information.

C. OVERVIEW OF THE ANALOG-TO-DIGITAL (A/D) CONVERTERS FUNCTIONALITY

As mentioned above, the A/D converters are devices designed and built as the interface between the AN/SPS–65(V)1 and the SRC–6E. The A/D converters provide three basic functions: sample the voltage levels of the analog signals from the radar; convert the analog signals to digital; and present digitally encoded radar signals to the SRC general-purpose I/O port. This port is described in further detail in Chapter III. The following material is an overview of the A/D converters from [2].

1. Operational Speed and Effects

The A/D converters require a clock source to effectively sample the 30-MHz signals coming from the radar system. This is within the capabilities of the A/D converters which were designed to operate at 100 or 200 MHz.

2. Signal Provided to the A/D Converters from the SRC–6E

The A/D converters, however, do not provide an internal clock source. The clock for the A/D converters is sourced from the SRC–6E general-purpose I/O port via operations built into the FPGAs. These operations, or macros, can route and multiply the SRC–6E clock signal. The SRC–6E has an internal clock which operates at 100 MHz. Through multiplication, the clocking signal can easily double to 200 MHz.

3. Signals Provided to the SRC–6E from the A/D Converters

Depending on the mode of operation, the A/D converters provided the following signals to the SRC–6E: I primary, I secondary, Q primary, Q secondary, Data Ready, PRF and AGC.

a. I and Q Primary and Secondary Signals

In the 100-MHz mode, the A/D converters sample the I and Q channels and provide an 8-bit signal for each channel at a sampling rate of 100 MHz. These 8-bit signals are called I primary and Q primary, respectively. When sampling at 200 MHz, the A/D converters represent each channel with two 8-bit signals. While the I and Q channel information is sent to the SRC–6E at 100 MHz, the effective data rate is 200 MHz. The four signals are referred to as I primary, I secondary, Q primary, and Q secondary. The A/D converters sampled the radar signals at 100 MHz for testing throughout this thesis.

b. Data Ready Signal

The Data Ready signal is essentially the clock used by the A/D converters. When the Data Ready signal transitions from low-to-high (voltage), the data presented on the other signals is valid. This signal was used to capture valid data for storage. Data Ready is a 1-bit signal and originates from the A/D converters.

c. PRF

The PRF signal within the A/D converters is a level-shifted version of the PRF signal from the AN/SPS–65(V)1. The AN/SPS–65(V)1 PRF signal was level-shifted and fed directly in the SRC–6E as a 1-bit signal.

d. AGC

The AGC signal is the sampled version of the AGC signal from the radar system. As with the I and Q channels, the AGC signal from the A/D converters are two 8-bit signals, AGC primary and AGC secondary. Both the primary and secondary signals are used if the A/D converters are sampling at 200 MHz and only the primary signal at 100 MHz.

4. Voltage Representation of the Radar Signals

The relative strength of the radar signal return is captured as a voltage level within the AN/SPS–65(V)1. Objects which reflect a greater portion of the radar pulse are represented with a higher voltage level on the I and Q channels of the radar than those with a lesser reflectivity. The A/D converters represent the sampled voltage level with 8-bits. This provides 256 discrete levels which represent the input voltage from the radar.

The most negative value of the input voltage is represented by 0, the voltage maximum by 256, and the zero voltage level by 128. Given a 5-volt peak-to-peak input sine wave, the voltages +2.5 v, 0 v, and –2.5 v would be represented by 255, 128, and 0, respectively, with eight bits. This format for representing data is also known as offset binary.

Once the 8-bit representation has been sampled and stored in the SRC–6E, it must be converted again to reproduce the actual voltage presented to the A/D converters by the AN/SPS–65(V)1. This was done with the following equation where the signal voltage is:

$$s[\mathrm{V}] = a(b - 128). \tag{2.1}$$

The constant a is a simple scaling factor used to reconstruct the actual voltage level at the time of sampling and b is the 8-bit value. Using Eq. (2.1), it possible to recreate both the positive and negative voltage levels sampled from the radar.

The voltage conversion equation above will be used in later chapters as part of the radar signal processing. The signals presented by the AN/SPS–65(V)1 to the A/D con-

verters are also used in the processing of the radar signals in the SRC–6E general-purpose reconfigurable computing platform. An adequate discussion of the image processing must take into account the platform on which the processing took place. An exploration of that platform is the subject of the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

III. THE SRC–6E SYSTEM

A. INTRODUCTION

In this project, the A/D converters and the radar system ultimately connect to the SRC–6E. The sampling, storing, and a portion of the processing of the radar signal will be done within the SRC–6E. The means by which the processing takes place is the subject of following chapters. Those discussions, however, must be based on an understanding of the hardware, software, and programming environment of the SRC–6E. That is the subject of this chapter.

B. SRC HARDWARE ENVIRONMENT

As mentioned in the introduction, the SRC–6E is a general-purpose reconfigurable computing platform. The SRC–6E contains two Intel® processing systems and one Multi-Adaptive Processing (MAP®) board [5] containing four FPGAs.

1. Microprocessor Side of the SRC–6E

The microprocessor side of the SRC–6E is composed of two general-purpose personal computers (PC). Each PC has two Intel Pentium® III processors. We used only one of the general-purpose computers. As with most PCs, there is a memory bus connecting various components within the system. The interconnect between the microprocessor and MAP side of the SRC–6E is done via a SNAP™ card interface that connects the microprocesor memory bus to one half of the MAP board [6]. Memory transfer between the microprocessor side and the MAP side can be accomplished through standard Direct Memory Access (DMA) methods [5] across this interconnect. Memory space on the microprocessor is Common Memory, as programs running on the microprocessor or the MAP can access it [6].

2. MAP Side of the SRC–6E

The MAP board is composed of two MAP processors. Each MAP processor is composed of two FPGAs, associated memory, and the required control circuitry to implement the functionality of the MAP processor.

a. MAP Connectivity

As described above, one of MAP processors (MAP0) is connected to a microprocessor via a SNAP interconnect. The second MAP processor (MAP1) is connected to the second microprocessor system [7]. Individual MAP processors, MAP0 and MAP1 in this example, each have an input and an output chain port. A chain port is a connection used to connect two or more MAPs together in a daisy chain. Through user-developed macros, a chain port can be converted into a general-purpose I/O port [8].

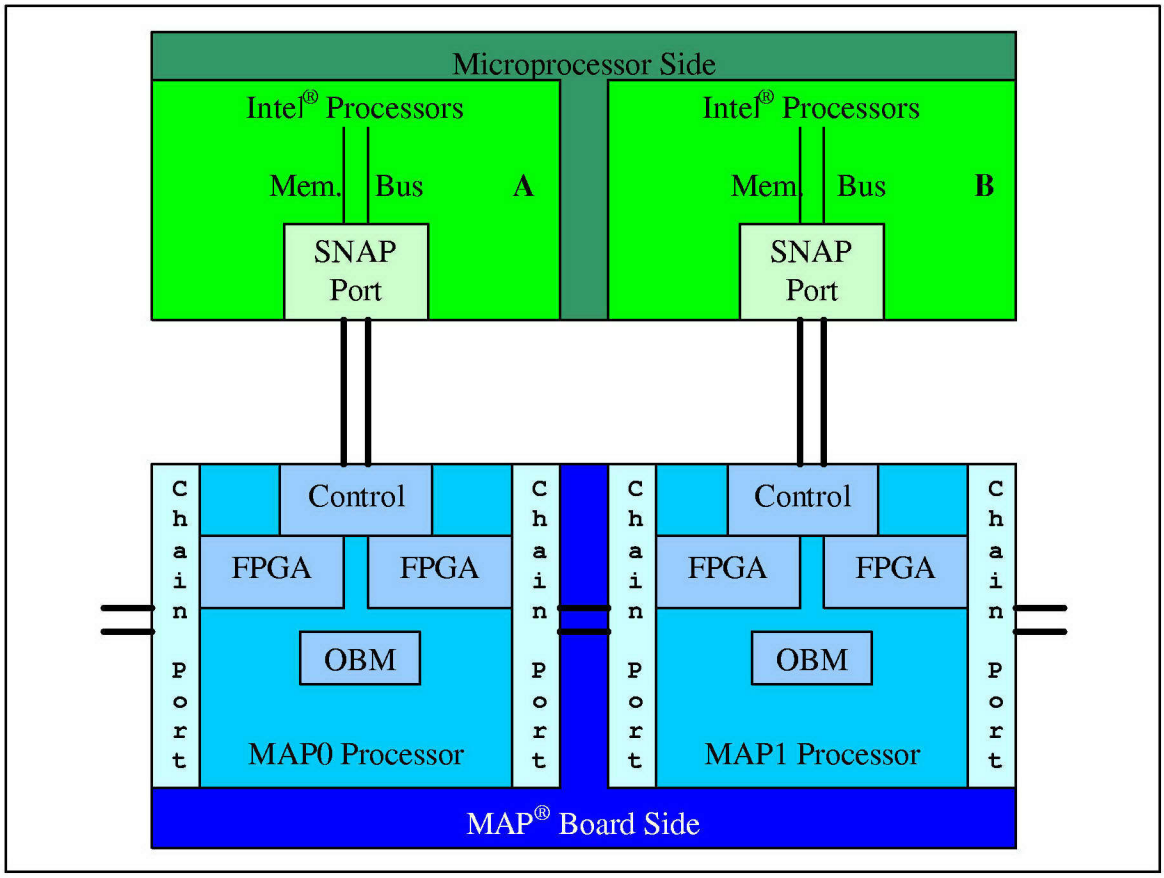

Figure 1. Internal components of the SRC–6E.

While a chain port and a general-purpose I/O port are physically the same device, they are functionally different. The general-purpose I/O port function is used to connect a MAP system to some external device. Chain ports connect MAPs to other MAPs. The port which the A/D converters connect to the SRC–6E is a general-purpose I/O port.

Each general-purpose I/O port has a collection of I/O pads. This is where physical contact between the A/D converters and the SRC–6E takes place. The specfic connection names between these to systems is outlined in Appendix B.

b. MAP Memory

MAP functions (program fragments running on the MAP) can access the memory resident on the MAP. One component on a MAP is the On Board Memory (OBM) as shown in Fig. 1. There is a total of 24 MB of memory on an individual MAP. The 24 MB is divided evenly between six banks, each of 4 MB. A single bank of OBM is conceptually organized as a two-dimensional array where each element is 64 bits wide. The maximum number of elements in a single OBM bank is 523,776. The organization of OBM memory allows for the access of multiple (six) memory locations on each clock cycle. This aspect of memory access within the MAP facilitates parallel processes. [5, 6].

C. SRC–6E SOFTWARE ENVIRONMENT

The SRC–6E software components were designed to abstract the details of the hardware system itself from the general programmer. When writing programs designed to take advantage of the FPGAs, a programmer can code in either FORTRAN or C [6]. The main program should call on functions which have been ported over to the MAP to take advantage of the potential speed-up an FPGA offers. It is up to the individual programmer to determine which aspect of a program would benefit, in terms of speed, by running on an FPGA. The C language was used on the microprocessor and MAP side functions of the SRC–6E throughout this thesis.

A program that is executed on the MAP side is written as a C function and is called a MAP function. The main program, running on the microprocessor side, calls and passes data in a manner similar to that of regular function calls in C.

MAP functions, while written in C, are converted to a Hardware Description Language (HDL) by the SRC–6E compiler. Compiling C to HDL is accomplished through a series of SRC–6E defined macros. These system-defined macros are segments of code that provide the abstraction from system hardware that programmers have come

to expect in modern systems. Programmers can, however, extend the range of functionality provided by system-defined macros by creating user-defined macros. These user-defined macros must be written in an HDL, such as VERILOG or VHDL. While MAP functions cannot call upon other functions, they can explicitly call on user-defined macros.

The user-defined macros developed in Chapter IV were 'written' orginally as circuit diagrams. These circuit diagrams were automatically converted into VHDL and then ported into the SRC–6E programming environment.

This chapter, in summary, outlined the basic hardware and software components of the SRC–6E. Memory, DMA, microprocessor side, MAP side, MAP functions, and macros are all concepts upon which this thesis is built. It is now possible to begin the discussion of the various elements developed to accomplish the thesis objectives. The next chapter is a discussion of the macros developed to interface the logical components of the SRC–6E and A/D converters.

IV. BUILDING A MACRO IN THE SRC–6E ENVIRONMENT

A. INTRODUCTION

As mentioned in Chapter III, a macro within the SRC–6E is a section of code written in a HDL that allows the programmer to define special tasks for MAP functions to call upon. MAP functions cannot call other functions as is possible in a C programming environment. MAP functions call on a collection of system-defined and/or user-defined macros to accomplish special-purpose tasks.

User-defined macros allow the programmer to extend the range of applications for MAP functions from what is provided by the system-defined macros. Programming a user-defined macro requires knowledge of digital logic design, an HDL language (or symbology), and possibly a set of tools to allow for the generation of HDL code from a macro schematic.

This chapter outlines the process taken to create a series of macros that, in turn, allowed for the collection of radar signals for processing in the SRC–6E. To further explain the macros created in this thesis, a discussion of the different types of macros and how a macro is interfaced in the SRC–6E is required.

B. MACROS TYPES AND INTERFACING PROGRAMS AND PROGRAMABLE COMPONENTS IN THE SRC–6E

Within the SRC–6E MAP programming environment, there are five types of macros [6]. Of these five, two were used in this thesis, the purely functional macro and external macro.

1. Purely Functional Macros

Purely functional macros are called and then return a value or values to the calling MAP function. Such macros do not have any state values. These macros can be pipelined such that they can accept new input data while internally processing the results from previous input values [6].

Since the radar processing interface works at the internal clock speed of the SRC–6E MAP, 100 MHz, it is important that any macro developed operate at that clock speed.

A significant feature of purely functional macros is that they can be pipelined and executed in parallel to maximize clocking efficiency with respect to data input.

2. External Macros

External macros interact with parts of the system beyond the code block in which it operates [6]. It was initially thought that these types of macros would be required to interface with the A/D converters to retrieve data presented on the general-purpose I/O port. This turned out to be incorrect. It was, however, not discovered until later in the macro development cycle and is covered in more detail later in this chapter.

External macros require an implicit start and done signal added to the macro code as well as the black box file (see Section 3.a). The start and done signals are system control signals that facilitate the passing of system control to and from an external macro. The start signal is initiated by the SRC to the external macro indicating that the external macro should begin execution. The done signal originates in the macro and indicates to the SRC that it has completed execution.

3. Macro-to-MAP Function Interface

In order to facilitate compilation of MAP function calls of macro code, two user-defined files characterize the nature of the interface. These files are called 'black box' and 'info' within SRC documentation [6].

a. Black Box Files

Black box files are a description of the macro inputs and outputs *from the perspective of the MAP function and* SRC–6E *system-defined signals*. An example of a SRC defined signal would be the clock signal, labeled CLK in Fig. 2. No other functionality of the macro is revealed in the black box file. Figure 2 contains a typical black box schematic of a macro, while Fig. 3 contains the actual coding used to represent it.

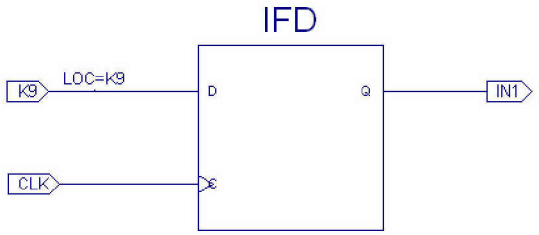

Figure 2. Black box diagram of a macro.

```
module bitin (CLK, IN1);
      input CLK;
      output IN1;
endmodule
```

Figure 3. Black box file written in Verilog HDL.

Notice that the K9 input signal shown in Fig. 2 is not included in the black box file in Fig. 3. As mentioned above, a black box file is written from the perspective of a MAP function and system-defined signals. In Fig. 2, the signal IN1 is returned to the calling MAP function. For this example, assume the input K9 is not a standard system signal, like CLK, and is not provided by a MAP function to the macro. As such, the signal K9 is not written into code in Fig. 3.

If the K9 input was included in the black box file, the SRC–6E compiler would attempt to connect the input to essentially a random MAP function variable defined in the black box file. This system-induced connection then causes ambiguity in the circuit connections and a resulting compiler error.

This SRC-induced nuance is counterintuitive when thinking of black box descriptions in general. Discovering this nuance took several weeks of research and it is mentioned here in hopes that follow-on research will benefit from this discussion.

b. Info Files

Info files establish the mapping of operators and calls in the MAP function to the macro signal names [6]. The info file defines the name of the macro to be called, the type of macro (pure functional, external, or one of the other three types), the macro latency, and other characteristics of the macro.

Figure 4 is the info file code required for the macro diagram shown in Fig. 2. It defines the macro as not stateful and not external. As a result, the macro type defaults to functional. The info file characterizes the macro as taking one clock cycle to execute (latency). Further, the macro can be pipelined. There are no inputs and there is one output which is one bit wide. The macro also uses the standard SRC–6E system signal 'clock.' Once again, the input K9 was not defined in the info file for the same reasons that it was not defined in the black box file.

```
BEGIN_DEF "bitin"

 MACRO = "bitin";
 STATEFUL = NO;
 EXTERNAL = NO;
 PIPELINED = YES;
 LATENCY = 1;

 INPUTS = 0:
 ;

 OUTPUTS = 1:
  O0 = INT 1 BITS (IN1)
 ;

 IN_SIGNAL: 1 BITS "CLK" = "CLOCK";
END_DEF
```

Figure 4. Sample info file.

C. MARCO DEVELOPMENT FOR RADAR IMAGE RETRIEVAL

Initially, a macro was to be written to provide:

- 16 bits of I channel data to the MAP function from the A/D converters.
- 16 bits of Q channel data to the MAP function from the A/D converters.
- 16 bits of AGC data to the MAP function from the A/D converters.
- 1 bit of PRF data to the MAP function from the A/D converters.
- A Data Ready signal to the components within the macro from the A/D converters.
- A doubled SRC–6E system clock signal from the macro to the A/D converters. For various reasons, this was later changed to 100 MHz while maintaining the capability to double the clock speed.
- Synchronization of the clock domains between the SRC–6E and the A/D converters in order to minimize clock skewing problems.

Macro development that would accomplish the above points was decomposed into two subcategories, writing data out of and reading data into the SRC–6E system across a general-purpose I/O port.

All macros were drawn as schematics and then automatically translated by the Xilinx® application Project Navigator, release version 6.2.03i, into VHDL code. The resultant code was then imported as a macro file into the SRC–6E programming environment. For the purposes of this thesis, macros will be represented by schematics for sim-

plicity. As the schematics are logic circuits, the term circuit will be used synonymously with the term macro. Pad names follow the conventions set forth in the connectivity diagrams in Appendix B.

1. Data/Clock Out of the SRC–6E

Figure 5 is the schematic that was used to generate the clock signal out of the SRC–6E at 100 MHz via a Digital Clock Manager (DCM). It was initially thought that, since this macro would interface with elements outside of the code block, the macro must be defined as external. This turned out later to not be the case and subsequent versions of the clock output macro were defined as purely functional.

In Fig. 5, the clock output cannot be fed directly to a Xilinx pad but must first be buffered by an output buffer. For this experiment, an OBUF_F_24 was used which provides a fast slew rate and drive power of 24 mA for a low voltage transistor-transistor logic (LVTTL) pad [9]. The BUFG, Global Clock Buffer [9], provides a tap point for the clock feedback line. The clock feedback circuitry synchronized the clock output of the DCM (CLK0) to the clock input into the DCM (CLK). The START/DONE elements of the circuit served to fulfill the SRC–6E requirements for external macros.

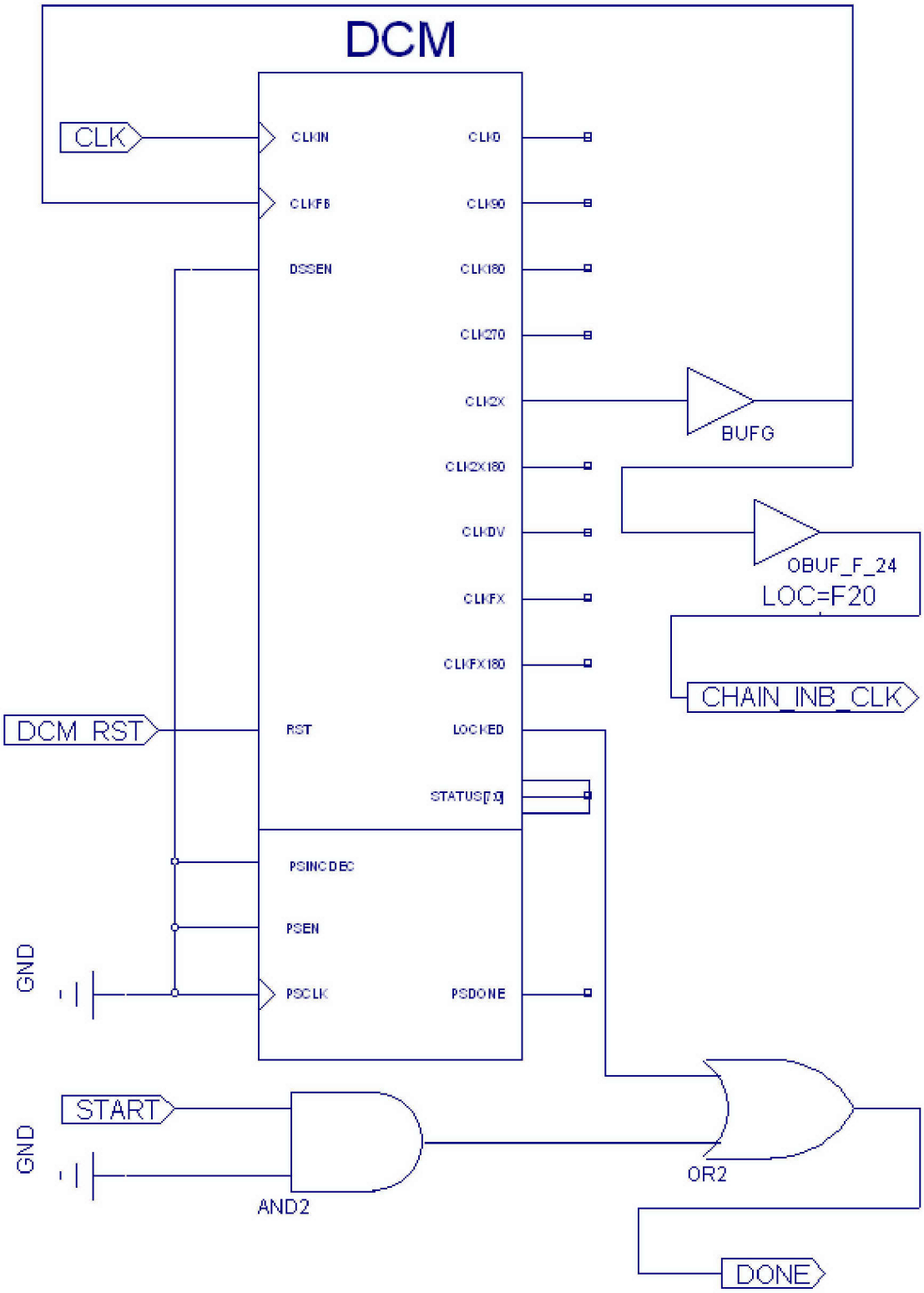

Figure 5. Initial clock output macro/circuit.

Using this macro and a digital oscilloscope, the voltage output from the SRC–6E was recorded as shown in Fig. 6. The clock provided by the SRC–6E is a 100-MHz square wave, but due to imperfect impedance matching and the poor frequency response of the observing equipment, the clock signal appears sinusoidal.

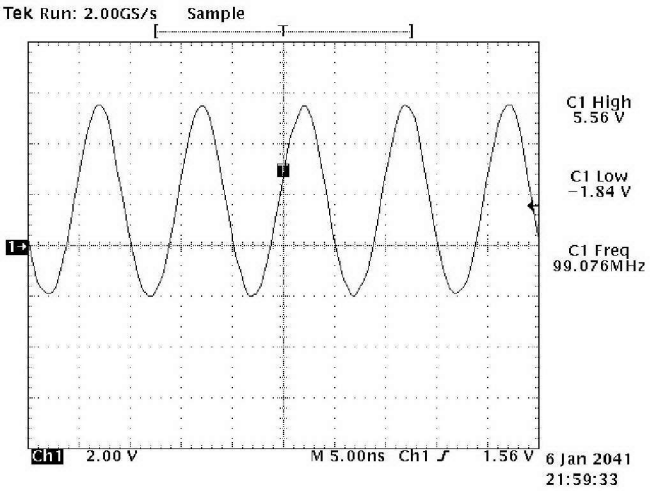

Figure 6. 100-MHz clock output using externally defined macro.

With subsequent experiments, the clock speed could be manipulated to nearly any desired value when using flip flops or counters. The DCM, provided by Xilinx, has a feature called synthetic frequency division which is supposed to divide the frequency of the signal. It was found, however, that synthetic frequency division via the DCM did not work. SRC documentation supported this finding [10]. Whether this is an artifact of the Xilinx implementation of a DCM, an SRC implementation of a Xilinx FPGA, or a combination of the two is unknown to this author.

2. One-Bit Data Input Into the MAP of the SRC–6E

After establishing an output clock, reading data as variables into the SRC–6E remained. While the eventual goal was to read upwards of 25–50 data pins, initial efforts focused on simply reading one bit at a clock rate of 100 MHz and storing that data for analysis.

A purely functional macro schematic was created as shown in Fig. 7. The macro stores the logic value from the pad into an I/O D-flip flop on every positive edge from the SRC–6E system clock. Once stored, the data is presented on the output labeled IN1. Through the interfaces defined in the info and black box files, IN1 was read in as a 32-bit

integer variable, sign extended as a 64-bit integer, and stored as a 64-bit element in an OBM bank on the MAP. Once several thousand samples were collected, the OBM bank contents were transferred via a DMA transaction to the microprocessor side and either displayed on screen or stored as a file for later analysis.

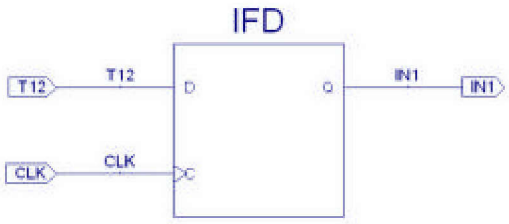

Figure 7. One-bit input macro.

The input pad, T12, was connected to a function generator during the tests. The function generator created a square wave signal operating at 1 kHz between at the ranges of logic values 1 and 0 for LVTTL circuitry with at 50% duty cycle. For a given cycle in the square wave at 1 kHz and a sampling rate of 100 MHz, there were approximately 50 logic 1s and 50 logic 0s in the output file. This indicated that the input signal was sampled and stored at a rate of 100 MHz.

3. Multiple Bit Input into the SRC–6E

Working from the techniques defined in the one-bit input example, a two-bit-in-parallel input macro was created but proved not to function properly. As mentioned earlier, the one-bit input signal was implicitly read as a 32-bit integer before being sign extended into a 64-bit element in an array. The 'expansion' of the one-bit signal to a 64-bit variable was done implicitly by the SRC compiler with system-defined macros. This implicit process did not seem to be supported by the compiler for signal values two or four bits wide. Signal values of eight, 16, and 32 bits were properly converted to 64-bit variables by the compiler.

Eventually, an encompassing macro was developed to read two 8-bit I channel words, two 8-bit Q channel words, a one-bit PRF signal, and a one-bit Data Ready signal. The I and Q channel bits were defined as a single 32-bit integer variable. The least significant 16-bits represented the I channel information while the remaining bits represent the Q channel information as shown in Table 1 below.

Table 1. Data division of input signals.

	Q Channel		I Channel	
Bits represent	Q secondary	Q primary	I Secondary	I primary
Bit positions	31:24	23:16	15:8	7:0

This purely functional macro was tested and operated as expected.

4. Combining Clock Output with Data Input in a Macro

The next step in the macro development process was to combine the externally defined clock output macro with the purely functional data input macro. The resultant circuit would effectively become an externally defined macro. Upon testing this circuit, however, the sampling-and-storage rate dropped to approximately 10 MHz.

Externally defined macros actually operate in a code block that is separate from the MAP function. When the MAP function called the external macro, control of the system was passed to the macro. When the macro finished, control was passed back to the MAP function. This passing of system control incurred a severe performance penalty of 90%. The external definition was removed and the sampling-and-storage rate returned to 100 MHz.

The macro finally developed to read and write data from the A/D converters interface to the SRC–6E is fully described pictorially and in VHDL in Appendix C.

D. TESTING THE RADAR INTERFACE MACRO

While testing of the macros continued throughout the design process, only the final design testing process is discussed here.

1. Test Equipment Setup

The I channel primary connection from the A/D converters were connected to the SRC–6E per the tables in Appendix B. Instead of the AN/SPS–65(V)1 connected to the A/D converter interface, a signal generator emitting a 1-kHz sine wave with a 200-mV peak–to–peak swing was connected. The macro in Appendix C running at a sampling rate of 100 MHz was used to sample the data from the A/D converters.

2. Test Results

Figure 8 depicts the results of the SRC–6E sampling a sine wave input which was saved as a file and then plotted. Figure 8 depicts the first 1000 samples of the 500,000

samples taken and the voltage level recorded for each sample. Over the 500,000 samples there was a 1.90% error rate, where an error is any sampled voltage level outside the expected input voltage range. In this case, the expected voltage range was ± 100 mV . The source of this error was not determined, as it was not significant for the purposes of this thesis. At a sampling rate of 100 MHz, it was expected that a single cycle of a 1-kHz sine wave would take 100 samples. This supposition was supported by the findings shown in Fig. 8.

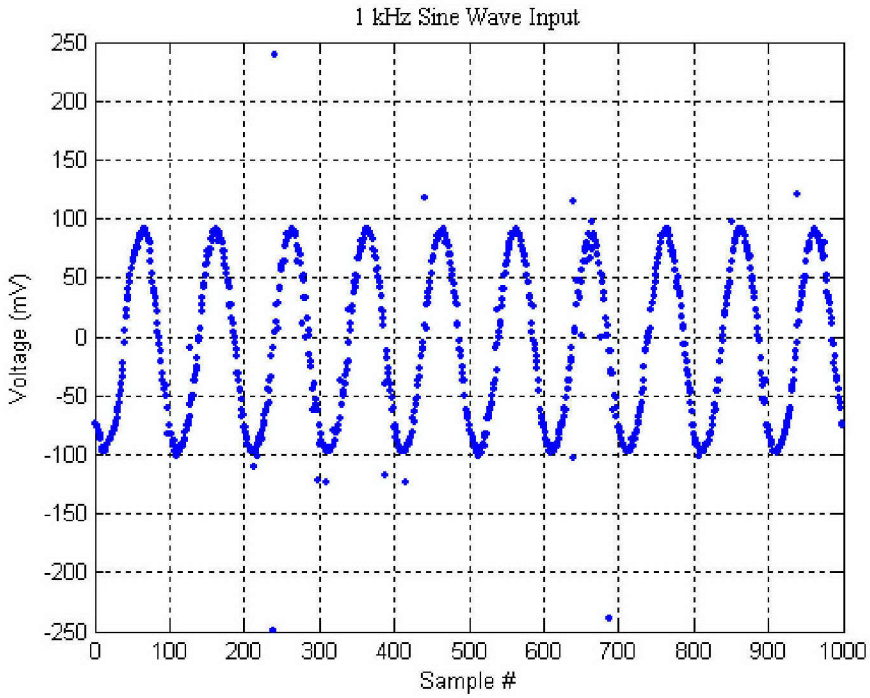

Figure 8. Sampling of a 1-kHz sine wave input to the A/D converters and sampled at a rate of 100 MHz.

Given the 1.90% error, the sine wave was adequately sampled to reproduce the wave itself and this test was seen as a proof of a functioning macro as well as the A/D converters.

The sampling of the A/D converter signal was the culminating point for the significant portion of the macro development for this thesis, which started with exporting the of the SRC clock to the A/D converters. With the development of a functioning sampling system, the data collected must be manipulated in such a way as to produce accurate radar imagery. To do this, the raw, sampled radar signals must be processed using basic radar theory, as described in the next chapter.

V. BASIC RADAR SIGNAL GENERATION THEORY AND THE AN/SPS–65(V)1

A. INTRODUCTION

Sampling and storing the I and Q channel information was a major portion of this thesis work, as described in Chapter IV. With the creation of an adequate sampling interface to the AN/SPS–65(V)1, the next step was to process the data captured. This chapter outlines basic radar signal generation theory applicable to many radar platforms. Building on this theory is a detailed discussion of the AN/SPS–65(V)1 operation.

B. BASIC RADAR IMAGE GENERATION THEORY

A radar system has two basic components, a transmitter and a receiver. The transmitter generates high power, short duration pulses which are radiated by the antenna. Once transmission of the pulse ends, a sensitive antenna mounted receiver collects transmitted pulse reflections for a period of time. When using a rotating antenna, this process is repeated many times as the antenna sweeps through an entire revolution.

From the transmitter, the high energy pulse travels, for the purposes of this thesis, at the speed of light, $c = 3.0 \times 10^8$ m/s. As the pulse collides with objects along the transmission path, some of the pulse energy is reflected off the object. A portion of this reflected energy travels back along the transmission path and is detected by the receiver. This detected signal is then sampled, stored, and/or processed.

The distance, d, that the object is from the radar can be determined by the formula:

$$d = \frac{ct}{2}. \tag{5.1}$$

The variable t is the round trip time of the signal.

This general method was used to generate a basic range map of a sampled analog signal coming from the AN/SPS–65(V)1 radar. Prior to discussing the signal processing on the SRC–6E, a few facts about the radar system used must be explored.

C. AN/SPS–65(V)1 AND RADAR THEORY

1. Basic Radar Theory Applied to AN/SPS–65(V)1 Characteristics

The AN/SPS–65(V)1 operated at a pulse repetition frequency (PRF) of 3,064 transmitted pulses per second. Each transmitted pulse has a width of 7 μs. The transmission and reception of the pulse occurs on a rotating antenna, where $r_a = 4$ seconds per rotation.

The PRF period, T_{PRF}, the time between pulses, is given by:

$$T_{\text{PRF}} = \frac{1}{\text{PRF}} = \frac{1}{3064} = 32.637\ \mu\text{s}. \tag{5.2}$$

Using Eq. (5.1), and inserting T_{PRF} for time, the effective detection distance of the radar is summarized in Table 2. The theoretical range of the signal is d_{ther}.

Table 2. Effective theoretical range of the AN/SPS–65(V)1.

	km	miles	nautical miles
Range	48.956	30.43	26.46

Given the PRF and r_a of the AN/SPS–65(V)1, the total number of pulse intervals per revolution (PPR) is calculated by:

$$\text{PPR} = \text{PRF} \cdot r_a = (3{,}064)(4) = 12{,}256\ \frac{\text{pulses}}{\text{rev}}. \tag{5.3}$$

2. AN/SPS–65(V)1 Pulse Width and Timing

The AN/SPS–65(V)1 uses a trigger signal to initiate the transmitted pulse. The pulse rate of this signal, as described above, was 3,064 pulses per second for all experiments. At the start of, and initiated by, the trigger signal, the transmitter begins to transmit and the receiver turns off. The transmitter transmits for 7 μs , after which the transmitter turns off and the receiver turns on. After T_{PRF}, the trigger signal goes high and the cycle repeats. During this process, the antenna is rotating 360° every four seconds. Thus, the angle rotated through for a given T_{PRF} is the angle per pulse (APP):

$$\text{APP} = \frac{360^\circ}{\text{PPR}} = \frac{360^\circ}{12{,}256} = 0.0294^\circ/\text{pulse}. \tag{5.4}$$

Given basic radar theory applied to the characteristics of the AN/SPS–65(V)1, a number of values were determined throughout this chapter. These values are summarized in Table 3.

Table 3. Figures of interest for Chapter V.

PRF [Hz]	r_a [s per rev]	PPR [pulse/rev]	T_{PRF} [µs]	d_{ther} [km]	APP [º/pulse]
3,064	0.25	12,256	32.64	48.96	0.029

These calculated values are used when sampling the data with the programs written on the SRC–6E. As will be shown in the next chapter, the values in Table 3 are important in the creation of a radar signal processor.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. RADAR SIGNAL PROCESSING WITH THE SRC–6E

A. INTRODUCTION

As shown in Chapter V, the analog radar signal is generated as a result of reception of a reflected radar pulse. The received analog radar signal is then fed through a series of amplifiers and filters and presented in analog form on the I and Q channels for digital conversion [3]. The I and Q channel voltage level are then digitized into a series of 8-bit digital signals at a rate of 100 MHz. The A/D converters, in turn, present the digital data signals to the SRC–6E general-purpose I/O port. The macros described in Chapter IV read the digital signals in at 100 MHz and present the signals as a variables for storage and/or processing.

At this point, the processing of the retrieved radar signals begins. The objective of this chapter is to describe the radar signal processing methodologies explored in the attempts to generate a range map. This chapter will examine the sampling rate used, the resultant data rate incurred, the physical characteristics of the SRC–6E, and the programming means used to process the radar signal given the SRC–6E environment.

B. SAMPLING OF THE ANALOG I CHANNEL SIGNALS

To build a range map, it is only necessary to analyze information from one of the channels. As such, in the remainder of this discussion we focus on processing the I channel information exclusively.

1. Sampling Rates of the A/D Converter and SRC–6E

As mentioned before, the A/D converters operate off the digital clock from the SRC–6E. The clock rate output of the SRC–6E is 100 MHz. The A/D converters sample the radar output (I and Q channels) at the SRC-provided clock rate of 100 MHz [2]. The data from the A/D converters is brought into the SRC via the macro interface described in Chapter IV. The data presented by the macros can be sampled by the MAP functions at a rate of up to 100 MHz. It is important to distinguish between the sampling rate and the storage rate.

2. Ideal Sampling and Effects on Data Rates and Storage Requirements

Referring back to Chapter V, the radar signal is divided into a series of distinct time periods. Within each time period, T_{PRF}, a 7-µs pulse is transmitted and then there is a period of time where the receiver collects reflected energy from the pulse. As shown in Eq. (5.2), the total time takes 32.637 µs and is delimited by trigger signals. There are 32,637 samples per pulse (SPP) repetition as calculated by:

$$\text{SPP} = (32.637\ \mu\text{s})(100\ \text{MHz}) = 32{,}637. \tag{6.1}$$

As the radar output is being sampled at 100 MHz, the range bin of each sample represents a distance of 1.5 m as shown by:

$$\text{Range Bin} = \frac{d_{\text{ther}}}{\text{SPP}} = \frac{48{,}956}{32{,}637} = 1.5\ \text{m}. \tag{6.2}$$

The voltage level of the sampled signal is represented as eight bits. As such, it is possible to determine the ideal data rate, which is 32,637 bytes per pulse. This equates to a data rate of ~100 MBps, given the PRF. To store every sample at the ideal rate for one revolution of the antenna, it would take 400 MB of storage capacity.

3. SRC–6E Available Data Rates and Storage Capacity

The macro provides the sampled data to a MAP function every clock period. Therefore, the data either has to be stored or processed before being lost. There are two possible storage locations within the SRC–6E, the microprocessor side or the MAP side.

To store information to the microprocessor, the data must be streamed via a DMA operation across the internal SNAP port. Unfortunately, the sustained data transmission speed of the SNAP port from MAP to microprocessor is 195 MBs [7]. While this is adequate for just I primary channel data, it would be insufficient for the transmission of I secondary and Q primary and secondary information which would quadruple the data bandwidth. In addition, streaming the data in this manner would bypass the potential advantages of performing the processing on the FPGA in the MAP.

This leaves the MAP for the storage and/or processing of the I primary channel information. As described, a single-side FPGA within the MAP has 24 MB of memory divided evenly between six banks of memory. In order to use the parallel processing capabilities of the MAP, only two, or ideally, one bank of memory should be used to store

data at a time. This leaves the remaining banks of memory free to process previously stored data or for the transfer of data to the microprocessor concurrent to memory storage from the macro. This self-imposed restriction leaves one 4-MB bank of memory available for the storage. At SPP from Eq. (6.2), it would take approximately 122 pulses to fill a bank of memory. Using Eq. (5.4), for the APP, this would be a radar sweep of ~3.5º. Clearly this is inadequate when attempting to generate a range map for an entire radar sweep of 360º.

C. STRATEGIES TO MINIMIZE STORAGE REQUIREMENTS OF THE RADAR IMAGE

In the course of this thesis, two basic strategies were considered in minimizing the storage requirements while retaining the capacity for parallel execution. Those strategies were centered on limiting the number of samples stored and, paradoxically, not storing the samples at all. The next two sections discuss these strategies in detail.

1. Limiting the Number of Samples Stored

The main focus of this strategy was to maintain a storage rate of 100 MHz but limit the range of samples actually stored. The overall goal of this approach was to limit the number of samples stored to 500,000 samples, the approximate number of elements in a single bank of OBM. By storing several samples in a single array element, data packing, eight 8-bit words were stored in a single 64-bit element. Using data packing, it would be possible to store up to 4M samples. Due to coding complexity and time constraints, the data packing approach was not explored. There were, however, a number of approaches considered that limit the number of samples stored.

a. Limit the Range of Sampling

Instead of storing the entire effective range of the radar, as shown in Table 3, it was thought that by limiting the range, the storage requirements would be minimized. To accomplish this, only a limited number of samples would actually be stored from every pulse

If only one sample per pulse per revolution of the antenna was stored, 12,256 elements of memory would be required. If 40 samples per pulse per revolution were stored, it would require 490,024 storage elements, slightly less than the 500k ele-

ments per bank. Using data packing, it would be possible to store 320 samples per pulse. Each stored sample, per Eq. (6.2), represents 1.5 m in range. With data packing, the stored range of the radar would be 480 m and without packing, 60 m. This method severely limited the stored range of radar signals and was abandoned as a viable approach.

b. Limit the Arc Stored

As shown in Eq. (5.4), each pulse represents ~0.03º along the entire sweep of the radar antenna. By chaining together PPR, the entire 360º image is generated. By skipping pulses, the entire effective range could be stored while minimizing the storage capacity required. Each pulse requires SPP number of elements per Eq. (6.1). In terms of storage, 120 pulses with data packing, and 15 without data packing, could be stored in a single bank of OBM. Given pulse storage skipping, each *stored* pulse would represent a 3º- or 24º- sweep of the antenna with respect to data packing. At best, this is a loss in arc resolution of 100 times that provided by the AN/SPS–65. This approach was also abandoned due to the limited arc resolution.

2. Storing a Representation of the Sampled Signal

Limiting the number of samples stored via limiting the range, arc resolution, or some combination was not viable when attempting to compress the data into one unpacked bank of OBM. An alternate strategy is to store a representation of the signals rather than each signal value [11].

a. Averaging a Range of Sampled Values

This method focused on simply averaging the returned voltage levels over a number of samples and storing the average. To represent d_{ther} for 360º in a single OBM, it would require the averaging blocks of 800 samples. The majority of the sampled values were at or near 0 V and reflected images were typically small in scale with values around 20 mV.

Using averaging, the majority samples would tend to dominate the results. This tended to drive the average value below the noise threshold where it was difficult to determine whether there was an actual reflected radar signal. Per Eq. (6.2), a returned radar image would need to be at least 600 m long before significantly affecting the radar image using averaging. Recalling the purpose of the AN/SPS–65(V)1, averaging would

provide adequate detection of missiles and aircraft exceeding 600 m in length. Without reference to actual threat sizes, it was quickly determined that averaging was an inadequate method.

b. Summing a Range of Samples for Storage

Instead of storing the average value, the sum of values was stored. By adding together 700 samples, it was nearly possible to store the complete range for one full rotation of the antenna in a single, unpacked bank of OBM. A block of 700 samples represents 7 μs of time which corresponds to 1.05 km per Eq. (5.1). By summing the sampled values, actual targets were not lost in the sampling method as was with the averaging method.

Summing 700 samples alone was not adequate to meet the unpacked OBM bank size requirement. Each pulse is sampled SPP times. As sampling is continuous throughout this process, the first 700 samples after the PRF trigger signal do not contain useful information. This corresponds to the transmission of pulse in which the receiver is shut off. This leaves $\mathrm{SPP} - 700 = 32{,}637 - 700 = 31{,}937$ samples of interest to represent radar returns between the PRF trigger.

By summing every 700 samples, there are 45.62 sample blocks per pulse. This means there would be ~46 elements to store for each pulse of the radar. Given the PPR, it would take $45.62 \cdot \mathrm{PPR} \approx 559{,}119$ samples to represent one revolution worth of data. This is clearly too large to fit in an unpacked single bank which at maximum (M_{OBM}), 523,776 64-bit elements long [6].

Given $M_{\mathrm{OBM}}/\mathrm{PPR} = 42.74 \approx 42$, 42 is the maximum number of sample blocks per pulse that would fit in an unpacked OBM bank. At 700 samples per block, each pulse would be represented by 29,400 samples at 100 MHz. The effective range of this storage system would be 44.1 km for an entire revolution of the antenna and an arc resolution of ~0.03º. Using this process for storing the radar return information, it is possible to represent radar returns from 1.05 to 45.15 km. Table 4 shows these values and then compares them to the theoretical ranges from Table 3. Comparing this methodology to the theoretical range, there is a difference of 4.86 km. This methodology is appeared to be a likely candidate for storage of the radar range information.

Table 4. Effective range using summation compared to theoretical range.

	km	miles	Nautical Miles
Start	1.05	0.65	0.57
End	45.15	28.06	24.41
Eff. Range	44.10	27.41	23.84
Theory	48.96	30.43	26.46
Difference	4.86	3.02	2.62

The expressed goal of this chapter was an examination of the methodology used to generate a range map from the generated radar returns. In the course of this objective, the ideal sampling rate was shown to exceed the storage capacity in the SRC–6E. Two strategies to circumnavigate this limitation were explored. The resulting strategy stores a radar image in such a way to allow for parallel processing within the FPGA. This final method coupled summation of samples with range limitation. The following chapter is an examination of code and the data collected using derivatives of the summation/range restriction technique.

VII. PROCESSED RADAR SIGNALS AND ANALYSIS OF RESULTS

A. INTRODUCTION

The last chapter examined, in detail, the various factors which shaped the radar signal processing approach. This chapter discusses methods used to process the radar signals from the ANSPS–65(V)1. The methods used here divided the signal processing between three separate coding environments: the MAP functions, the SRC microprocessor, and MATLAB® code. The goal of the initial attempt was two-fold, to verify correct operation of the sample-and-store system and to validate the coded processing environment in conjunction with a 'live feed' from the ANSPS–65(V)1.

B. CODE USED TO STORE AND PROCESS RADAR RETURNS

The programming code used to sample-and-store the raw data was done within a MAP function. The stored data was then transferred to the microprocessor side of the SRC–6E where limited processing occurred prior to being saved as a text file. The text file was imported into MATLAB for static processing.

This coding method departs slightly from the methods described in Chapter VI. The departure is due to the goals of the initial coding test, to prove the sampling system works and perform limited radar processing on stored data. In this manner, the code was not optimized with respect for parallel processing on the FPGAs.

The sampling of data from the A/D converters occurred via a macro (see Chapter IV) called by a MAP function described below. The MAP function was called, in turn, by a microprocessor function which stored the data returned by the MAP function to a file.

Figure 9 shows an abbreviated version of the C-language program used on the microprocessor-side of the SRC–6E to initiate the sample-and-store process. The complete and actual code used can be found in Appendix D, which contains SRC–6E-specific code. This program defines several arrays and then calls a MAP function, STRBITIN, to

fill the arrays with sampled data. The returned data is then translated from A/D converter values to actual voltage levels as described in Eq. (2.1). This converted data is then saved in a file 'output.txt'.

```
#include<stdio.h>

void strbitin ();      // Function Definition

int main () {

    FILE *outfilep;
    outfilep = fopen ("./output.txt","w"); // Output file name

    int i;                // Loop counter
    long long *Ipri;      // I primary channel data
    long long *Isec;      // I secondarychannel data
    long long *Qpri;      // Q primary channel data
    long long *Qsec;      // Q primary channel data
    long long *PRF;       // PRF data

    // Call to the MAP function: start bit input
    strbitin (Ipri, Isec, Qpri, Qsec, PRF);

    // Print the results in the arrays to a file

    int tIp, tIs, tQp, tQs;

    for (i=0;i<MAX_OBM_SIZE;i++) {
        tIp = (Ipri[i] -128)* 1.95;  // Voltage conversion scale
        tIs = (Isec[i] -128)* 1.95;  // Voltage conversion scale
        tQp = (Qpri[i] -128)* 1.95;  // Voltage conversion scale
        tQs = (Qsec[i] -128)* 1.95;  // Voltage conversion scale

        fprintf(outfilep,"%-7d %-4d %-4d %-4d %-4d %-4lld\n",
        i, tIp, tIs, tQp, tQs, PRF[i]);    //Print arrays
    }   // End for (i=0;i<MAX . . .

    return(0);
}   // End main.c
```

Figure 9. Code used in main.c.

Figure 10 shows a compressed version of the code presented in Appendix D.2. In this MAP function, the macro 'bitin' is called. Bitin is the macro developed in Chapter IV. This macro returns one 32-bit integer, *radarSig*, and one 1-bit integer, PRF. The variable *radarSig* is the composed of four 8-bit integers, each representing the I and Q primary and secondary channels. PRF is the signal that originates from the AN/SPS–65(V)1, indicating the triggered beginning of a radar pulse as described in Chapters II

and V. The MAP function, STRBITIN, collects data from the macro, unpacks the data, and stores it in a separate array. As shown in the Appendix D.2, each array is actually a bank of OBM memory.

```
/*********************************************************************
* Thesis: Interface 32 bits input to main.c, four 8-bit and one 1-bit
*         variable out to calling program.
*
* Macro Called: bitin.  This macro samples 33 data lines and
* returns the values to this function.
*
* Programs calling this function:
*     Pass by reference five 64-bit integer element length arrays
*     ~500,000 elements long.
*
* This function then splits out the 32 bit input returned by the
* macro bitin into four arrays where each array represents:
*           A = I primary   channel information = bits 7:0
*           B = I secondary channel information = bits 15:8
*           C = Q primary   channel information = bits 23:16
*           D = Q secondary channel information = bits 31:24
*           E = PRF information from a separate variable
* When an array is filled to MAX_OBM_SIZE, the OBM banks are
* passed back to the calling function.
*
*********************************************************************/

void strbitin (long long A[], long long B[], long long C[],
               long long D[], long long E[])               {

    int radarSig, PRF, i; // storage var, stor var, counter

    out = 55;  // Initialize out

    for (i=0;i<MAX_OBM_SIZE;i++) {

        bitin(&radarSig, &PRF);  // Get data from pin(s)

        // Variable Out packed with four variables.  Unpacking here.
        A[i] =  radarSig & 0x00000000000000ff;           // I prim
        B[i] = (radarSig & 0x000000000000ff00) >>  8;    // I sec
        C[i] = (radarSig & 0x0000000000ff0000) >> 16;    // Q prim
        D[i] = (radarSig & 0x00000000ff000000) >> 24;    // Q sec
        E[i] = PRF;                                     // PRF
    }   // End for (i=0;i<MAX . . .
} // End strbinin
```

Figure 10. MAP function STRBITIN used to store data from the macro bitin.

The data stored by main was then imported into MATLAB. The complete code for this is included in Appendix D.3. The MATLAB code performed the basic process-

ing described in Chapter VI, Section C.2.b, without the range limitation. Also, the plot in Fig. 11 was generated via MATLAB.

C. TESTING OF CODE AND GENERATED IMAGE

With the code described in the section above, all the necessary components to test the SRC-to-radar interface were implemented and ready for evaluation.

1. Test Setup

a. AN/SPS–65(V)1 System Setup

The AN/SPS–65(V)1 antenna returned a signal along a fixed azimuth approximately true north of the antenna position. The only values returned to the I and Q channels were readings along this fixed azimuth.

b. A/D Converter Setup

An A/D converter was connected to the radar I channel provided by the AN/SPS–65(V)1. The converter only provided the 8-bit I primary channel data. The A/D board was physically connected to the SRC via the general-purpose I/O port within the SRC–6E.

c. SRC and Signal Processing Setup

Using the macro developed in Chapter IV and the code discussed in this chapter, an executable program was run several times. Each time, a collection of data samples were stored for later processing. A series of approximately 16 transmit/receive cycles (pulse interval) were recorded and processed along the fixed azimuth path. Since this setup does not store an entire antenna revolution worth of data, there were no range restrictions imposed by the processing method. Recorded ranges, using this method, should correspond to the values shown in Table 3.

2. Results

An analysis of the output file directly shows that there were 32,639 samples between the start of trigger signals. Compared to SPP, this is a sampling error of 0.006% from the expected SPP. The sampling system closely matched the PRF signal expectations.

Figure 11 shows a plot of the processed information as captured with the system. The plot is the summation of voltage values returned by the radar system compared to the

range in kilometers. In addition, the summation of the trigger signal, PRF, was plotted. When the PRF transitions from low to high, 0 to 20,000, it is an indication of the beginning of a transmitted pulse by the AN/SPS–65(V)1. From there, the range calculations began.

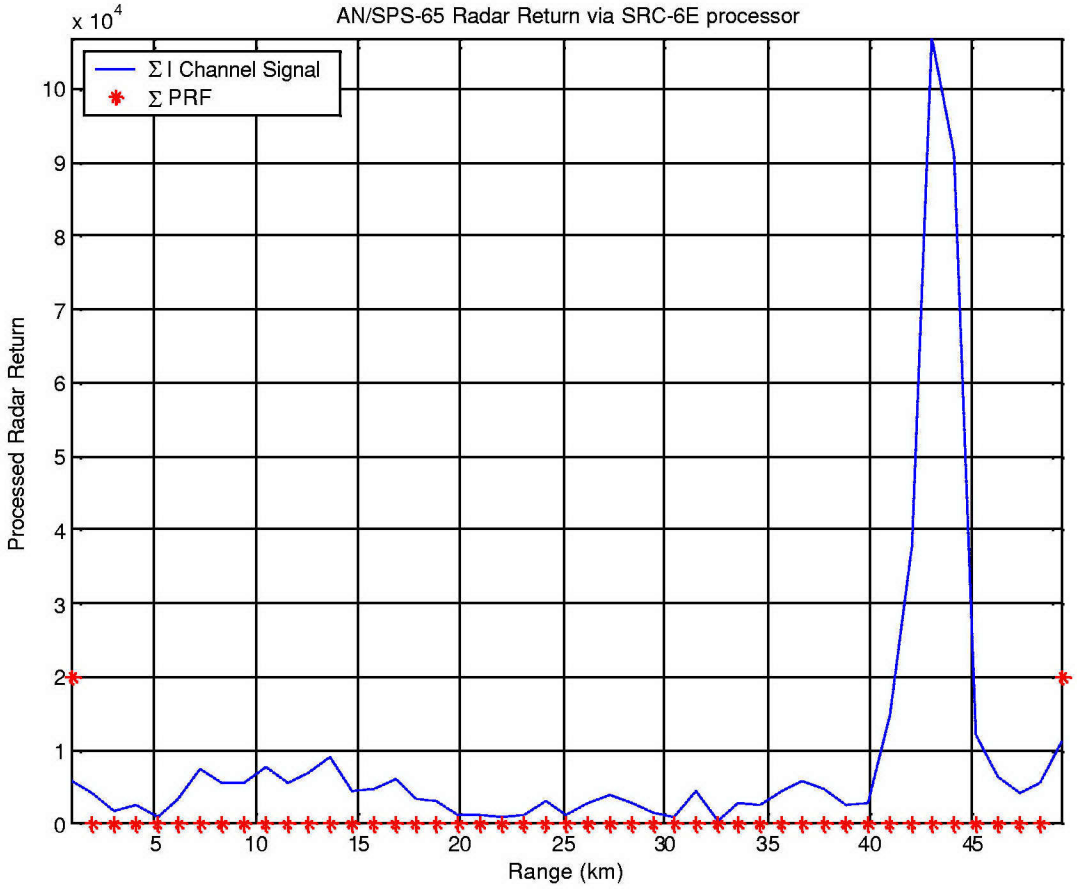

Figure 11. Processed radar image.

At approximately 40 km, there is the beginning of a large radar return. Figure 12 shows the distance from the antenna position to the Santa Cruz Mountain range. This distance is shown to be approximately 40 km, suggesting that this is the source of the return. (Figure 12 was generated with the Department of Defense geological information system standard program ESRI® ArcMap, version 8.3, and data generated by the Monterey Bay Aquarium Research Institute with the assistance Arlene Guest, Senior Lecturer, Department of Oceanography, Naval Postgraduate School.)

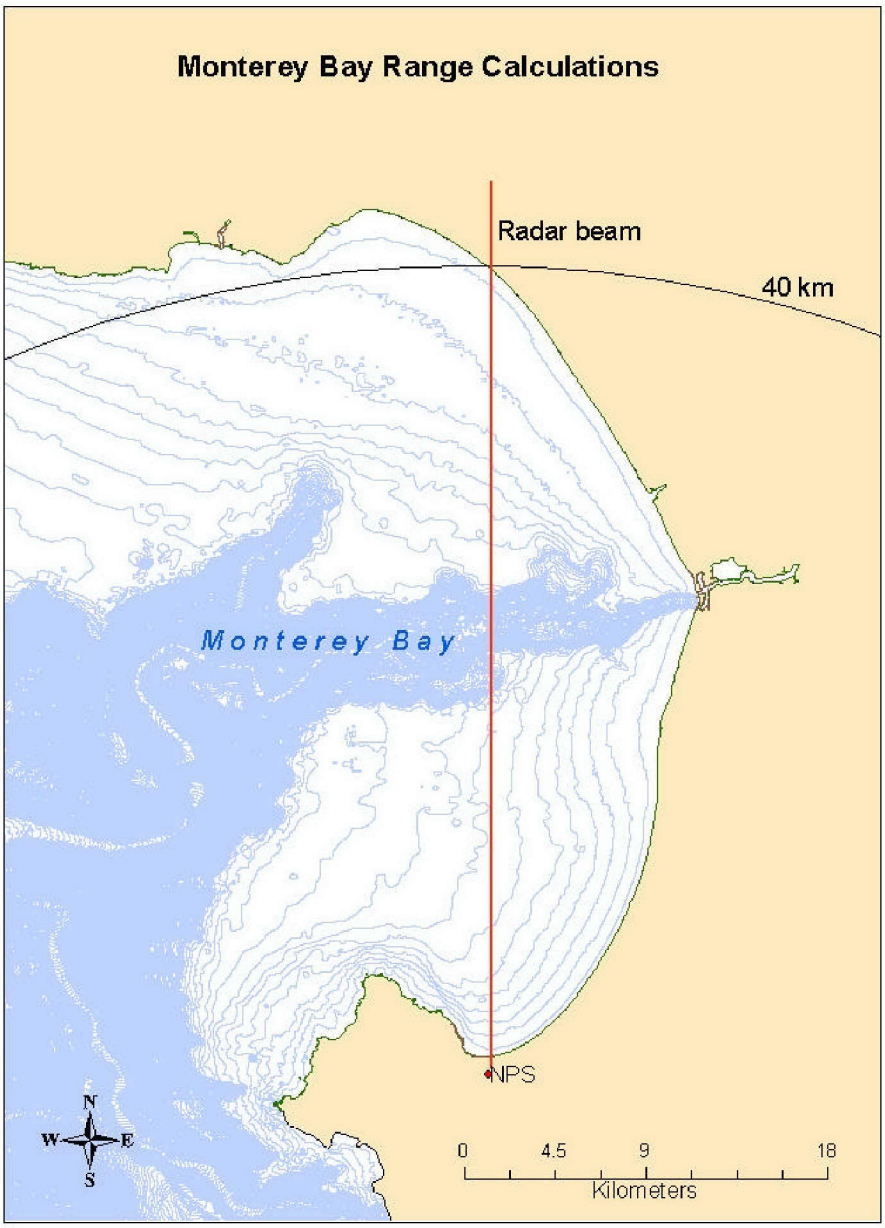

Figure 12. Range from the Naval Postgraduate School (NPS) to Santa Cruz Mountains.

The apparent success served as the capstone to the material of this chapter which outlined the programs developed to process a radar signal. Also, the radar signal processing was tested and evaluated against real world data. The results of which satisfied the expressed objectives of this thesis work as noted in the conclusion, Chapter VIII.

VIII. CONCLUSION

A. SUMMARY

The objective of this thesis was to build a software interface to the AN/SPS 65(V)1 that would allow for the limited processing of radar signals. The intricacies of the various systems used, the radar, the A/D converters, and the SRC–6E, were discussed in order to provide a basis for understanding the thesis work.

The thesis work focused on three major constructs: macro development, radar theory and image processing, and programming and testing a radar processing device. The macro development progressed sequentially along two lines, clocking and data input. Eventually, the two lines of development were merged to form a cohesive macro which provided a unified software interface to the A/D converters for later radar processing.

The radar processing research explored a variety of processing options. For various reasons, as described above, a number of the processing options were abandoned. A suitable processing method was discovered and used as a basis for the developed software system.

The programmed radar processor spanned three environments: the SRC MAP function, microprocessor-based processing, and a third program that provided a static display of the radar signal. By integrating the various components, a radar image was sampled, stored, processed and displayed. As shown in Chapter VII, the system implemented accurately reported on received radar returns.

B. SUGGESTED FUTURE WORK

While this thesis work completed the objectives, there were still uncompleted project goals. Namely, sophisticated radar processing software was not implemented and the potential advantages of reconfigurable computing were not demonstrated fully as outlined in the project objectives.

The software developed here was simply a tool to prove or disprove the correct functioning of the A/D converters, the SRC–6E general-purpose I/O ports, macros, and associated software to sample and store radar signal data. The implemented software

lacks the sophistication of modern radar signal processing systems. The system built can be greatly improved to provide a robust suite of radar processing techniques.

The system implemented in this thesis did not fully demonstrate the potential advantages that reconfigurable computing is reported to offer. It is possible that the system developed could process not only one type of radar, but multiple radar systems in a nearly simultaneous manner. The feasibility of this supposition is one which could be further explored by harnessing the inherent parallelism that a reconfigurable computing platform offers. Chapter VI provides a possible framework from which that research could start.

Despite the limitations described in this section, a viable software interface to a radar system was developed and tested. The skills called upon to do this encompassed:

- General circuit theory and design.
- Logic circuit theory and design.
- Familiarization with several programming languages (C, VHDL, MATLAB).
- Basic and advanced programming topics such as function calls and parallel processing.
- Basic radar theory and design.
- Basic radar processing.
- Signal processing.
- Timing analysis of circuits, logic, and programmed code.
- Thorough working knowledge of the general reconfigurable computing platform used.
- Working knowledge of logic circuit design tools.

The suggested future work would require, in addition to this skill set, an intermediate understanding of signal processing and radar theory.

APPENDIX A: TRADEMARKS

Several terms used throughout this thesis are trademarked. This appendix is a list of trademark terms used in this document.

Intel® is a registered trademark of the Intel Corporation.

ISE, or Integrated Software Environment, is a trademark of Xilinx, Inc.

MAP® is a registered trademark of SRC Computers, Inc.

MATLAB® is a registered trademark of The Mathworks, Inc.

SNAP™ is a trademark of SRC Computers, Inc.

Virtex®-II is a registered trademark of Xilinx, Inc.

Xilinx® is a registered trademark of Xilinx, Inc.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: CONNECTIVITY DIAGRAMS

The following appendix is a copy of the pin-out configuration of the Xilinx Virtex II showing the pad names and the respective connections and name on the A/D boards provided. The first table is a connectivity diagram from the perspective of the physical layout of the pins. The second table rearranges the information into a data view perspective. Both tables contain essentially the same information, but in views that proved useful during the programming and configuration portions of the thesis.

Table 5. Physical view of the pin-out connections.

Mictor/Physical View					
Breakout Board	**Mictor Pin**	**Xilinx Pad**	**Breakout Board**	**Mictor Pin**	**Xilinx Pad**
D00	1	U12	D57	58	F1
D01	2	T12	D58	59	G1
D02	3	M2	D59	60	G5
D03	4	N2	D60	61	H5
D04	5	R11	D61	62	K8
D05	6	T11	D62	63	J8
D06	7	P8	D63	64	F3
D07	8	N8	D64	65	G3
D08	9	M3	D65	66	H7
D09	10	N3	D66	67	H6
D10	11	M5	D67	68	N12
D11	12	N5	D68	69	M12
D12	13	R10	D69	70	E2
D13	14	P10	D70	71	F2
D14	15	K1	D71	72	F5
D15	16	L1	VAL0	73	G6
D16	17	L4	FULL	74	M11
D17	18	M4	VAL1	75	L12
D18	19	P9	VAL2	76	E4
D19	20	N9	VAL3	77	F4
D20	21	L3	VAL4	78	E1
D21	22	L6	NC_1	79	F20
D22	23	M6	SPRO	80	NOT CONN
D23	24	R12	SPR1	81	D2
D24	25	P12	SPR2	82	E3
D25	26	J2	SPR3	83	B4
D26	27	K2	SPR4	84	C4
D27	28	J4	SPR5	85	C6
D28	29	K4	SPR6	86	C5
D29	30	M8	SPR7	87	E8

Mictor/Physical View						
Breakout Board	**Mictor Pin**	**Xilinx Pad**		**Breakout Board**	**Mictor Pin**	**Xilinx Pad**
D30	31	L8		SPR8	88	F8
D31	32	H1		IDAT	89	A5
D32	33	J1		ICLK	90	A4
D33	34	L7		DCLK	91	B6
D34	35	M7		I-D0	92	B5
D35	36	P11		I-D1	93	H11
D36	37	N11		PRST	94	H12
D37	38	D1		MRST	95	C8
D38	39	F19		PR0	96	C7
D39	40	NOT CONN		PR1	97	E7
D40	41	K5		PR2	98	D8
D41	42	J6		PRS	99	E10
D42	43	K6		RSVD	100	E9
D43	44	N10		BNK0	101	NOT CONN
D44	45	M10		BNK1	102	NOT CONN
D45	46	H3		BNK2	103	J10
D46	47	J3		BNK3	104	H10
D47	48	G4		SEG0	105	J11
D48	49	H4		SEG1	106	G10
D49	50	M9		SEG2	107	G9
D50	51	L9		SEG3	108	F9
D51	52	G2		NC_2	109	NOT CONN
D52	53	H2		NC_3	110	NOT CONN
D53	54	J7		NC_4	111	NOT CONN
D54	55	K7		NC_5	112	NOT CONN
D55	56	L10		WR CLK	113	NOT CONN
D56	57	K9		MST CLK	114	NOT CONN

Table 6. Data view of the pin-out configuration.

	Data View							
Breakout	**I**	**Mictor Pin**	**Xilinx Pad**		**Q**	**Mictor Pin**	**Xilinx Pad**	**Breakout**
D00	I0	1	U12	Least Sig	Q0	2	T12	D01
D02	I1	3	M2		Q1	4	N2	D03
D04	I2	5	R11		Q2	6	T11	D05
D06	I3	7	P8		Q3	8	N8	D07
D08	I4	9	M3		Q4	10	N3	D09
D10	I5	11	M5		Q5	12	N5	D11
D12	I6	13	R10		Q6	14	P10	D13
D14	I7	15	K1	Most Sig	Q7	16	L1	D15
D16	I8	17	L4	Least Sig	Q8	18	M4	D17
D18	I9	19	P9		Q9	20	N9	D19
D20	I10	21	L3		Q10	22	L6	D21
D22	I11	23	M6		Q11	24	R12	D23
D24	I12	25	P12		Q12	26	J2	D25
D26	I13	27	K2		Q13	28	J4	D27
D28	I14	29	K4		Q14	30	M8	D29
D30	I15	31	L8	Most Sig	Q15	32	H1	D31

Breakout	**AGC**	**Mictor Pin**	**Xilinx Pad**	
D33	AGC0	34	L7	Least Sig
D35	AGC1	36	P11	
D37	AGC2	38	D1	
D39	AGC3	40	NOT CONN	
D41	AGC4	42	J6	
D43	AGC5	44	N10	
D45	AGC6	46	H3	
D47	AGC7	48	G4	Most Sig
D49	AGC8	50	M9	Least Sig
D51	AGC9	52	G2	
D53	AGC10	54	J7	
D55	AGC11	56	L10	
D57	AGC12	58	F1	
D59	AGC13	60	G5	
D61	AGC14	62	K8	
D63	AGC15	64	F3	Most Sig

Misc.	**Mictor Pin**	**Xilinx Pad**
Clock Out	79	F20
Clock In	39	F19
PRF In	57	K9

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: MACRO DEVELOPED

Chapter IV outlined the creation of a macro which would sample the signals provided by the A/D converters. What follows is a series of schematics and VHDL code used for the final macro developed.

This schematic is an overall view of the schematic used. The scale of this schematic is such that not all the details are immediately visible. The following schematics in this appendix provide a detailed view of the various sections.

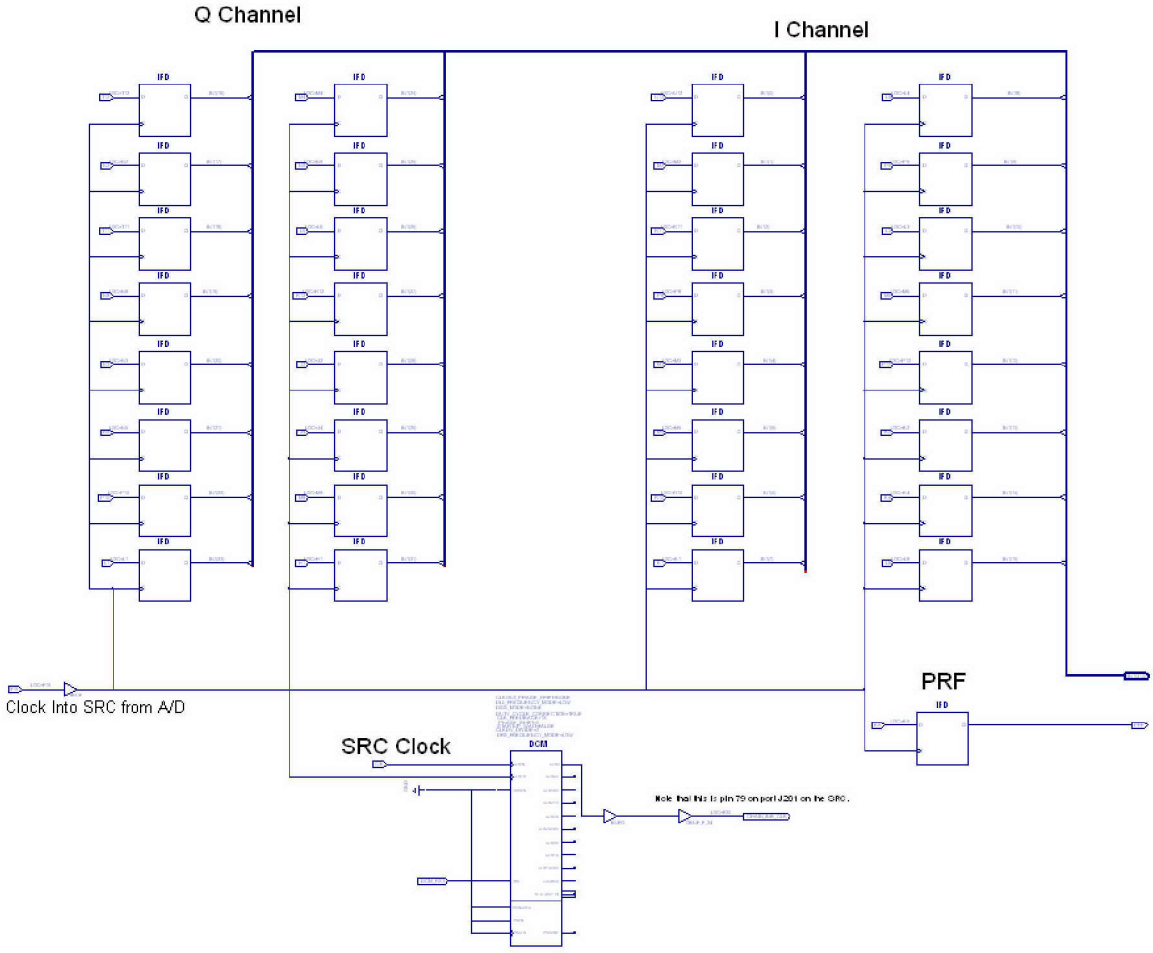

Figure 13. Overview of the macro developed.

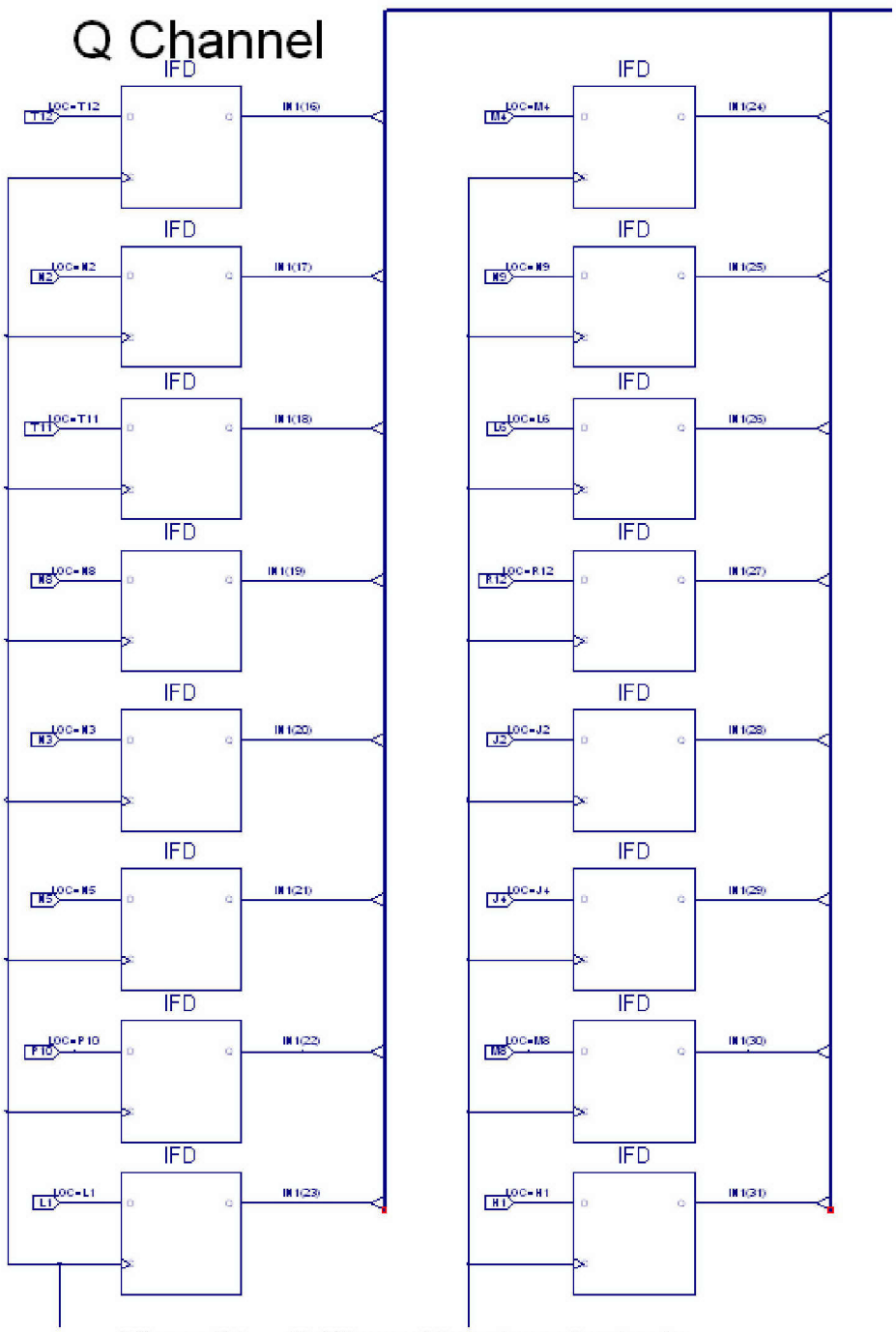

Figure 14. Q Channel inputs and outputs.

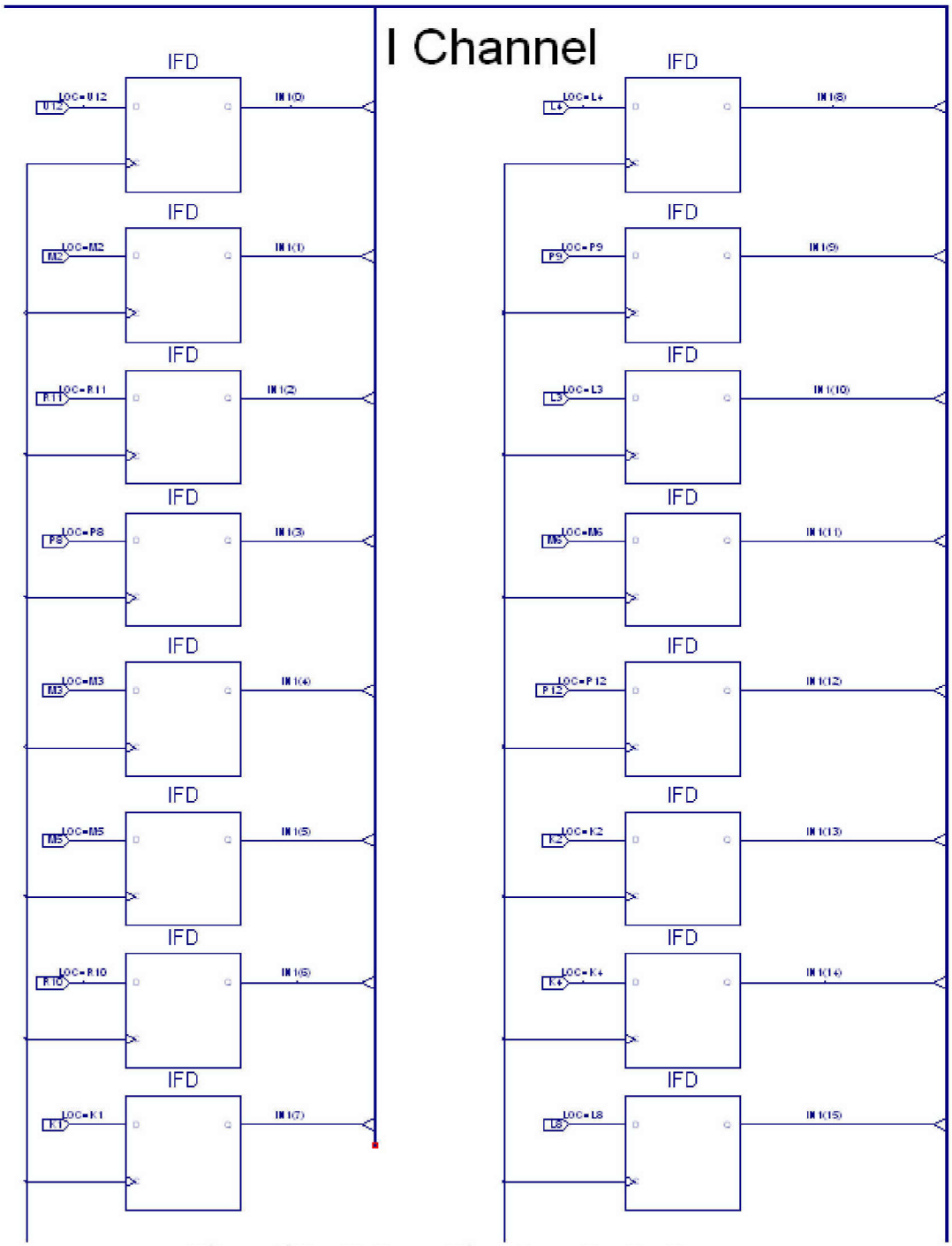

Figure 15. I channel inputs and outputs.

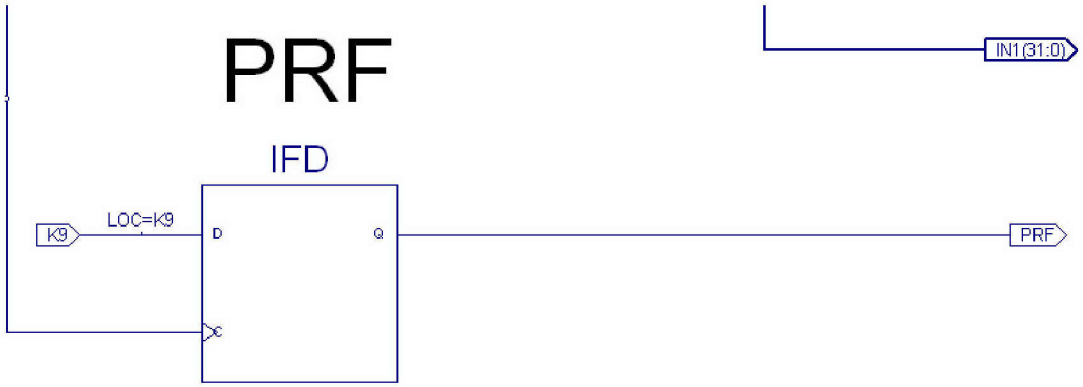

Figure 16. PRF signal input and output. Also the output marker for the I and Q channels.

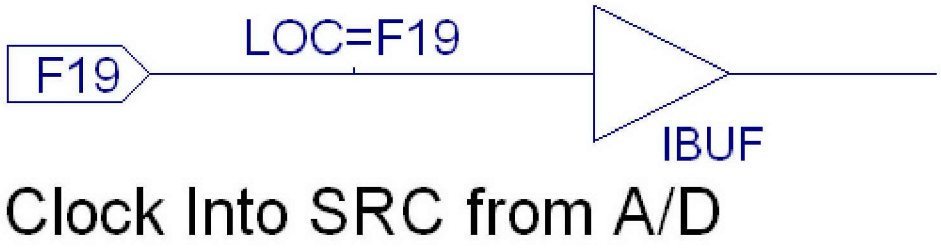

Figure 17. Data Ready Signal interface.

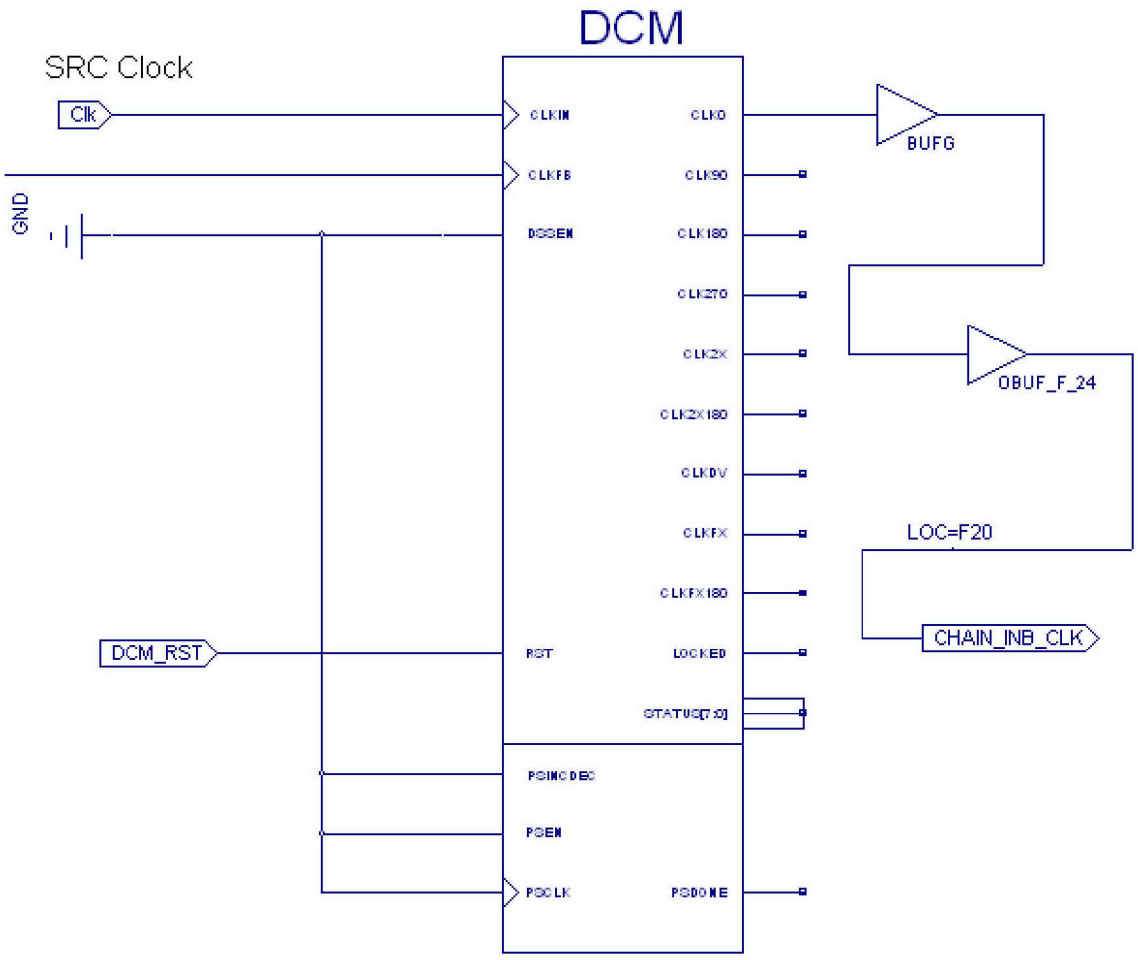

Figure 18. DCM portion of the macro.

```
-- VHDL model created from C:\Xilinx\virtex2\data\drawing\ifd.sch - Sun
Jan 30 12:48:11 2005

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
-- synopsys translate_off
library UNISIM;
use UNISIM.Vcomponents.ALL;
-- synopsys translate_on

entity IFD_MXILINX_in33 is
   port ( C : in    std_logic;
          D : in    std_logic;
          Q : out   std_logic);
end IFD_MXILINX_in33;

architecture BEHAVIORAL of IFD_MXILINX_in33 is
   attribute INIT       : STRING ;
   attribute BOX_TYPE   : STRING ;
   attribute IOB        : STRING ;
   attribute IOSTANDARD : STRING ;
   signal D_IN   : std_logic;
   signal XLXN_1 : std_logic;
   signal XLXN_2 : std_logic;
   component FDCE
      -- synopsys translate_off
      generic( INIT : bit :=  '0');
      -- synopsys translate_on
      port ( C   : in    std_logic;
             CE  : in    std_logic;
             CLR : in    std_logic;
             D   : in    std_logic;
             Q   : out   std_logic);
   end component;
   attribute INIT of FDCE : COMPONENT is "0";
   attribute BOX_TYPE of FDCE : COMPONENT is "BLACK_BOX";

   component IBUF
      port ( I : in    std_logic;
             O : out   std_logic);
   end component;
   attribute IOSTANDARD of IBUF : COMPONENT is "LVTTL";
   attribute BOX_TYPE of IBUF : COMPONENT is "BLACK_BOX";

   component VCC
      port ( P : out   std_logic);
   end component;
   attribute BOX_TYPE of VCC : COMPONENT is "BLACK_BOX";

   component GND
      port ( G : out   std_logic);
   end component;
   attribute BOX_TYPE of GND : COMPONENT is "BLACK_BOX";
```

```
   attribute IOB of I_36_15 : LABEL is "TRUE";

begin
   I_36_15 : FDCE
      port map (C=>C, CE=>XLXN_2, CLR=>XLXN_1, D=>D_IN, Q=>Q);

   I_36_24 : IBUF
      port map (I=>D, O=>D_IN);

   I_36_26 : VCC
      port map (P=>XLXN_2);

   I_36_29 : GND
      port map (G=>XLXN_1);
end BEHAVIORAL;

-- VHDL model created from in33.sch - Sun Jan 30 12:48:11 2005

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;
-- synopsys translate_off
library UNISIM;
use UNISIM.Vcomponents.ALL;
-- synopsys translate_on

entity in33 is
   port ( Clk           : in    std_logic;
          DCM_RST       : in    std_logic;
          F19           : in    std_logic;
          H1            : in    std_logic;
          J2            : in    std_logic;
          J4            : in    std_logic;
          K1            : in    std_logic;
          K2            : in    std_logic;
          K4            : in    std_logic;
          K9            : in    std_logic;
          L1            : in    std_logic;
          L3            : in    std_logic;
          L4            : in    std_logic;
          L6            : in    std_logic;
          L8            : in    std_logic;
          M2            : in    std_logic;
          M3            : in    std_logic;
          M4            : in    std_logic;
          M5            : in    std_logic;
          M6            : in    std_logic;
          M8            : in    std_logic;
          N2            : in    std_logic;
          N3            : in    std_logic;
          N5            : in    std_logic;
          N8            : in    std_logic;
          N9            : in    std_logic;
          P8            : in    std_logic;
          P9            : in    std_logic;
          P10           : in    std_logic;
          P12           : in    std_logic;
```

```
          R10           : in    std_logic;
          R11           : in    std_logic;
          R12           : in    std_logic;
          T11           : in    std_logic;
          T12           : in    std_logic;
          U12           : in    std_logic;
          CHAIN_INB_CLK : out   std_logic;
          IN1           : out   std_logic_vector (31 downto 0);
          PRF           : out   std_logic);
   attribute LOC : STRING ;
   attribute LOC of F19 : SIGNAL is "F19";
   attribute LOC of H1 : SIGNAL is "H1";
   attribute LOC of J2 : SIGNAL is "J2";
   attribute LOC of J4 : SIGNAL is "J4";
   attribute LOC of K1 : SIGNAL is "K1";
   attribute LOC of K2 : SIGNAL is "K2";
   attribute LOC of K4 : SIGNAL is "K4";
   attribute LOC of K9 : SIGNAL is "K9";
   attribute LOC of L1 : SIGNAL is "L1";
   attribute LOC of L3 : SIGNAL is "L3";
   attribute LOC of L4 : SIGNAL is "L4";
   attribute LOC of L6 : SIGNAL is "L6";
   attribute LOC of L8 : SIGNAL is "L8";
   attribute LOC of M2 : SIGNAL is "M2";
   attribute LOC of M3 : SIGNAL is "M3";
   attribute LOC of M4 : SIGNAL is "M4";
   attribute LOC of M5 : SIGNAL is "M5";
   attribute LOC of M6 : SIGNAL is "M6";
   attribute LOC of M8 : SIGNAL is "M8";
   attribute LOC of N2 : SIGNAL is "N2";
   attribute LOC of N3 : SIGNAL is "N3";
   attribute LOC of N5 : SIGNAL is "N5";
   attribute LOC of N8 : SIGNAL is "N8";
   attribute LOC of N9 : SIGNAL is "N9";
   attribute LOC of P8 : SIGNAL is "P8";
   attribute LOC of P9 : SIGNAL is "P9";
   attribute LOC of P10 : SIGNAL is "P10";
   attribute LOC of P12 : SIGNAL is "P12";
   attribute LOC of R10 : SIGNAL is "R10";
   attribute LOC of R11 : SIGNAL is "R11";
   attribute LOC of R12 : SIGNAL is "R12";
   attribute LOC of T11 : SIGNAL is "T11";
   attribute LOC of T12 : SIGNAL is "T12";
   attribute LOC of U12 : SIGNAL is "U12";
   attribute LOC of CHAIN_INB_CLK : SIGNAL is "F20";
end in33;

architecture BEHAVIORAL of in33 is
   attribute BOX_TYPE              : STRING ;
   attribute CLK_FEEDBACK          : STRING ;
   attribute CLKDV_DIVIDE          : STRING ;
   attribute CLKFX_DIVIDE          : STRING ;
   attribute CLKIN_PERIOD          : STRING ;
   attribute CLKFX_MULTIPLY        : STRING ;
   attribute CLKIN_DIVIDE_BY_2     : STRING ;
   attribute CLKOUT_PHASE_SHIFT    : STRING ;
   attribute DESKEW_ADJUST         : STRING ;
```

```
attribute DFS_FREQUENCY_MODE    : STRING ;
attribute DLL_FREQUENCY_MODE    : STRING ;
attribute DSS_MODE              : STRING ;
attribute DUTY_CYCLE_CORRECTION : STRING ;
attribute PHASE_SHIFT           : STRING ;
attribute STARTUP_WAIT          : STRING ;
attribute HU_SET                : STRING ;
attribute IOSTANDARD            : STRING ;
signal LD            : std_logic;
signal XLXN_355      : std_logic;
signal XLXN_356      : std_logic;
signal XLXN_606      : std_logic;
component BUFG
   port ( I : in    std_logic;
          O : out   std_logic);
end component;
attribute BOX_TYPE of BUFG : COMPONENT is "BLACK_BOX";

component OBUF_F_24
   port ( I : in    std_logic;
          O : out   std_logic);
end component;
attribute BOX_TYPE of OBUF_F_24 : COMPONENT is "BLACK_BOX";

component DCM
   -- synopsys translate_off
   generic( CLK_FEEDBACK : string :=  "1X";
            CLKDV_DIVIDE : real :=  2.000000;
            CLKFX_DIVIDE : integer :=  1;
            CLKIN_PERIOD : real :=  0.000000;
            CLKFX_MULTIPLY : integer :=  4;
            CLKIN_DIVIDE_BY_2 : boolean :=  false;
            CLKOUT_PHASE_SHIFT : string :=  "NONE";
            DESKEW_ADJUST : string :=  "SYSTEM_SYNCHRONOUS";
            DFS_FREQUENCY_MODE : string :=  "LOW";
            DLL_FREQUENCY_MODE : string :=  "LOW";
            DSS_MODE : string :=  "NONE";
            DUTY_CYCLE_CORRECTION : boolean :=  true;
            PHASE_SHIFT : integer :=  0;
            STARTUP_WAIT : boolean :=  false);
   -- synopsys translate_on
   port ( CLKFB    : in    std_logic;
          CLKIN    : in    std_logic;
          DSSEN    : in    std_logic;
          PSCLK    : in    std_logic;
          PSEN     : in    std_logic;
          PSINCDEC : in    std_logic;
          RST      : in    std_
```

Figure 19. VHDL code of developed macro.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D: CODE USED IN THESIS

1. COMPLETE MAIN.C SAMPLE AND STORAGE OF RADAR SIGNAL

```
#include<stdio.h>
#include<libmap.h>    // New lib to facilitate map calls
#include <map.h>      // New lib to facilitate map calls

void strbitin ();     // Function Definition

int main () {

    FILE *outfilep;
    outfilep = fopen ("./oS30Jan5M.txt","w"); // Writing to file

    int nmap, mapnum,i;  // New Number of maps, map # to be used, loop
    long long *Ipri;     // I primary channel data
    long long *Isec;     // I secondarychannel data
    long long *Qpri;     // Q primary channel data
    long long *Qsec;     // Q primary channel data
    long long *PRF;     // PRF data


    Ipri = (long long*) Cache_Aligned_Allocate (MAX_OBM_SIZE*sizeof(int64_t));
    Isec = (long long*) Cache_Aligned_Allocate (MAX_OBM_SIZE*sizeof(int64_t));
    Qpri = (long long*) Cache_Aligned_Allocate (MAX_OBM_SIZE*sizeof(int64_t));
    Qsec = (long long*) Cache_Aligned_Allocate (MAX_OBM_SIZE*sizeof(int64_t));
    PRF  = (long long*) Cache_Aligned_Allocate (MAX_OBM_SIZE*sizeof(int64_t));

    mapnum = 0;

    nmap = 1;

    // allocate map to this problem.  Get a map ready
    if (map_allocate (nmap)) {
       fprintf (stdout, "Map allocation failed.\n");
       exit (1);
    }  // End if (map . . .

    // Call to the MAP function: start bit input
    strbitin (Ipri, Isec, Qpri, Qsec, PRF, mapnum);

    if (map_free (nmap)) {
        printf ("Map deallocation failed. \n");
        exit (1);
    }  // End if (map . . .

    // Print the results in the arrays to a file

    int tIp, tIs, tQp, tQs;

    for (i=0;i<MAX_OBM_SIZE;i++) {
        tIp = (Ipri[i] -128)* 1.95;
        tIs = (Isec[i] -128)* 1.95;
        tQp = (Qpri[i] -128)* 1.95;
        tQs = (Qsec[i] -128)* 1.95;

        fprintf(outfilep,"%-7d %-4d %-4d %-4d %-4d %-4lld\n",
                i, tIp, tIs, tQp, tQs, PRF[i]);    //Print arrays
    }   // End for (i=0;i<MAX . . .
  return(0);
}
```

2. MAP FUNCTION START BIT INPUT

```
/*********************************************************************
* Thesis: Interface 32 bits input to main.c, four 8-bit and one 1-bit
*         variable out to calling program.
*
* Tom Guthrie  24 Jan 05
*
* This function calls a macro.  The macro samples 33 data lines and
* returns the values to this function.
*
* This function then splits out the 32 bit input into four OBM banks
* where:
*           Bank A = I primary   channel information = bits 7:0
*           Bank B = I secondary channel information = bits 15:8
*           Bank C = Q primary   channel information = bits 23:16
*           Bank D = Q secondary channel information = bits 31:24
*           Bank E = PRF information from a separate variable
*
* When filled to MAX_OBM_SIZE, the OBM banks are passed back to the
* calling function.
*********************************************************************/

#include<libmap.h>          // New lib to facilitate map calls

void strbitin (long long A[], long long B[], long long C[],
               long long D[], long long E[], int mapnum)   {

    int out, out2, i, temp; // storage var, stor var, counter, temp var

    OBM_BANK_A (AL, int64_t, MAX_OBM_SIZE)  // I primary
    OBM_BANK_B (BL, int64_t, MAX_OBM_SIZE)  // I secondary
    OBM_BANK_C (CL, int64_t, MAX_OBM_SIZE)  // Q primary
    OBM_BANK_D (DL, int64_t, MAX_OBM_SIZE)  // Q secondary
    OBM_BANK_E (EL, int64_t, MAX_OBM_SIZE)  // PRF
    out = 55;  // Initilize out

    for (i=0;i<MAX_OBM_SIZE;i++) {
        bitin(&out, &out2);  // Get data from pin(s)
        // Variable Out packed with four variables.  Unpacking here.
        AL[i] =  out & 0x00000000000000ff;           // I prim
        BL[i] = (out & 0x000000000000ff00) >>  8;    // I sec
        CL[i] = (out & 0x0000000000ff0000) >> 16;    // Q prim
        DL[i] = (out & 0x00000000ff000000) >> 24;    // Q sec
        EL[i] =  out2;                                // PRF
    }   // End for (i=0;i<MAX . . .

    temp = MAX_OBM_SIZE*sizeof(int64_t);  // Here to facilitate DMA->CM.
                                          // is the length of the entire
                                          // array.
   // Transfer the I primary channel data array back to calling program
   DMA_CPU (OBM2CM, AL, MAP_OBM_stripe(1,"A"), A, 1, temp, 0);
   wait_DMA (0);
   // Transfer the I secondary channel data array back to calling program.
   DMA_CPU (OBM2CM, BL, MAP_OBM_stripe(1,"B"), B, 1, temp, 0);
   wait_DMA (0);
   // Transfer the Q primary channel data array back to calling program.
   DMA_CPU (OBM2CM, CL, MAP_OBM_stripe(1,"C"), C, 1, temp, 0);
   wait_DMA (0);
   // Transfer the Q secondary channel data array back to calling program.
   DMA_CPU (OBM2CM, DL, MAP_OBM_stripe(1,"D"), D, 1, temp, 0);
   wait_DMA (0);
   // Transfer the PRF data array back to calling program.
   DMA_CPU (OBM2CM, EL, MAP_OBM_stripe(1,"E"), E, 1, temp, 0);
   wait_DMA (0);
} // End strbinin
```

3. MATLAB CODE USED TO TEST SAMPLED DATA

```
% Thesis Test Data
% Tom Guthrie
%
% Date: 24 Feb 05

clear; close all;

% Get data from input file for processing and graphing
% The input file name varies depending on the sample
% taken.

BlkSz  = 700;
numBlk = 47;
% w & x used to select a range of samples.
w = 198992;             % Start range
x = w + BlkSz*numBlk;   % End range
                        % 523776 = MAX_OBM_SIZE

load -ASCII oR4Febkf.txt % File to be loaded

i    = oR4Febkf(w:x,1);  % Sample number
Ipri = oR4Febkf(w:x,2);  % I primary channel voltage
%Isec = oR4Febkf(w:x,3);  % Not used in this example
%Qpri = oR4Febkf(w:x,4);
%Qsec = oR4Febkf(w:x,5);
PRF  = 100 * oR4Febkf(w:x,6);  % PRF signal
                               % Scaled for plot
                               % visibility.

% Summing the raw data values of the radar system.
% This loop sums every BlkSz samples then stores it
% in a one dimensional array.  The I primary
% voltage level and the PRF are targets for the
% the summation process.
% The outer loop (q) divides the number of samples
% samples (i) into blocks which are BlkSz long
% samples long.  The inner loop sums the BlkSz
% values which are stored in sumI and sumPRF.
% sumPRF is scaled previously by a constant to make it
% highly visible in subsequent plots.  Notice the
% summation is of the absolute voltage value of
% the I primary voltage.  In this coding example,
% i = sample number, I = Ipri = voltage level of
% of the I channel.

temp   = 0;  % Temporary storage variables
temp2  = 0;

for q = 1:numBlk
    for r = 1:BlkSz
        temp  = temp  + abs(Ipri((q-1)*BlkSz+r));
        temp2 = temp2 + abs(PRF ((q-1)*BlkSz+r));
    end
    sumI(q)   = temp;
    sumPRF(q) = temp2;
    temp      = 0;  % Reset the temp variables
    temp2     = 0;
end

kmpBlk = BlkSz * 1.5/1000;
mpBlk  = kmpBlk * 0.6215;
nmpBlk = kmpBlk * 0.540541;

Rkm = linspace(1, kmpBlk * numBlk, numBlk); % Range km
Rm  = linspace(1, mpBlk  * numBlk, numBlk); % Range miles
```

```
Rnm = linspace(1, nmpBlk * numBlk, numBlk); % Range naut. miles

figure;
plot(Rkm, sumI, Rkm, sumPRF, 'r*');
axis tight;
xlabel('Range (km)');
ylabel('Processed Radar Return');
title ('AN/SPS-65 Radar Return via SRC-6E processor');
legend('\Sigma I Channel Signal','\Sigma PRF',2);
grid on;

figure
plot(Rnm, sumI, Rnm, sumPRF, 'r*');
axis tight;
xlabel('Range (nautical miles)');
ylabel('Processed Radar Return');
title ('AN/SPS-65 Radar Return via SRC-6E processor');
legend('\Sigma I Channel Signal','\Sigma PRF',2);
grid on;

figure
plot(Rm, sumI, Rm, sumPRF, 'r*');
axis tight;
xlabel('Range (miles)');
ylabel('Processed Radar Return');
title ('AN/SPS-65 Radar Return via SRC-6E processor');
legend('\Sigma I Channel Signal','\Sigma PRF',2);
grid on;
```

LIST OF REFERENCES

[1] Zhi Guo, Walid Najjar, Frank Vahid, and Kees Vissers, "A quantitative analysis of the speedup factors of FPGAs over processors," *Proc. of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, pp. 162–170, ACM Press, New York, 2004.

[2] Timothy L. King, *Hardware interface to connect an AN/SPS–65 radar to an SRC–6E reconfigurable computer*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 2005.

[3] Buczynski, Paul, "AN/SPS–65(V)1 Radar System," Revision 7/02, Naval Postgraduate School, July 2002, (unpublished).

[4] Arnold E. Acker, "How to Speak Radar: Basic Fundamentals and Applications of Radar," Varian Associates, Inc., Palo Alto, California, August 1988.

[5] David Caliga and David Peter Barker, "Delivering Acceleration: The Potential for Increased HPC Application Performance Using Reconfigurable Logic," ACM No. 1-58113-293-X/01/0011, Colorado Springs, November 2001.

[6] "SRC Carte® Programming Environment v1.8 Guide," SRC–007–13, SRC Computers Inc., Colorado Springs, October 5, 2004

[7] Notes for SRC–6E Training Class, SRC Computers, 2003, (unpublished).

[8] Chris Orth, "User hardware interfacing to the chain port of the SRC MAP® REV D board," SRC-013-00, SRC Computer, Inc., Colorado Springs, 2004 (unpublished).

[9] *Libraries Guide*, ISE 6.Ii, XILINX®, San Jose, 2003.

[10] "SRC-6E MAP© Macro Developers Guide," SRC–008–01, SRC Computers Inc., Colorado Springs, September 23, 2002.

[11] D. Fouts and P. Pace, private conversation at Naval Postgraduate School, January 2005.

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. Chairman, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

4. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

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

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

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