AFRL-IF-RS-TR-2004-108
Final Technical Report
April 2004

SIGNAL COLLECTION PROCESSING ENHANCEMENTS

Black River Systems, Inc.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-108 has been reviewed and is approved for publication

APPROVED: /s/
ALFREDO VEGA IRIZARRY
Project Engineer

FOR THE DIRECTOR: /s/
JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 2004	**FINAL** **Jul 03 – Jan 04**

4. TITLE AND SUBTITLE

SIGNAL COLLECTION PROCESSING ENHANCEMENTS

5. FUNDING NUMBERS

C - F30602-03-C-0205
PE - 62702F
PR - 558B
TA - BA
WU - 07

6. AUTHOR(S)

David Welchons

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Black River Systems Company, Inc.
162 Genesee Street
Utica NY 13502

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFEC
32 Brooks Road
Rome NY 13441-4114

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2004-108

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Alfredo Vega Irizzary/IFEC/(315) 330-2382 Alfredo.Vega_Irizzary@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The objective of this effort was to develop a Signal Collection Processing (SCP) software testbed that would provide a user with the capability to rapidly prototype and evaluate signal processing systems starting with MATLAB algorithms or portions of a MATLAB algorithm, and then transitioning the systems to an embedded real-time processing architecture. This testbed allows for automated signal analysis and data collection. An Air Force Research Laboratory (AFRL) algorithm, Polyphase Filter Channelizer ((PFC) and reconstruction algorithm application were selected for validating the Signal Collection Processing Testbed Polyphase Filter Channelization and reconstruction has uses in many signal processing applications.

14. SUBJECT TERMS

Real Time Testbed Architecture, Signal Collection Processing, Automated Data Collection, Signal Identification, Polyphase Filter Channelizer Algorithm

15. NUMBER OF PAGES 63

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

FIGURES

TABLES

1.0 Signal Collection Processing Testbed Overview

The objective of this effort was to develop a Signal Collection Processing (SCP) software testbed that would provide a user with the capability to rapidly prototype and evaluate signal processing systems starting with MATLAB algorithms or portions of a MATLAB algorithm and then transitioning the system to an embedded real-time processing architecture. This testbed allows for automated signal analysis and data collection. The testbed concept was validated using a selected Air Force Research Laboratory (AFRL) application. A Polyphase Filter Channelizer (PFC) and reconstruction algorithm application was selected for validating the Signal Collection Processing Testbed Polyphase Filter Channelization and reconstruction has uses in many signal processing applications. The PFC algorithm allows wide-band signals to be "demultiplexed" into multiple narrow band signals, with each narrow-band channel at a significantly lower data rate than the original wide-band signal.

The testbed hardware architecture for this project is a Mercury Computer Systems AdapDev system. The AdapDev system provides the capability to use real-time hardware in an environment which also provides windows-based real-time software development tools. The AdapDev hardware architecture consists of a 1.26 GHz Pentium III processor, 1GB SDRAM, 18.2 GB hard drive, 52X CD-ROM, and 2 Mercury Vantage RT High Compute (HCD) boards. The HCD boards connect to the PCI bus and the Mercury RACE++ fabric. These boards each contain 4 MCP7410 PPC processors with Altivec vector processing capability, 2 MB L2 cache, 256 MB SDRAM. This environment is ideal for the testing of real-time signal processing algorithms since the Mercury hardware, utilizing the Race++ high speed communication interconnect fabric between the 8 PPC Computer Environments (CEs), is a similar architecture to what is found in many operational embedded systems. The real-time software development tools in this testbed consist of the MULTI® Integrated Development Environment (IDE), containing a text editor, debugger, builder, version control, error checker, profiler, graphical browser and C compiler. Other tools in the environment are the TATL Trace Analysis Tool, the Scientific Algorithm Library (SAL), and the Parallel Acceleration System (PAS) library.

A unique capability of the key software architecture developed under this effort is the ability to call a C function defined in a Dynamically Linked Library (DLL) from the MATLAB environment. This capability allows access to the Mercury real-time hardware from MATLAB without having to totally rewrite the source code. A systems engineer would typically conduct signal processing algorithm analysis and design using the MATLAB environment because of MATLAB's rapid prototyping capability The software architecture developed under this effort extends that capability allowing the systems engineer to analyze and design the algorithm in an environment that closely resembles the operational target environment. This software architecture provides the systems engineer with a procedure to develop, run, and analyze an algorithm on Mercury real-time hardware, while still working in the MATLAB environment. The developer

can check the results of the real-time algorithm against the results of the MATLAB algorithm in the MATLAB environment.

The software architecture consists of four pieces.

- A MATLAB .m file script providing the user's MATLAB interface, which gets pointers to the input and output data and passes them into the DLL
- A DLL function which runs on the host and is called from the .m file script, and sends the input data and requested algorithm type to the Mercury target processor
- An interface function on the target processor to recognize which algorithm is to be executed
- The Mercury implementation of the algorithm itself.

Figure 1 below provides a diagram of the Signal Collection Processing software architecture.

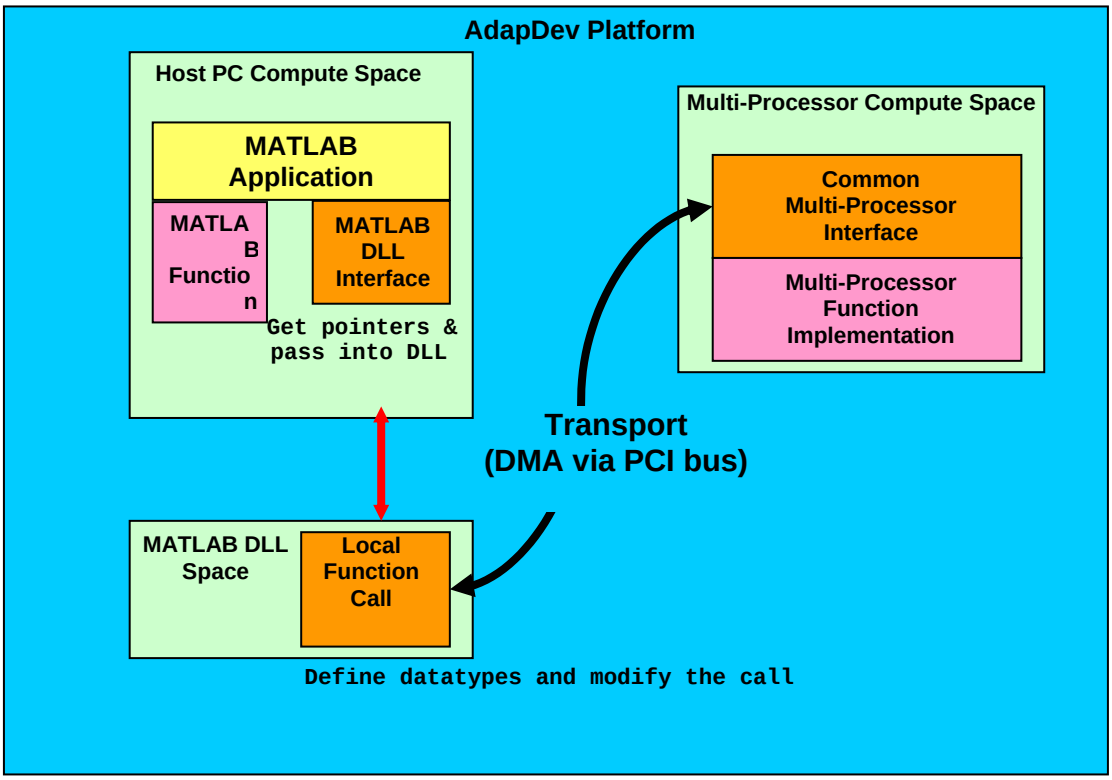


Figure 1 Signal Collection Processing Software Architecture

2.0 Using the Signal Collection Processing Environment

This section describes the steps required for converting a MATLAB algorithm to a real-time algorithm that can be executed both in the MATLAB environment and the SCP environment. To illustrate the steps a simple transpose algorithm will be presented.

2.1 First Step: Create a MATLAB M File

The first step is to create a MATLAB .m file script in a "matlab" directory with the name of the file beginning with **"mc"**.

EXAMPLE: Creating the M file

D:/Projects/SCP/matlab/mc_transpose.m

This script is interpreted by MATLAB and does not require any compilation. Next in the M file the following steps need to be done:

1. Check to make a line is added to the M file to load the DLL.
2. Create any intermediate or output vectors that are necessary.
3. Create pointers to vectors required by DLL C function.
4. Perform a "calllib" to the DLL function, perform any necessary conversions on output data.

EXAMPLE: Below is a sample MATLAB mfile for transpose that has addressed the list above.

```
function c = mc_transpose(a)

ll = libisloaded('ml_to_mc');

if ll
   if isreal(a)
      [ra,ca] = size(a);
      c = zeros(ca,ra);

      % Since matlab internal structure of matrix is by columns
      % and C internal structure of matrix is by rows, must
      % perform transpose before/after calling C function.
      at = transpose(a);
      atp = libpointer('singleRef',at);
      ct = transpose(c);
      ctp = libpointer('singleRef',ct);
      calllib('ml_to_mc', 'mc_transpose_real',ra, ca, atp, ctp )
      ct = get(ctp,'value');
      c = ct';
```

```
   else
      % Complex matrix a
      [ra,ca] = size(a);
      c = zeros(ca,ra);

      % Since matlab internal structure of matrix is by columns
      % and C internal structure of matrix is by rows, must
      % perform transpose before/after calling C function.
      at = transpose(a);
      atrp = libpointer('singleRef',real(at));
      atip = libpointer('singleRef',imag(at));
      ct = transpose(c);
      ctrp = libpointer('singleRef',real(ct));
      ctip = libpointer('singleRef',imag(ct));
      calllib('ml_to_mc', 'mc_transpose_complex_split',ra, ca, atrp,atip,ctrp,ctip);
      ctr = get(ctrp,'value');
      cti = get(ctip,'value');
      ct = complex(ctr,cti);
      c = transpose(ct);
   end;
else
   fprintf('ml_to_mc library not loaded\n');
   c = [];
end;
```

2.2 Step 2: Create a Function in the DLL

The second step is to create a function in the DLL.

EXAMPLE: Creating the c and h files

D:/Projects/SCP//host_VC/ml_to_mc/ml_to_mc/ml_to_mc.c
D:/Projects/SCP/include/ml_to_mc.h).

This C program is compiled for execution on the host using the Microsoft Visual C++ compiler. In the C file the following list below should be addressed:

1. Declare arrays of inputs pointers, output pointers, sizes of inputs, sizes of outputs, and data types of inputs and outputs.
2. Call mc_rpc_call and pass the ce number, number of inputs, size inputs array, data type inputs array, inputs pointer array, number of outputs, size outputs array, data type outputs array, outputs pointer array.
3. Example of transpose_complex_split() function in ml_to_mc.c is:

EXAMPLE: Sample C file addressing items described above

```
declspec( dllexport ) void mc_transpose_complex_split(int r,
        int c,
        float *ar,
        float *ai,
        float *br,
        float *bi)
{
    /*    int i,j;
          int vec_b_i, vec_a_i;
        */
    long   rc;
    CEID   ce;
    void  *inputs          [4] = {&r, &c, ar, ai};
    void  *outputs          [2] = {br, bi};
    int    size_inputs      [4] = {1,  1,  r*c, r*c};
    int    size_outputs     [2] = {r*c, r*c};
    char   *data_type_inputs [4] =
{"INTEGER32","INTEGER32","FLOAT32","FLOAT32"};
    char   *data_type_outputs[2] = {"FLOAT32","FLOAT32"};

    /* b[c][r] = transpose ( a[r][c] ) */
    /*    for (i=0; i<r; i++)
          {
        for (j=0; j<c; j++)
        {
        vec_b_i = j*r+i;
        vec_a_i = i*c+j;
        b[j*r+i] = a[i*c+j];
        }
          }
        */
    ce = allocate_ce();
    if (ce == -1) {
        fprintf(fp,"mc_transpose_complex_split: Error on allocate_ce()\n");
        return;
    }
    rc = mc_rpc_call(ce,
          "transpose_complex_split",
          4,
          size_inputs,
          data_type_inputs,
          inputs,
          2,
          size_outputs,
```

```
        data_type_outputs,
        outputs);

   rc = free_ce(ce);
   if (rc == -1) {
      fprintf(fp,"mc_transpose_complex_split: Error on free_ce(ce=%d)\n",ce);
      return;
   }
}
```

2.3 Step 3: Defining the Mercury Interface Code

The next step is to add in the necessary interface code to in order to access the Mercury hardware platform.

EXAMPLE: Mercury c file

D:/Projects/SCP/target/slave_mc_rpc.c

This C program needs to be compiled for execution on the host using the MULTI C compiler. This file will also need the following modification made to it.

1. Add "else if" to the string compare done on function type, and call the new algorithm function.

EXAMPLE: slave_mc_rpc.c modification

```
    } else if (strcmp(local_master_smb_cmd.func,"transpose_complex_split") == 0) {
        transpose_complex_split();
```

2.4 Step 4: Include Additional Mercury Code

The next step is to add Mercury code to implement the new algorithm using a SAL call. In the file **slave_mc_rpc.c** the following changes are required:

1. The input and output data is obtained from global variables inputs, outputs, data_type_outputs and must be placed into the appropriate local variables.

2. The appropriate SAL function is called using the local variables.

   EXAMPLE: transpose_complex_split() in slave_mc_rpc.c:

EXAMPLE: Modified Mercury slave_mc_rpc.c file

```
void transpose_complex_split()
{
   COMPLEX_SPLIT cs_input;
   COMPLEX_SPLIT cs_output;
   int *pr = inputs[0];
   int *pc = inputs[1];
   long rc = 0;

   /* Call SAL routine for complex split transpose. */
   if (verbose) fprintf(fp,"%s chosen\n",local_master_smb_cmd.func);
   if (strcmp(local_master_smb_cmd.data_type_outputs[0],"FLOAT32") == 0) {
      rc = tatl_trace_user_event(trid, CE_ALG_TRANSPOSE_COMPLEX_SPLIT, 0, 0);
      if (rc) {
         fatal(rc, "tatl_trace_user_event, CE_ALG_TRANSPOSE_COMPLEX_SPLIT");
      }
      if (verbose) fprintf(fp,"FLOAT32 chosen\n");
      cs_input.realp = inputs[2];
      cs_input.imagp = inputs[3];
      cs_output.realp = outputs[0];
      cs_output.imagp = outputs[1];
      zmtransx(&cs_input,
            1,
            &cs_output,
            1,
            *pc,
            *pr,
            SAL_NNN  /* ESAL_FLAG  CCN.e.g.cache.cache.noncache */
            );
      if (verbose) fprintf(fp,"After zmtransx\n");
   } else {
      fprintf(fp,"FLOAT32 not chosen..instead %s\n",
            local_master_smb_cmd.data_type_outputs[0]);
   }
}
```

2.5 Executing on the Mercury Hardware

Before executing on the Mercury hardware, the hardware must be initialized. This is can be accomplished by typing the MATLAB command

mc_init('2','3-9')

The mc_init('2','3-9') designates CE 2 as the name server and initializes the CEs 2-9 on the Mercury hardware. The numbering starts at 2 since the host is referred to as CE 1.

The mc_init command performs the **configmc** and **sysmc** commands which are necessary to initialize the Mercury hardware. The configmc command uses a text file to initialize the Mercury hardware for the current configuration of hardware components and their connections. The text file that was used under this effort is located in the directory D:/MercurySoftware/AdapDev/AdapDev_welchons.cfg. The sysmc command loads the Mercury operating system on each of the 8 nodes of the Mercury boards. After mc_init has been run, any of the "mc_" algorithms can be run. The command **mc_stop** is used to reset the Mercury hardware. This command also results in the creation of text files produced by prints on the host and CEs. These files are named output_from_ceN.txt where N is 1 for the host, 2 for CE 2, 3 for CE 3, etc. These files will usually be found in D:/Projects/SignalCollection/ProcessingEnhancements/PolyphaseFilter/welchons/host_VC/ml_to_mc/ml_to_mc, unless the MATLAB current directory has been changed by a "cd", in which case the files will be found in that "cd" directory.

2.6 Testing

The MATLAB environment can be used to compare the results of the MATLAB algorithm to the outputs of the Mercury real-time algorithm. Below is a simplified set of steps for testing an algorithm.

1. Choose a small test data set for use by the MATLAB algorithm.
2. Run the MATLAB algorithm and record the results including the intermediate results.
3. Create a Mercury 'by-parts" version of the algorithm (m file).
   a. Start with the MATLAB version of the algorithm.
   b. Replace the low-level functions with low-level Mercury function calls.
4. Run the Mercury "by-parts" version and compare to the Mercury results, making corrections to the Mercury low-level functions where necessary.
5. Create a complete Mercury version of the MATLAB algorithm, including prints of intermediate results and TATL calls in the code implemented for the Mercury target.
6. Run the code and compare the prints to the results in the MATLAB and Mercury "by-parts" versions. In the DLL environment, prints can only be made to files. The files are dumped by the MATLAB command **mc_stop**. These files are named **output_from_ceN.txt** where **N** is 1 for the host, 2 for CE 2, 3 for CE 3, etc. The files for this effort have been placed in the directory called D:/Projects/SignalCollection/ProcessingEnhancements/PolyphaseFilter/welchons/host_VC/ml_to_mc/ml_to_mc, unless the MATLAB current directory has been changed by a "cd", in which case the files will be found in that "cd" directory. Before running again, the MATLAB **mc_init** command must be reissued. To evaluate algorithm timing information, the Mercury **tatlview** tool can be used. To execute tatlview, a MKS Kornshell window can be opened, and the tatlview command is typed in that window. The file to be opened is **tatl_dump_data.tat.** This file is created when the **mc_stop** command is executed, and it will be found in the same directory described above for the output_from_CEN.txt files. The events in this file are defined in the file stored in the directory

D:/Projects/SignalCollectionProcessingEnhancements/PolyphaseFilter/welchons/include/tatl_events.h, and are placed in the target algorithms via calls to **tatl_trace_user_event()** to represent important time measurement locations in the algorithm. For additional information refer to the Mercury TATL Developer's Guide.

2.7 Timing the Algorithms

To evaluate algorithm timing information, the Mercury **tatlview** tool can be used. To execute tatlview, a MKS Kornshell window is opened, and the tatlview command is typed in that window. The file to be opened is **tatl_dump_data.tat**. This file is created when the **mc_stop** command is executed, The events in this file are defined in the file D:/Projects/SignalCollectionProcessingEnhancements/PolyphaseFilter/welchons/include/tatl_events.h, and are placed in the target algorithms via calls to **tatl_trace_user_event()** to represent important time measurement locations in the algorithm. There are other calls which are required to initialize the TATL data collection capability and to terminate the data collection. These calls are **tatl_trace_create(), tatl_trace_start(), tatl_trace_stop(), tatl_trace_shutdown()**. For additional information refer to the Mercury TATL Developer's Guide. A macro can be developed to contain both the initialization commands together, or both the termination commands together, but the current implementation executes the functions individually. Examples of using these functions are provided for the host in:

D:/Projects/SignalCollectionProcessingEnhancements/PolyphaseFilter/welchons/host_VC/ml_to_mc/ml_to_mc/ml_to_mc.c

and for the target in:

D:/Projects/SignalCollectionProcessingEnhancements/PolyphaseFilter/welchons/target/slave_mc_rpc.c.

A separate command must be issued on the host in order to dump the file of data. In the TATL Developer's Guide the command is described as **sys_tatl()** when called from a program. Problems were encountered in using this function, so instead the current implementation is a call to **system(tatl_cmd)** and example of the mc_stop() command is provided in

D:/Projects/SignalCollectionProcessingEnhancements/PolyphaseFilter/welchons/host_VC/ml_to_mc/ml_to_mc/ml_to_mc.c

on the SCP hardware.

2.8 Aborts

The execution of a Mercury "mc_" function may occasionally result in a MATLAB abort. In most cases the reason for the abort will be found in one of the text output files produced by the host or one of the CEs. These files are named **output_from_ceN.txt** where **N** is 1 for the host, 2 for CE 2, 3 for CE 3, etc. These files will usually be found in

D:/Projects/SignalCollection/ProcessingEnhancements/PolyphaseFilter/welchons/host_VC/ml_to_mc/ml_to_mc,

unless the MATLAB current directory has been changed by a "cd", in which case the files will be found in that "cd" directory. If there is no file output, then it may be necessary to use the MATLAB debugger or the Visual C++ debugger to determine the source of the abort.

3.0 Validation of the SCP Testbed

A Polyphase Filter Channelization (PFC) and Perfect Reconstruction algorithm developed by AFRL/IFEC was used for validating the SCP testbed. PFC and Perfect Reconstruction have many uses in signal processing applications. PFC allows wide-band signals to be "demultiplexed" into multiple narrow band signals, with each narrow-band channel at a significantly lower data rate than the original wide-band signal. Within each narrow-band channel, signal filtering, detection, and measurement is often much simpler, and less computationally demanding, than equivalent processing on the original signal. After narrow band processing is complete, the individual channels can be recombined to re-form the now-processed wide-band signal. Although the addition of channelization and reconstruction add processing to the original wide-band algorithm, the total processing of the multiple narrow-band channels is often significantly lower than wide-band processing would require.

A basic block diagram of a Polyphase Filter Channelizer is shown in Figure 2.

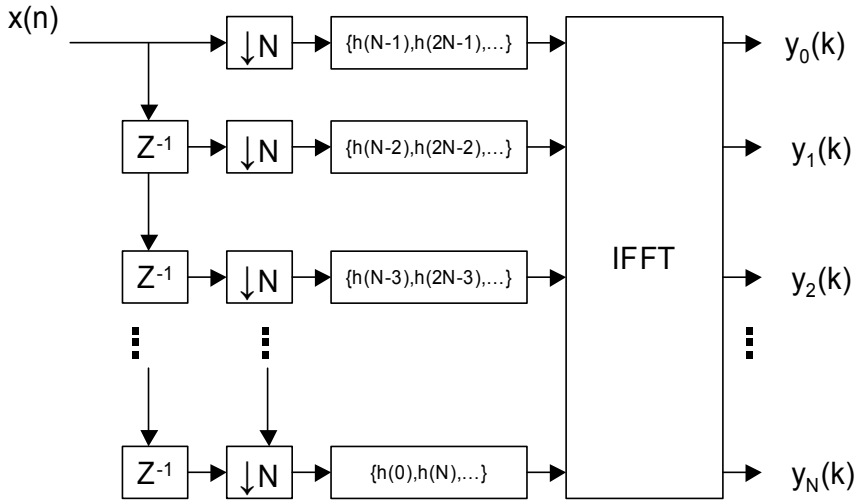


Figure 2 Polyphase Filter Channelization Algorithm

The incoming data samples, x(n), are passed through a series of delays (Z^{-1}), down-sampled (↓N)), and filtered (h(…)), before being passed through an N-point Inverse Fast Fourier Transform (IFFT). The outputs from the IFFT are the channelized data streams.

A block diagram of a Channel Reconstruction filter is shown in Figure 3.

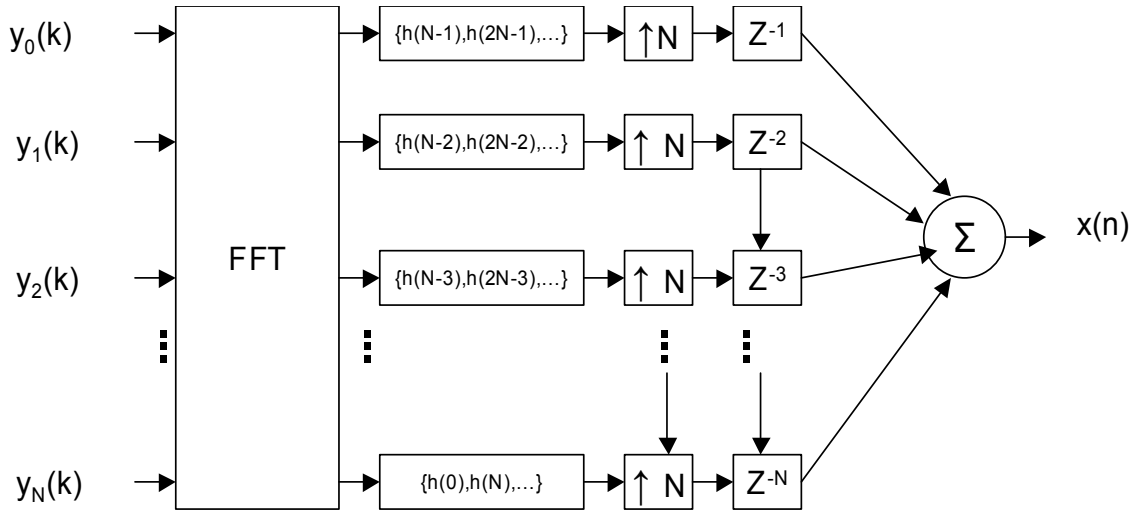


Figure 3 Channel Reconstruction Algorithm

The channelized data streams, y(k), are put through an FFT, re-filtered (h(…)), up-sampled, delayed (Z^{-1}), and summed to recreated the wideband signal stream X(n). These algorithms as well as low-level algorithms used in these algorithms were implemented using the SCP testbed.

3.1 Low Level Algorithms Implemented in SCP Testbed

In order to the implement the PFC and Perfect Reconstruction algorithm some low level algorithms needed to be implemented in the SCP testbed. The low-level algorithms that were used and implemented in the SCP testbed consisted of a Transpose, Reorder Rows, Convolution, FFT, and an IFFT algorithm. Below is a description of each of the algorithms:

3.1.1 Transpose

Algorithm Name	b = mc_transpose(a)
Algorithm Description	Perform matrix transpose (interchange the Rows with the columns in a matrix.
Inputs	Matrix a
Outputs	Matrix b
Limitations	Real or Complex data
MATLAB Filename	mc_transpose.m
DLL Function Name	mc_transpose_real() mc_transpose_complex_split()
Target Function Names	transpose_real() transpose_complex_split()

3.1.2 Reorder Rows

Algorithm Name	c = mc_reorder_rows(a)
Algorithm Description	Reverse the order of the rows in a matrix.
Inputs	Matrix A
Outputs	Matrix B
Limitations	Real or Complex data
MATLAB Filename	mc_reorder_rows.m
DLL Function Name	mc_reorder_rows_real() mc_reorder_rows_complex_split()
Target Function Names	reorder_rows_real() reorder_rows_complex_split()

3.1.3 Convolution

Algorithm Name	y = mc_convolve_use_state(x,h,state)
Algorithm Description	Prepend the state to x and perform the convolution of x and h.
Inputs	Vector x, Vector h, Vector State
Outputs	Vector y
Limitations	Real or Complex data. No zero padding of input vector is performed. To match MATLAB convolution, the state vector must be one less than the size of h and consist of zeros, since MATLAB assumes that much zero padding on the x input vector.
MATLAB Filename	mc_convolve_use_state.m
DLL Function Name	mc_convolve_output_size_real() mc_convolve_output_size_complex_split()
Target Function Names	convolve_output_size_real() convolve_output_size_complex_split()

3.1.4 FFT

Algorithm Name	mc_fft(x)
Algorithm Description	Perform Fast Fourier Transform
Inputs	Vector x
Outputs	Vector y
Limitations	Real or Complex input data. Only complex output data, No FFT provided for size 2 which is not a power of 2. An algorithm to replace FFT of size 2 is provided later in this document.
MATLAB Filename	mc_fft.m
DLL Function Name	mc_fft_complex_split()
Target Function Names	fft_complex_split()

3.1.5 IFFT

Algorithm Name	mc_ifft(x)
Algorithm Description	Performs Inverse Fast Fourier Transform.
Inputs	Vector x
Outputs	Vector y
Limitations	Real or Complex input data. Only complex output data. No IFFT provided for size 2 which is not a power of 2. An algorithm to replace IFFT of size 2 is provided later in this document.
MATLAB Filename	mc_ifft.m
DLL Function Name	mc_ifft_complex_split()
Target Function Names	inverse_fft_complex_split()

3.2 Polyphase Filter Channelization Algorithms Implemented in SCP Testbed

Below is a brief description of the Polyphase Filter Channelization algorithms that were implemented in the SCP Testbed.

3.2.1 PFC Algorithm that Executes on 1 Node

Algorithm Name	mc_polyphase_whole_use_state(x,h,state) (This algorithm executes on 1 node of the Mercury AdapDEV architecture)
Algorithm Description	Prepend (vertically) the state to x and perform the Polyphase filter of x using the filter coefficients h.
Inputs	Matrix x, Matrix h, matrix state
Outputs	Matrix y
Limitations	Complex data only. The number of rows in state must be one less than the number of rows in h for correct Mercury convolution setup. The number of columns in x (channels) must equal the number of columns in h. The number of columns in x (channels) must be a power of 2, but not equal to 2. An algorithm to replace FFT of size 2 is provided later in this document.
MATLAB Filename	mc_polyphase_whole_use_state.m
DLL Function Name	mc_polyphase_output_size_complex_split()
Target Function Names	polyphase_output_size_complex_split()

3.2.2 PFC Algorithm that executes on Nodes 1,2,4,8

Algorithm Name	mc_polyphase_whole_nprocs_use_state(x,h,n,state) (This algorithm executes on nodes 1,2,4,8 of the Mercury AdapDEV architecture)
Algorithm Description	Prepend (vertically) the state to x and perform the Polyphase filter of x using filter coefficients h, on n nodes.
Inputs	Matrix x, Matrix h, matrix state, integer n
Outputs	Matrix y
Limitations	Complex data only. The number of rows in state must be one less than the number of rows in h for correct Mercury convolution setup. The number of columns in x (channels) must equal the number of columns in h. The number of columns in x (channels) must be a power of 2, but not equal to 2. An algorithm to replace FFT of size 2 is provided later in this document. The number of rows in x (channel length or cl=blocksize/ch; where ch is the number of channels and blocksize is the size of the current block of input data to be processed) must be divisible by n. The output data from this function consists of n matrices of dimension (ch,cl/n) vertically concatenated in a single matrix of dimension (ch*n,cl/n). To form the correct polyphase output format, the same n matrices of dimension (ch,cl/n) must be horizontally concatenated in a single matrix of dimension (ch,cl).
MATLAB Filename	mc_polyphase_whole_nprocs_use_state.m
DLL Function Name	mc_polyphase_output_size_complex_split()
Target Function Names	polyphase_output_size_complex_split()

3.2.3 Testing the PFC Algorithm

The single processor version and the multiprocessor version of the Polyphase Filter algorithm were tested in the **SCP_FDM.m** script, the same script in which the MATLAB version runs. This script writes the output of the Polyphase Filter algorithm into files (FDM.1.01, FDM.1.02, …, FDM.1.32 for 32 channels) in the Temp directory. A binary difference (using the UNIX cmp -lx command in a MKS Kornshell) was performed on the 32 4MB files produced by the MATLAB version and the files produced by the Mercury version, using the standard 256 MB input file

(wide95_5.bin). In these 32 4MB files there were 319 differences found, and each of these differences were a value of one least significant bit. These differences are attributed to the use of 64 bit double operations in MATLAB algorithm and 32 bit floating point operations in the Mercury algorithm. The results were the same for both the single processor and the multiprocessor versions of the algorithm. An audio test was also performed in order to test this algorithm. This test involved using the Modulation function in the SCP MATLAB application to select a channel (e.g. 7) which contained an audio signal as FM, and to process it, thus producing an audio file (e.g. Audio.1.07) in the Audio directory. The channel was chosen after using the SCP Adjust function and View option to observe which audio signals were present in various channels. The resulting file (Audio.1.07) was played using a Windows media player program, and was found to be an audio signal from a FM radio station, as expected.

3.3 Perfect Reconstruction Algorithms Implemented in the SCP Testbed

Below is a brief description of the Perfect Reconstruction algorithms that were implemented in the SCP Testbed.

3.3.1 Perfect Reconstruction Algorithm that Executes on 1 Node

Algorithm Name	mc_perfect_reconstruction_whole_use_state(y,h,state)
Algorithm Description	Prepend (horizontally) the state to y and perform the perfect reconstruction of y using filter coefficients h.
Inputs	Matrix y, Matrix h, matrix state
Outputs	Matrix x
Limitations	Complex data only. The number of columns in state must be one less than the number of rows in h for correct Mercury convolution setup. The number of rows in y (channesl) must equal the number of columns in h. The number of rows in y (channels) must be a power of 2, but not equal to 2. An algorithm to replace FFT of size 2 is provided later in this document.
MATLAB Filename	mc_perfect_reconstruction_whole_use_ state.m
DLL Function Name	mc_perfect_reconstruction_output_size_complex_split( )
Target Function Names	perfect_reconstruction_output_size_complex_split()

3.3.2 Perfect Reconstruction Algorithm Executes on Nodes 1,2,4,8

Algorithm Name	mc_perfect_reconstruction_whole_nprocs_use_state (x,h,n,state)
Algorithm Description	Prepend (horizontally) the state to y and perform the perfect reconstruction of y using filter coefficients h, on n nodes.
Inputs	Matrix y, Matrix h, matrix state, integer n
Outputs	Matrix x
Limitations	Complex data only. The number of columns in state must be one less than the number of rows in h for correct Mercury convolution setup. The number of rows in y (channels) must equal the number of columns in h. The number of rows in y (channels) must be a power of 2, but not equal to 2. An algorithm to replace FFT of size 2 is provided later in this document. The number of columns in y (channel length or cl=blocksize/ch where ch is number of channels and blocksize is the size of the current block of input data to be processed) must be divisible by n. The output data from this function consists of n matrices of dimension (cl/n,ch) vertically concatenated in a single matrix of dimension (cl,ch). This matrix is in the correct perfect reconstruction output format.
MATLAB Filename	mc_perfect_reconstruction_whole_nprocs_use_state.m
DLL Function Name	mc_perfect_reconstruction_nprocs_output_size_complex_split()
Target Function Names	perfect_reconstruction_output_size_complex_split()

3.3.3 Testing the Perfect Reconstruction Algorithm

The single processor version and the multiprocessor version of the Perfect Reconstruction algorithm were tested in the **SCP_PR.m** script, the same script in which the MATLAB version runs. This script writes the output of the Perfect Reconstruction algorithm into a file (PR.1.32 for 32 channels) in the Input directory.

A binary difference (using the UNIX cmp -lx command in a MKS Kornshell) was performed on this 128 MB file produced by the MATLAB version and the file produced by the Mercury version, using the standard 256 MB input file (wide95_5.bin). In these 128 MB files there were 1722 differences found, and each of these differences was a value of one least significant bit. These differences are attributed to the use of 64 bit double operations in MATLAB algorithm and 32 bit floating point operations in the Mercury algorithm. The results were the same for both the single processor and the multiprocessor versions of the algorithm. An audio test was also performed in order to test this algorithm. This test involved first running Polyphase Filter with 128 channels, Bandwidth of 40,000, Rejection of 90, and offset %BW of 50. Then the SCP Adjust function and View option were used to find in which channels an audio signal existed. Then Perfect Reconstruction was used to reconstruct those channels (e.g. 25, 26 27 28) needed to include one of the audio signals, using FM in the Modulation field in the SCP PR function. The resulting file (Audio.5.28 in the Audio directory) was played using a Windows media player program, and was found to be an audio signal from a FM radio station, as expected.

3.4 Results Obtained in Testing and Validating SCP Testbed

3.4.1 Direct Memory Access (DMA) Transfer Times

The DMA transfer times from the host to the CE on the SCP hardware architecture were found to be longer than expected. Test programs were developed to measure various DMA transfer times for the AdapDev Mercury hardware. A host test program was created and can be located on the AFRL AdapDEV system in the directory called:

D:/Projects/SignalCollectionProcessingEnhancements/PolyphaseFilter/welchons/dx_test/host/dx_test.c

To call this test program the following commands need to be used:

./dx_test –src 1 –dst 2 –b
./dx_test –src 1 –dst 2 –d
./dx_test –src 1 –dst 2 -memcpy
./dx_test –src 2 –dst 1 -b
./dx_test –src 2 –dst 1 –d

A target test program was also created. The target test program is located in:

D:/Projects/SignalCollectionProcessingEnhancements/PolyphaseFilter/welchons/dx_test/target/dx_test.c

To call this program the following commands need to be used:

runmc –ce 2 dx_test –src 1 –dst 2
runmc –ce 2 dx_test –src 1 –dst 2 –b
runmc –ce 2 dx_test –src 1 –dst 2 –d

runmc –ce 2 dx_test –src 1 –dst 2 –memcpy
runmc –ce 2 dx_test –src 2 –dst 1 –b
runmc –ce 2 dx_test –src 2 –dst 1
runmc –ce 2 dx_test –src 2 –dst 3

Data collected with these programs showed that the DMA transfer times could be reduced by controlling the DMA transfer (i.e. executing the dx_ operations) from the CE rather than from the host. Table 1 below shows that a data transfer rate using a test program for a host-controlled DMA transfer from host to Mercury node 2 was 75 MB/sec. But if the DMA transfer is controlled by a Mercury node (Program Executing on CE 2), the transfer rate is 191 MB/sec. So an improvement was achieved by controlling the DMA transfer from the Mercury node instead of from the host.

Program Executing on CE	Source CE	Destination CE	Use Bridge Engine	Use Destination Engine	Use memcpy	Thruput (MB/sec)
host	host	2	Yes			75
host	host	2		Yes		83
host	host	2			Yes	21
host	2	host	Yes			150
host	2	host		Yes		144
2	host	2				191
2	host	2	Yes			93
2	host	2		Yes		93
2	host	2			Yes	4
2	2	host	Yes			184
2	2	host				226
2	2	3				253

Table 1 DMA Transfer Timing Data

3.4.2 Algorithm Execution Times

Data was collected for the following:

- The total execution time within the MATLAB environment for the MATLAB version of the PFC and Perfect Reconstruction Algorithms and the Mercury versions of the PFC and Perfect Reconstruction algorithms.
- Timing of the Mercury algorithm portion of the PFC and Perfect Reconstruction algorithms using TATL.
- Data was also collected for a budget containing the parts of the Mercury algorithm of the two algorithms. Figure 4 through Figure 8 presents the results of this timing analysis.

Performance of Polyphase Filter Algorithm, 4096-Tap Filter,

	Processor ID	32 Channel	8 Channel	4 Channel
Host	0	164.375	39.668	
CE	1	142.594	33	
CE	2	111.344	27.172	
CE	4	95.776	23.375	
CE	8	87.938	21.359	

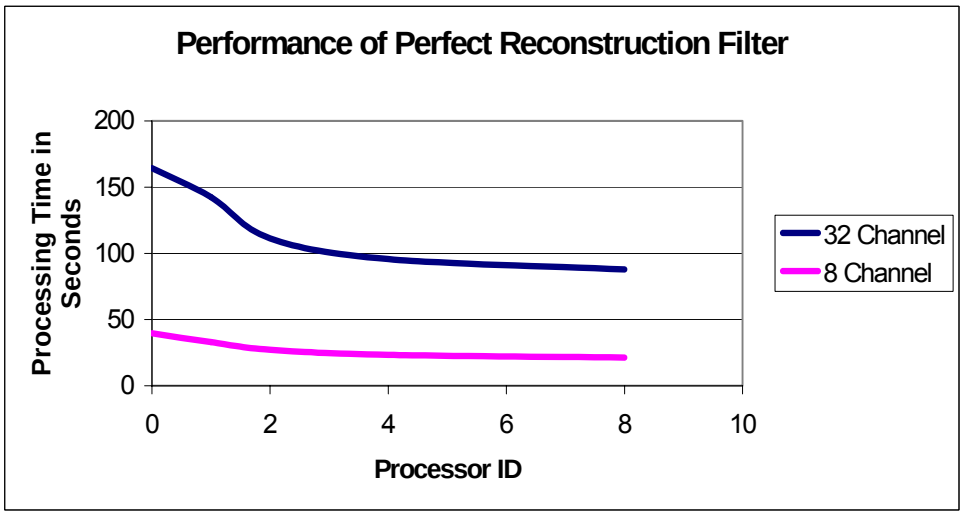


Figure 4 Performance of Perfect Reconstruction Filter

Performance of Polyphase Filter Algorithm, 1024-Tap Filter,

	Processor ID	32 Channel	8 Channel	4 Channel
Host	0	100.187	23.672	11.875
CE	1	107.359	26.266	12.895
CE	2	93.11	22.765	11.265
CE	4	86.75	21.094	10.453
CE	8	83.672	20.312	

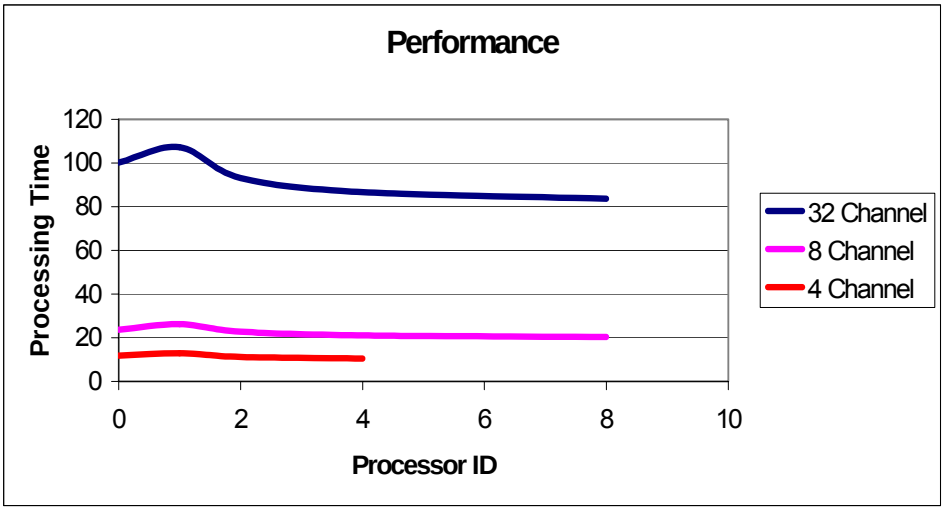


Figure 5 Performance of PFC Algorithm

Performance of Polyphase Filter Algorithm, 1024-Tap Filter, per Mbyte of Data

	Processor ID	32 Channel	8 Channel	4 Channel
Host	0	25.04675	1.4795	0.37109375
CE	1	26.83975	1.641625	0.40296875
CE	2	23.2775	1.4228125	0.35203125
CE	4	21.6875	1.318375	0.32665625
CE	8	20.918	1.2695	0

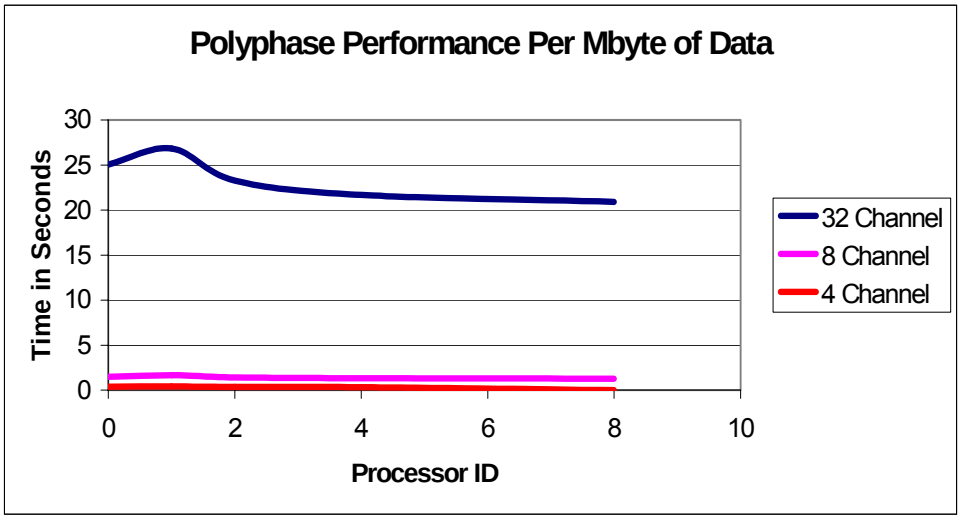


Figure 6 PFC Performance Per Mbyte of Data

Function		Transpose x	Filter	FFT
Filter Order	31	0.09788	0.839906	0.229875
Filter Order	255	0.098094	0.940969	0.230906
Filter Order	1023	0.100625	1.503344	0.231938
Filter Order	4095	0.099562	3.509281	0.2295

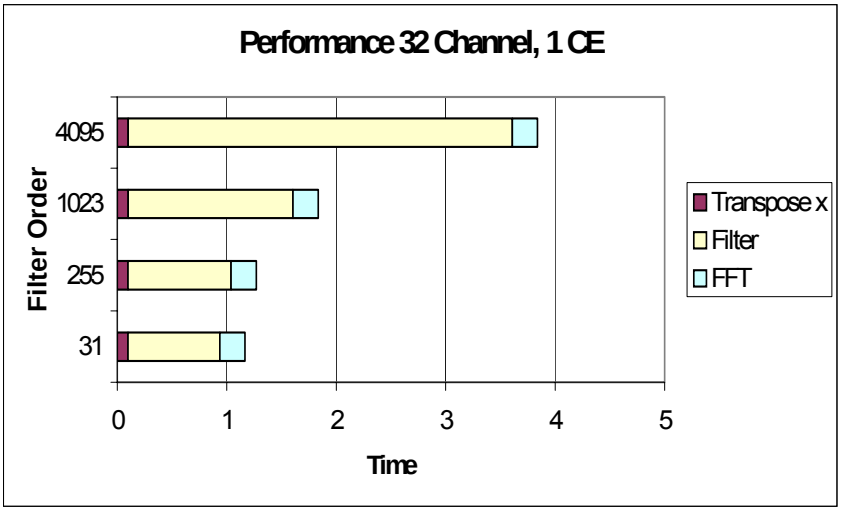


Figure 7 Performance of 32 Channel on 1 CE

Function		Transpose x	Filter	FFT
Filter Order	31	0.000485	0.777	0.171848
Filter Order	255	0.003333	0.987152	0.172394
Filter Order	1023	0.001394	1.526576	0.170848
Filter Order	4095	0.00097	3.47297	0.169485

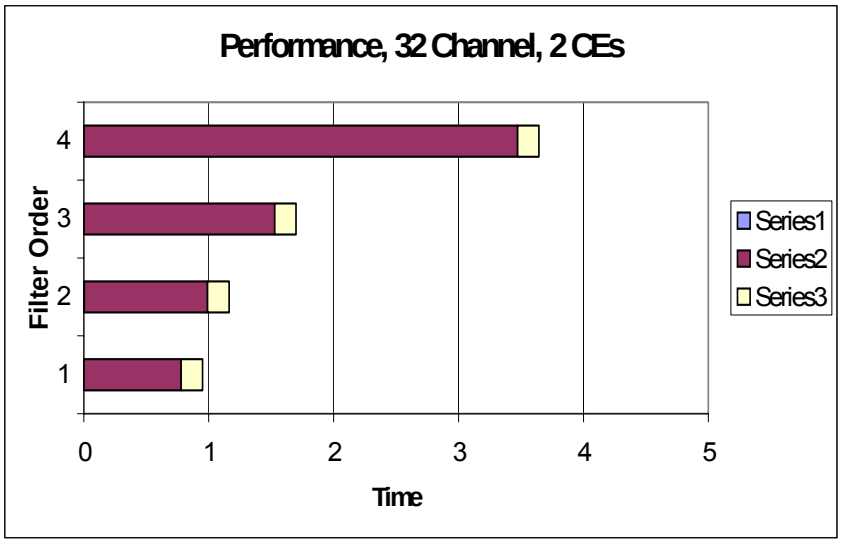


Figure 8 Performance of 32 Channel on 2 CE's

Table 2 through Table 29 below provide complete data of all the testing and timing performed on the PFC and Perfect Reconstruction algorithms in the SCP testbed.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	96.378	130.968	254.66	785.406
8	89.266	110.515	174.656	423.062
32	89.234	96.016	113.906	178.25
1024	N/A	N/A	File problem	File problem

Table 2 Polyphase Filter MATLAB **Version Total Time * (secs)**

*Total time includes file reads of input data, MATLAB preprocessing, file writes of output data, and all other operations involved with the execution of the PFC algorithm within the SCP testbed.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	116.938	136.921	205.48	484.094
8	118.281	126.969	161.578	301.343
32	122.062	116.86	127.1720	162.85
1024	N/A	N/A	File problem	File problem

Table 3 Polyphase Filter Mercury **Version** (1 CE) **Total Time * (secs)**

*Total time includes file reads of input data, MATLAB preprocessing, file writes of output data, and all other operations involved with the execution of the PFC algorithm within the SCP testbed.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	118.266	128.6720	162.313	301.688
8	120.547	124.7810	142.063	212.3590
32	127.625	119.297	124.437	142.204
1024	N/A	N/A	File problem	File problem

Table 4 Polyphase Filter Mercury **Version** (2 CE) **Total Time * (secs)**

*Total time includes file reads of input data, MATLAB preprocessing, file writes of output data, and all other operations involved with the execution of the PFC algorithm within the SCP testbed.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	114.453	118.984	136.421	206.609
8	116.907	118.625	127.437	162.468
32	124.516	115.046	117.281	126.312
1024	N/A	N/A	File problem	File problem

Table 5 Polyphase Filter Mercury **Version** (4 CE) **Total Time * (secs)**

*Total time includes file reads of input data, MATLAB preprocessing, file writes of output data, and all other operations involved with the execution of the PFC algorithm within the SCP testbed.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	115.063	116.203	125.0160	160.063
8	116.594	115.937	120.078	137.797
32	116.125	113.938	114.344	118.594
1024	N/A	N/A	File problem	File problem

Table 6 Polyphase Filter Mercury **Version** (8 CE) **Total Time * (secs)**

*Total time includes file reads of input data, MATLAB preprocessing, file writes of output data, and all other operations involved with the execution of the PFC algorithm within the SCP testbed.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	732.4	1366.96	3539.0	12243.5
8	690.4	991	2052.3	6427.15
32	648.093	703.6	987.2	2093.947
1024	N/A	N/A	File problem	File problem

Table 7 Polyphase Filter Mercury **Version** (1 CE) **Algorithm Time * (msecs)**

*Algorithm time includes the DMA transfer time of data to and from the Mercury processor(s) and the time to execute the algorithm on the Mercury processor(s).

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	419.8946	738.5985	1823.4221	6175.7
8	401.908	552.8	1092.284	3267.7
32	373.56287	408.679	543.5774	1102.132
1024	N/A	N/A	File problem	File problem

Table 8 Polyphase Filter Mercury **Version** (2 CE) **Algorithm Time * (msecs)**

*Algorithm time includes the DMA transfer time of data to and from the Mercury processor(s) and the time to execute the algorithm on the Mercury processor(s).

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	260.122	420.022	962.7876	3139.89
8	244.84255	323.3158	596.7749	1684.3634
32	231.28	251.11992	319.58142	599.1163
1024	N/A	N/A	File problem	File problem

Table 9 Polyphase Filter Mercury **Version** (4 CE) **Algorithm Time * (msecs)**

*Algorithm time includes the DMA transfer time of data to and from the Mercury processor(s) and the time to execute the algorithm on the Mercury processor(s).

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	206.35614	263.68594	536.7605	1627.7687
8	203.59406	216.65898	336.36124	902.92303
32	211.11421	211.41342	215.98969	359.09882
1024	N/A	N/A	File problem	File problem

Table 10 Polyphase Filter Mercury **Version** (8 CE) **Algorithm Time * (msecs)**

*Algorithm time includes the DMA transfer time of data to and from the Mercury processor(s) and the time to execute the algorithm on the Mercury processor(s).

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	8.1720	9.453	11.875	19.75
8	17.094	19.234	23.672	39.668
32	75.282	82.484	100.187	164.375
1024	N/A	N/A	File problem	File problem

Table 11 Perfect Reconstruction MATLAB **Version Total Time * (secs)**

*Total time includes file reads of input data, MATLAB preprocessing, file writes of output data, and all other operations involved with the execution of the Perfect Reconstruction algorithm within the SCP testbed.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	10.890	12.281	12.895	17.172
8	21.937	24.781	26.266	33.0
32	94.219	103.078	107.359	142.594
1024	N/A	N/A	File problem	File problem

Table 12 Perfect Reconstruction Mercury **Version** (1 CE) **Total Time * (secs)**

*Total time includes file reads of input data, MATLAB preprocessing, file writes of output data, and all other operations involved with the execution of the Perfect Reconstruction algorithm within the SCP testbed.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	10.0	10.672	11.265	13.531
8	20.172	21.516	22.765	27.172
32	83.453	88.391	93.11	111.344
1024	N/A	N/A	File problem	File problem

Table 13 Perfect Reconstruction Mercury **Version** (2 CE) **Total Time * (secs)**

*Total time includes file reads of input data, MATLAB preprocessing, file writes of output data, and all other operations involved with the execution of the Perfect Reconstruction algorithm within the SCP testbed.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	9.609	10.156	10.453	11.61
8	19.578	20.532	21.094	23.375
32	83.89 (Hang Problem?)	84.531	86.75	95.776
1024	N/A	N/A	File problem	File problem

Table 14 Perfect Reconstruction Mercury **Version** (4 CE) **Total Time * (secs)**

*Total time includes file reads of input data, MATLAB preprocessing, file writes of output data, and all other operations involved with the execution of the Perfect Reconstruction algorithm within the SCP testbed.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	Ch not div by 8	Ch not div by 8	Ch not div by 8	Ch not div by 8
8	20.094	20.141	20.312	21.359
32	83.063	83.828	83.672	87.938
1024	N/A	N/A	File problem	File problem

Table 15 Perfect Reconstruction Mercury **Version** (8 CE) **Total Time * (secs)**

*Total time includes file reads of input data, MATLAB preprocessing, file writes of output data, and all other operations involved with the execution of the Perfect Reconstruction algorithm within the SCP environment.

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	656.83936	659.4294	921.29205	2025.7165
8	654.138	672.62506	924.50104	2033.7358
32	604.5327	659.95154	900.8933	2008.3659
1024	N/A	N/A	File problem	File problem

Table 16 Perfect Reconstruction Mercury **Version** (1 CE) **Algorithm Time * (msecs)**

*Algorithm time includes the DMA transfer time of data to and from the Mercury processor(s) and the time to execute the algorithm on the Mercury processor(s).

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	406.26575	384.63223	518.59094	1073.3363
8	375.51764	376.75406	518.28503	1074.681
32	339.51083	371.33768	508.4174	1063.8549
1024	N/A	N/A	File problem	File problem

Table 17 Perfect Reconstruction Mercury **Version** (2 CE) **Algorithm Time * (msecs)**

*Algorithm time includes the DMA transfer time of data to and from the Mercury processor(s) and the time to execute the algorithm on the Mercury processor(s).

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	240.03444	244.0698	310.92053	587.6643
8	225.9357	239.8542	311.36688	590.1201
32	219.00354 (Hang Problem?)	84.531	86.75	95.776
1024	N/A	N/A	File problem	File problem

Table 18 Perfect Reconstruction Mercury Version (4 CE) Algorithm Time * (msecs)

*Algorithm time includes the DMA transfer time of data to and from the Mercury processor(s) and the time to execute the algorithm on the Mercury processor(s).

Channels	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
4	Ch not div by 8	Ch not div by 8	Ch not div by 8	Ch not div by 8
8	210.24318	206.85384	218.23032	359.0603
32	206.525	211.40442	215.2299	356.4724
1024	N/A	N/A	File problem	File problem

Table 19 Perfect Reconstruction Mercury **Version** (8 CE) **Algorithm Time * (msecs)**

*Algorithm time includes the DMA transfer time of data to and from the Mercury processor(s) and the time to execute the algorithm on the Mercury processor(s).

Function (Begin TATL event to End TATL event)	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
Input (211 to 205)	109.152	110.10462	109.76028	110.64474
Transpose x,h (230 to 231)	80.84124	70.7120	70.142334	67.67004
Convolution (238 to 239)	64.29414	128.559	403.86145	1510.5447
FFT (244 to 245)	281.91702	275.83356	275.62573	275.97324
Output (208 to 210)	117.21137	116.71398	116.89308	116.813934
Total	656.5247	705.7529	979.35004	2084.7283

Table 20 PFC Mercury Version (1 CE) Algorithm Budget Time (msecs) 32 Channels

*Algorithm budget describes the most time-consuming parts of the PFC Algorithm

Function (Begin TATL event to End TATL event)	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
Input (211 to 205)	55.7295	56.01918	55.96542	56.69412
Transpose x,h (230 to 231)	33.11094	34.1928	33.78888	33.83676
Convolution (238 to 239)	32.49054	64.368	201.9853	757.08606
FFT (244 to 245)	134.10785	137.01277	134.19113	137.26704
Output (208 to 210)	58.75326	59.0673	58.47024	58.87494
Total	372.209	408.98178	541.62366	1103.1333

Table 21 PFC Mercury Version (2 CE) Algorithm Budget Time * (msecs) 32 Channels

*Algorithm budget describes the most time-consuming parts of the Polyphase Algorithm

Function (Begin TATL event to End TATL event)	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
Input (211 to 205)	28.93956	28.92264	29.1342	29.62446
Transpose x,h (230 to 231)	16.08078	17.10198	16.94082	17.27568
Convolution (238 to 239)	13.88304	31.0626	100.65	378.06595
FFT (244 to 245)	57.57126	57.25434	56.76606	56.8074
Output (208 to 210)	28.21152	28.2768	28.1604	27.88608
Total	231.03456	250.36577	319.8816	599.12744

Table 22 PFC Mercury Version (4 CE) Algorithm Budget Time * (msecs) 32 Channels

*Algorithm budget describes the most time-consuming parts of the PFC Algorithm

Function (Begin TATL event to End TATL event)	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
Input (211 to 205)	15.74874	15.81588	16.00134	16.68894
Transpose x,h (230 to 231)	4.70196	5.31156	5.48238	5.60754
Convolution (238 to 239)	4.60326	13.74072	49.24782	188.98625
FFT (244 to 245)	20.08272	20.06334	19.94454	19.93182
Output (208 to 210)	96.812096**	86.749985**	45.68748**	12.84882
Total	210.97278	211.4253	215.98969	348.5262

Table 23 Polyphase Filter Mercury Version (8 CE) Algorithm Budget Time * (msecs) 32 Channels

*Algorithm budget describes the most time-consuming parts of the PFC Algorithm
**Part of output time consists of idle time after this process complete, while waiting for other processes to start

Function (Begin TATL event to End TATL event)	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
Input (211 to 205)	110.165344	109.31214	109.32168	109.961586
FFT (244 to 245)	247.2159	237.30354	203.92769	201.5628
Transpose h (258 to 259)	0.00762	0.02088	0.06246	0.20124
Convolution (238 to 239)	63.70476	128.26392	403.74344	1511.9479
Transpose x (262 to 263)	66.13698	65.94996	66.1194	66.32772
Output (208 to 210)	114.88128	114.040016	114.42007	114.4629
Total	605.2166	657.99207	900.65045	2007.5855

Table 24 Perfect Reconstruction Mercury Version (1 CE) Algorithm Budget Time * (msecs) 32 Channels

*Algorithm budget describes the most time-consuming parts of the Perfect Reconstruction Algorithm

Function (Begin TATL event to End TATL event)	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
Input (211 to 205)	56.30286	56.17656	56.45154	56.82426
FFT (244 to 245)	101.8428	102.49386	100.9551	101.04906
Transpose h (258 to 259)	0.00918	0.01986	0.0657	0.27228
Convolution (238 to 239)	32.49744	64.08978	201.83418	756.3664
Transpose x (262 to 263)	32.8497	32.802	32.9082	32.69514
Output (208 to 210)	57.52578	56.61828	56.09508	56.54256
Total	339.51083	371.22803	508.4174	1063.814

Table 25 Perfect Reconstruction Mercury Version (2 CE) Algorithm Budget Time * (msecs) 32 Channels

*Algorithm budget describes the most time-consuming parts of the Perfect Reconstruction Algorithm

Function (Begin TATL event to End TATL event)	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
Input (211 to 205)	29.259	29.26206	29.23212	33.498
FFT (244 to 245)	44.0283	43.7508	43.10766	43.50552
Transpose h (258 to 259)	0.00882	0.02160	0.07098	0.23352
Convolution (238 to 239)	14.3661	31.52622	100.43334	377.81304
Transpose x (262 to 263)	14.96694	15.10206	14.70084	14.83458
Output (208 to 210)	30.67092	28.01496	28.00374	27.9606
Total	219.20334	235.77103	304.6032	583.32574

Table 26 Perfect Reconstruction Mercury Version (4 CE) Algorithm Budget Time * (msecs) 32 Channels

*Algorithm budget describes the most time-consuming parts of the Perfect Reconstruction Algorithm

Function (Begin TATL event to End TATL event)	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
Input (211 to 205)	16.11174	16.14396	16.19898	16.88874
FFT (244 to 245)	15.51576	15.95568	15.7386	15.91974
Transpose h (258 to 259)	0.00792	0.01938	0.06228	0.29862
Convolution (238 to 239)	4.61118	13.39566	49.19472	188.98362
Transpose x (262 to 263)	5.12784	5.42952	5.15022	5.39394
Output (208 to 210)	97.604706**	93.13361**	55.8024**	13.01532
Total	206.52493	212.15773	200.23405	342.33682

Table 27 Perfect Reconstruction Mercury Version (8 CE) Algorithm Budget Time * (msecs) 32 Channels

*Algorithm budget describes the most time-consuming parts of the Perfect Reconstruction Algorithm

**Part of output time consists of idle time after this process complete, while waiting for other processes to start

Function	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
Transpose x	0.09788	0.098094	0.100625	0.099562
Filter	0.839906	0.940969	1.503344	3.509281
FFT	0.229875	0.230906	0.231938	0.229500

Table 28 Polyphase Filter Matlab Version Algorithm Budget Time * (secs) 32 Channels

*Algorithm budget describes the most time-consuming parts of the PFC Algorithm

Function	Filter Order 31	Filter Order 255	Filter Order 1023	Filter Order 4095
FFT	0.171848	0.172394	0.170848	0.169485
Filter	0.777000	0.987152	1.526576	3.47297
Transpose x	0.000485	0.003333	0.001394	0.00097

Table 29 Perfect Reconstruction Matlab **Version Algorithm Budget Time * (secs)** 32 Channels

*Algorithm budget describes the most time-consuming parts of the Perfect Reconstruction Algorithm

3.4.3 DMA Corruption Status

During testing of the multiprocessor versions of the PFC and Perfect Reconstruction algorithms, the results were occasionally found to be incorrect. Sometimes they matched the single processor outputs, but at other times they did not. When the differences did occur they seemed to only occur for the data being sent back from CE 4 and CE 5. The differences were traced to the reception of bad input data on CE 4 and CE 5 as a result of DMA transfers. The bad data appeared to be a mixture of the current DMA transfer and the previous DMA transfer to that CE. This may be a hardware problem. The temperature in the lab area where the AdapDev system is located varies from 60's F to 80's F. This may be a factor, although no direct correlation between the temperatures and the errors has been determined at this time. A test program was used to confirm this problem. The test program performs 5 DMA transfers of 4 MBytes each (1048576 longs). In the program the destination buffer is zeroed, and then an incrementing pattern is used on each transfer, so that each transfer uses a pattern which starts at 1048576 greater than the previous transfer. When the errors occurred, the value in the destination buffer was 1048576 less than the expected value (or the value left there from the previous transfer). The test program that created to determine this problem is located on the AFRL SCP testbed hardware in a directory called:

D:/Projects/SignalCollectionProcessingEnhancements/PolyphaseFilter/welchons/dx_test/host/dx_test.c

To call this program the commands is:

./dx_test –src 1 –dst 4 –b

This test program confirmed that intermittent errors do occur on CE 4 (-dst 4) and CE 5 (-dst 5) , but no errors were observed on any other CEs (-dst 2, -dst 3, etc.). The test program also confirmed that when the intermittent error occurs, it only occurs for DMA and not for Programmed IO (PIO). PIO is selected by using the option –memcpy instead of the option –b:

./dx_test –src 1 –dst 4 –memcpy

Mercury Customer Support (support@mc.com) was contacted and a case 0362397 was created. The Mercury representative, Emilio Velilla, suggested several actions to better diagnose the problem. Both boards were moved to different PCI slots. The problem still intermittently occurred on CE 4 and CE 5. The Mercury representative suggested using the command sysmc -ce 2 -l2pe=1 -bcs=0 3-9. The problem still occurred as before. The Mercury representative then suggested switching the PCI slots of the 2 boards. This resulted in the problem occurring on CE 8 and CE 9

instead of CE 4 and CE 5. Since switching the boards' results in a renumbering of the CEs this seemed to indicate that the problem still was occurring on the same board. Mercury Customer Support was informed of this result. No response has been received at this time.

3.4.4 FFT Problem for size 2 and non-power of 2

It was discovered in validating the SCP testbed that the Mercury SAL FFT does not work for a size 2 and non-power of 2. As a result the current implementation of the testbed low-level FFT, PFC, and Perfect Reconstruction does not work correctly for size 2 and non-power of 2 FFT. An algorithm has been developed for size 2 FFT. This algorithm is shown below and could be implemented in future updates to the testbed. There is currently no fix for the non-power of 2 FFT problem.

One limitation found in the Mercury SAL library is that the Fast Fourier Transform (FFT) implementation does not handle the case of N=2.

The definition of the Discrete Fourier Transform (DFT) is

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \quad 0 \le k \le N-1$$

Where $W_N = e^{-j(2\pi/N)}$,

k = filter number,
N = total number of samples,
$x(n) = n^{\text{th}}$ sample in sequence of N samples,
$X(k) = k^{\text{th}}$ filter.

The Fast Fourier Transform is an efficient implementation of the DFT, and the outputs of either algorithm are identical. The derivation of an FFT algorithm achieves efficiency by recognizing that 1) sometimes the phase rotations W_N^{kn}, often called "twiddle factors", are sometimes simple additions (e.g., when n=0 or k=0) or subtractions (e.g., when k+n=N/2), and 2) symmetry and repetition in the phase rotations can be exploited in portions of the summations For large N, and when all filters (many k) are desired, careful formulation of the FFT can result in significant computational savings. However, for small N, or when a small percentage of filters (small number of k) are to be computed, the FFT ceases to yield computational advantages, and a simple brute-force formulation of the DFT is often preferred. Since a DFT for small N is simple to implement, it is understandable that the Mercury SAL library does not handle the case of N=2.

For the special case of $N = 2$, the DFT reduces to

$$X(k) = \sum_{n=0}^{1} x(n) e_1^{-j2\pi kn}, \quad k = 0,1$$

.

This can be further reduced to

$$X(0) = x(0)e^{-j2\pi(0)(0)/2} + x(1)e^{-j2\pi(0)(1)/2}$$
$$X(1) = x(0)e^{-j2\pi(1)(0)/2} + x(1)e^{-j2\pi(1)(1)/2}$$

and finally to

$$X(0) = x(0) + x(1)$$
$$X(1) = x(0) - x(1)$$

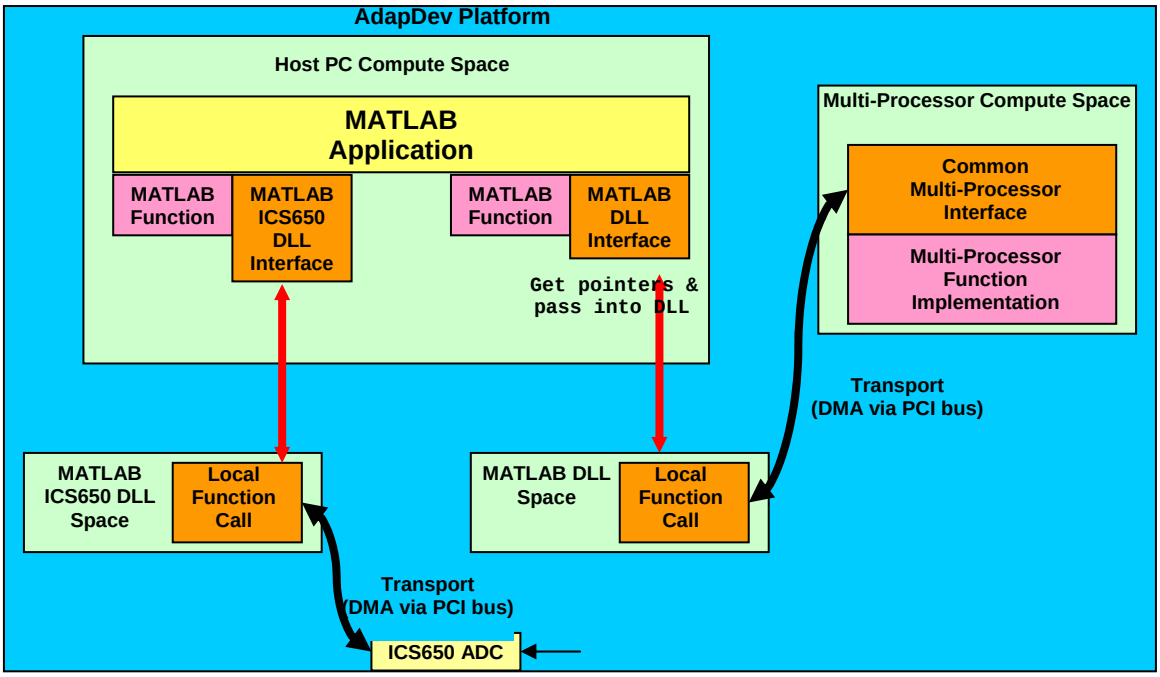
Besides keeping in mind that these two equations are performed on complex data, the implementation of a two point FFT (or DFT) is simple.

4.0 Developing a Streaming Data Capability

A library was purchased for interfacing from MATLAB to the ICS650 Analog-to Digital card. The MATLAB/ICS650 data collection/transfer was verified to be operational. The ICS650 operations are not yet integrated for streaming data within the testbed. The following steps will be required in the future.

1. The first step will be to transfer complex integer data (16 bit real part and 16 bit imaginary part) from the ICS650 to the host via DMA using the MATLAB ICS650 library. That data will then be converted to floating point and transferred to the Mercury DMA using the same mechanism currently used for executing the PFC algorithm within the testbed. This method focuses on only the ICS650 addition with no other changes. This method is shown in Figure 9 below. Once this method is verified, then an incremental change can be made to send the 16 bit complex data directly to the Mercury board without any conversion to floating point. This change will require changes to the matlab-to-mercury interface to handle the 16 bit data.

MATLAB, ADC, & Data Streaming
(1st step)



Call ICS650 Function from MATLAB on PC for ADC data collection
Call MP Function from MATLAB on PC for AdapDev processing

Figure 9 Step 1 for Developing a Streaming Data Capability

2. The second step will be to transfer the complex integer data (16 bit real part and 16 bit imaginary part) from the ICS650 directory to a Mercury CE via DMA using the MATLAB ICS650 library. This step will require modifications to the ICS DLL. It will also require modifications to the MATLAB-to-Mercury interface, since the Mercury side of the interface will now have to handle the receipt of the data directly from the ICS650 rather from the host side of the MATLAB-to-Mercury interface. This approach is shown in Figure 10 below.

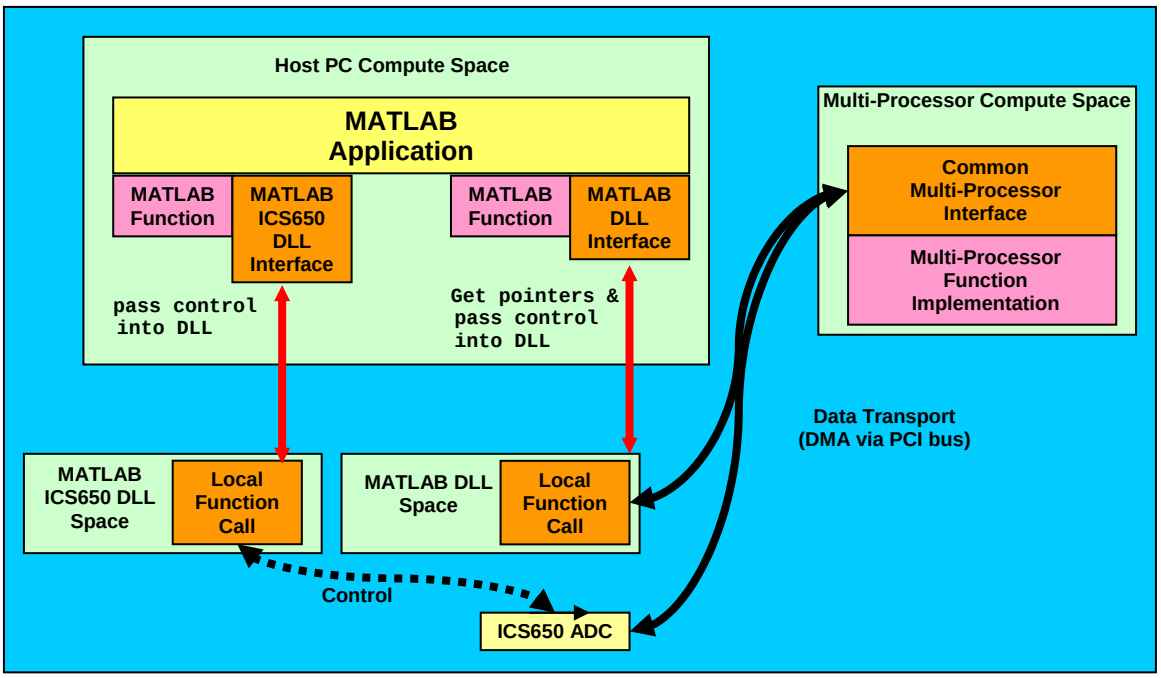


Figure 10 Step 2 for Developing a Streaming Data Capability

5.0 Procedures

5.1 Important Mercury Procedures

This section describes important procedures for working with the Mercury AdapDEV hardware architecture.

5.1.1 Configmc

The configmc command initializes the Mercury hardware in the system using a configuration text file which describes the hardware and connections of the Mercury system. It also creates a configuration database containing that configuration information and initializes the Mercury Operating System on the host by loading it with the configuration database. The hardware and database initializations together initialize the multicomputer. The hardware initialization leaves all CEs that configmc initialized reset with their memory cleared. This function can be executed from the command line in a MKS KornShell as:

configmc –cf C:/MercurySoftware/AdapDev/AdapDev_welchons.cfg init

For more information see the Mercury System Manager' Guide, Chapter 8 Creating Configuration Files (section Using configmc to Configure a Multicomputer), Chapter 15, Overview of System Startup (sections Starting a Multicompter).

This function can be executed from a host program by calling sys_configmc(). For more information see Chapter 6 Interprocessor Communication System (ICS) Functions in the MCOE API Reference Manual. Below is an example of executing configmc from a program.

```
char dash_cf_str[] = "-cf";
char config_file[] =

"C:/MercurySoftware/AdapDev/AdapDev_welchons.cfg";
char config_init[] = "init";
char *argv_config_init_strings[4] =  {dash_cf_str,
                                      config_file,
                                      config_init, 0};

rc = sys_configmc(argv_config_init_strings, 0);
```

5.1.2 Sysmc

The sysmc command loads and runs the Mercury MCOE executive on the target CEs to boot the CEs. This function can be executed from the command line in a MKS KornShell as:

```
sysmc –ce 2 –bcs=0 init 3-9
```

For more information see the Mercury System Manager's Guide, Chapter 15 Overview of System Startup (section Using sysmc to Load and Start MC/OS). This function can be executed from a host program by calling sys_sysmc(). For more information see Chapter 6 Interprocessor Communication System (ICS) Functions in the MCOE API Reference Manual. Below is an example of executing sysmc from a program.

```
char dash_ce_str[] =   "-ce";
char sys_name_server[] = “2”;
char sys_console_server[] = "-bcs=0";
char sys_init[] =  "init";
char sys_ces[] = “3-9”;
char *argv_sys_init_strings[6] = {dash_ce_str,
                                  sys_name_server,
                                  sys_console_server,
                                  sys_init,
                                  sys_ces,                    0};
```

```
rc = sys_sysmc(argv_sys_init_strings);
```

5.1.3 Runmc

The runmc command loads and starts a program running on a CE. This function can be executed from the command line in a MKS KornShell as:

```
runmc –ce 2 hello
```

For more information see MCOE Development Tools and Files Guide (Chapter 5 Development Tools Summary under Referencing MC/OS Development Tools) and the MCOE Developer's Guide (Chapter 5 Creating and Managing Processes under Creating Processes with runmc and sys_runmc). This function can be executed from a host program by calling sys_runmc(). For more information see Chapter 6 Interprocessor Communication System (ICS) Functions in the MCOE API Reference Manual.

5.1.4 Image_load() and Proc_spawn()

The image-load() and proc_spawn() are two functions that provide the same basic capability as sys_runmc(), but with more options for the user to customize the host's loading and running of the program on a CE. For more information see Chapter 6 Interprocessor Communication System (ICS) Functions in the MCOE API Reference Manual. Below is an example of executing image_load and proc_spawn from a program.

```
rc = image_load(&iid, 2, image, 0);
if (rc != CE_SUCCESS)
{
   err_string(rc,&err_str,0,ERR_GET_ALL);
   fprintf(fp,"mc_init() : %s\n",err_str);
   fatal(rc,"mc_init() - image_load() failed");
}
/* Request 64 MB of heap for malloc's done on the target. */
ps_runtime.heapsize = 64*1024*1024;
rc = proc_spawn(&pid, iid, PRCM_DISOWN, &ps_runtime, NULL,
0, NULL);
if (rc != CE_SUCCESS)
{
   err_string(rc,&err_str,0,ERR_GET_ALL);
   fprintf(fp,"mc_init() : %s\n",err_str);
   fatal(rc,"mc_init() - proc_spawn() failed");
}
```

5.1.5 Scanpcibus

This scanpcibus function provides the ability to show the bus and device numbers on the PCI bus. These values are used to describe motherboards on the board line of the config text file used by configmc. This function provides a method to determine the bus and device number of the Mercury boards. This function is executed by using Windows Explorer and double-clicking on the scanpcibus icon in the folder **C:/MercurySoftware/mercury/install/winnt-5_0-ix86**. A single-click is done on the + sign next to a bus. Then a double-click is performed on a device. A window of information will appear similar to the one in Figure 11. The key piece of information is the Vendor ID. If the Vendor ID is 0x1134, the device is Mercury hardware. For more information see Chapter 2 Installing Hardware Guide (specifically Installing Motherboards) in the Mercury System Manager's Guide.

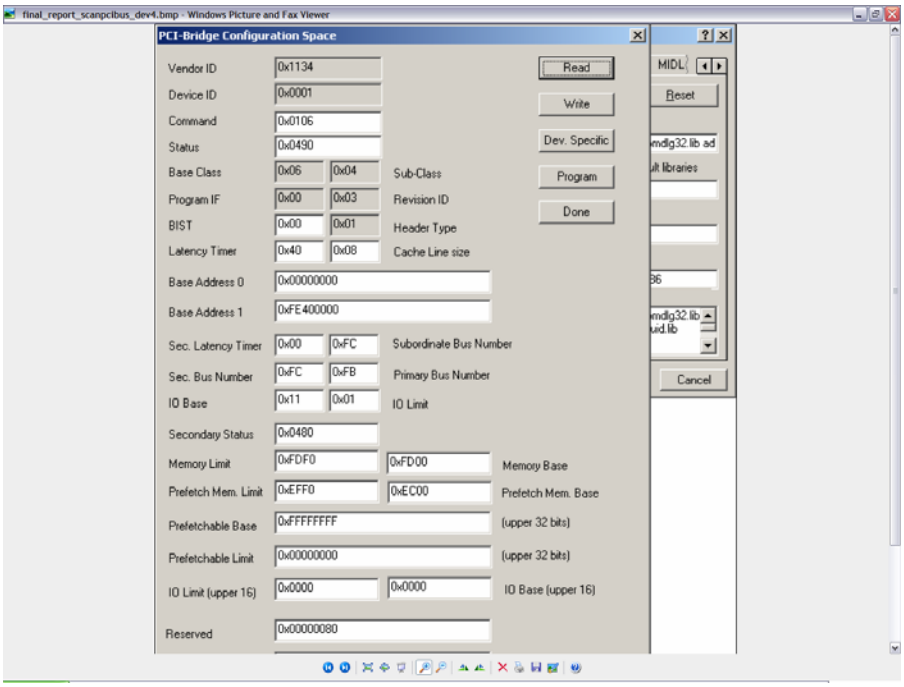


Figure 11 Scanpcibus output for a Mercury motherboard

5.1.6 Shared Memory Buffer (SMB):

This is the method of specifying endpoints for transfers between CEs in a Mercury system. The function smb_create is used to allocate physical memory on CE (determined by an argument of CE A to smb_create) which can serve as an endpoint accessible by other CEs. Data can be transferred to or from that SMB via Programmed IO (PIO) or DMA. PIO occurs when data is moved to or from a SMB via the action of a program, such as a call to memcpy(). In order for CE B to access a SMB created on CE A via PIO, two steps are necessary. The function smb_attach() must be executed by CE B for that SMB. Then smb_map() must be executed by CE

B to make the SMB accessible in the process's virtual address space. So PIO can occur between two SMBs mapped into a process's virtual address space, or between a local address and a SMB mapped into the processes virtual address space. DMA transfers can only occur between two SMBs. Typically one SMB is created on the local CE and one is attached (via smb_attach) on a remote CE. The SMBs do not need to be mapped into the process's virtual address space, unless the program has to write to or read from one of the SMBs. For more information see the MC/OS Developer's Guide (Chapter 12 Using Shared Memory Buffers (SMBs)). Sample SMB code is below which shows attaching to a remote SMB and creating and mapping a local SMB.

```
rc = smb_attach(master_data_smb_name,
      SMB_GLOBAL,
      SMB_MAX_SIZE,
      0,
      &smb_size,
      &master_data_smb_handle);
if (rc) {
   fatal(rc, "smb_attach() master_data_smb failed");
}

rc = smb_create(slave_data_smb_name,
      SMB_CREATE | SMB_GLOBAL,
      SMB_MAX_SIZE,
      MEM_CE_PHYS,
      my_ceid,
      0,
      &slave_data_smb_handle);

if (rc) {
   fatal(rc, "smb_create() slave_data_smb failed");
}

rc = smb_map(slave_data_smb_handle,
      0, 0,
      SMB_MAX_SIZE,
      0, 0, 0,
      &slave_data_smb_map_handle,
      &slave_data_smb_map_info);
if (rc) {
   fatal(rc, "smb_map() slave_data_smb failed");
}

slave_data_smb_bufp = slave_data_smb_map_info.process.vaddr;
```

5.1.7 DMA transfers (DX)

A DMA transfer can be used to move data between two SMBs on different CEs. The endpoint SMBs are specified using the dx_create() call which creates a transfer attribute object. A transfer request is made by a call to dx_copy() which specifies the size of the transfer. The transfer is actually queued (i.e. started as soon as any other active transfers complete) by dx_start(). Then dx_get_status() can be used to check on the completion status of DMA transfers which have been started asynchronously. When all transfers for the transfer request are complete, the transfer can be released by dx_release(). The transfer attribute object can be removed by dx_destroy(). For more information see the MC/OS Developer's Guide (Chapter 13 Using DX for Data Transfer). Sample DX code is below showing a DMA transfer from a remote CE to the local CE.

```
/*
 * Set up dx operations which will pull inputs
 * from master to slave.
 */
if (verbose) fprintf(fp,"%s before dx_create for pull\n",prog_name);
rc = dx_create(master_data_smb_handle,
       slave_data_smb_handle,
       0,0,0,0,
       0,
       &xfer_attr_pull);
if (rc) {
  fatal(rc,"dx_create for pull slave_data_smb_handle");
}

rc = dx_copy(xfer_attr_pull,
       0,0,
        buffer_size,
        0,0,
        DX_POLL,
        &xfer_req_pull);
if (rc) {
   fprintf(fp,"dx_copy i=%d data_type_inputs[%d]=%s\n",
            i,i,local_master_smb_cmd.data_type_inputs[i]);
   fatal(rc, "dx_copy error");
}

rc = dx_start(xfer_req_pull, DX_WAIT);
if (rc) {
   fprintf(fp,"dx_start i=%d data_type_inputs[%d]=%s\n",
            i,i,local_master_smb_cmd.data_type_inputs[i]);
   fatal(rc, "dx_start error");
}
```

```
rc = dx_release(xfer_req_pull, 0);
if (rc) {
   fprintf(fp,"dx_release i=%d \n",i);
   fatal(rc, "dx_release error");
}
```

5.1.8 POSIX Semaphores

When multiple processes require access to a shared object (e.g. SMB), some synchronization method is required to insure that only one process at a time is writing to the object. POSIX Semaphores provide this mechanism. A unique name represented by a character string is assigned for each semaphore to be created. Each process which will use this semaphore call sem_open() with an argument. Then in any process where a shared object is accessed, a call to sem_wait() is made before a write an object. A call to sem_post() is made after a write to an object. The call to sem_wait() will block while another process is in a state where sem_wait() has been called, but sem_post() has not been called. Sample code using semaphore operations is shown below. For more information see the MC/OS Developer's Guide (Chapter 17 Using POSIX Semaphores).

```
master_done_sem = sem_open (master_done_sem_name,
      O_CREAT,
      0,
      0);
if (master_done_sem == SEM_FAILED) {
   perror("slave: sem_open failure");
   fatal(rc, "master_done_sem = sem_open() failed");
}

rc = sem_wait(master_done_sem);
if (rc) {
   fatal(rc, "sem_wait master_done_sem failed");
}

rc = sem_post(master_done_sem);
if (rc) {
   fatal(rc, "master_done_sem failed");
}
```

5.2 Timing and Analysis Tool (TATL):

In a real-time multiprocessor system it is important to measure the timing of events and the relationship to those events on various processors. TATL provides this capability. This capability can be used on a host or target process. TATL_TRACE_ENABLE must be defined in order for tatl calls to be executed. It can be defined as -DTATL_TRACE_ENABLE in a target Makefile, or it can be added to the MULTI build

tool or the Microsoft Visual C++ build tool as a defined value. Below is sample code for setting up TATL in a host application program and a target application program. Also below is sample code for recording a user-defined event. For more information see TATL Developer's Guide, Chapter 3 Using the TATL API, Example: Basic TATL Setup section.

Host program setup:

```
/* TATL variables */
TATL_Trace_Id_T  trid;
TATL_Trace_Param trace_params = {0};

sprintf(trace_params.name,"host");
trace_params.size = TATL_TRACE_HEADER_SIZE +
(2000*TATL_TRACE_EVENT_SIZE);
trace_params.location = (char *) 0;
trace_params.options = TATL_CIRCULAR_Q;
rc = tatl_trace_create(&trace_params, &trid);
if (rc) {
   err_string(rc,&err_str,0,ERR_GET_ALL);
   fprintf(fp,"%s\n",err_str);
   fatal(rc,"mc_init() tatl_trace_create failed");
}
rc = tatl_trace_start(&trid, 1);
if (rc) {
   err_string(rc,&err_str,0,ERR_GET_ALL);
   fprintf(fp,"%s\n",err_str);
   fatal(rc,"mc_init() tatl_trace_start failed");
}
```

Target program setup:

```
/* TATL variables */
TATL_Trace_Id_T  trid;
TATL_Trace_Param trace_params = {0};

sprintf(trace_params.name,"slave_ce%ld",my_ceid);
trace_params.size = TATL_TRACE_HEADER_SIZE +
(1000*TATL_TRACE_EVENT_SIZE);
trace_params.location = (char *) 0;
trace_params.options = TATL_CIRCULAR_Q;

rc = tatl_trace_create(&trace_params, &trid);
if (rc) {
```

```
  err_string(rc,&err_str,0,ERR_GET_ALL);
  fprintf(fp,"%s\n",err_str);
    return -1;
}
rc = tatl_trace_start(&trid, 1);
if (rc) {
  err_string(rc,&err_str,0,ERR_GET_ALL);
  fprintf(fp,"%s\n",err_str);
    return -1;
}
```

Recording a user-defined event:

```
#define BEGIN_CE_RECEIVE_INPUTS    204

rc = tatl_trace_user_event(trid, BEGIN_CE_RECEIVE_INPUTS, 0,
0);
if (rc) {
   fatal(rc, "tatl_trace_user_event, BEGIN_CE_RECEIVE_INPUTS");
}
```

The initialization of the TATL time server must be performed before any of these previous functions are called. This initialization may be performed from a MKS KornShell as shown below where the CE specified must be the name server (CE 2 is the name server in this case).

```
tatl –ce 2 time_server_start
```

The TATL file can be dumped by the execution from a MKS KornShell as shown below where the CE specified must be the name server (CE 2 is the name server in this case).

```
tatl –ce 2 –df filename dump
```

Both these functions can also be executed from the host program. The TATL Developer's Guide indicates that these commands can be executed by using the sys_tatl() API (Chapter 3 Using the TATL API, Invoking the tatl Tool from within a Program). But problems were encountered in this implementation with sys_tatl. As a result the tatl commands were implemented using the system command as shown below:

```
char tatl_cmd[256];

sprintf(tatl_cmd,"tatl –ce 2 time_server_start”);
irc = system(tatl_cmd);
```

```
if (irc) {
   fatal(rc,"Error on system tatl time_server_start");
}

sprintf(tatl_cmd,"tatl –ce 2 –df tatl_dump_data.tat dump");
irc = system(tatl_cmd);
if (irc) {
   fatal(0,"mc_stop() Error on system tatl,dump");
}

sprintf(tatl_cmd,"tatl –ce 2 time_server_stop”);
irc = system(tatl_cmd);
if (irc) {
   fatal(0,"mc_stop() Error on system tatl,time_server_stop");
}
```

The timing data in the file can be analyzed using the Mercury tatlview tool. This tool can be invoked in a MKS KornShell window using the command tatlview. For more information on tatlview, see the TATL Developer’s Guide (Chapter 5 Using the TATL Viewer).

5.3 MULTI Builder Tool

Below in Figure 12 describing the procedure for building target application programs using the MULTI Builder, and for building a host application using the Microsoft Visual C++ Builder.

Programming Steps to Build a Multiprocessor Application within MATLAB

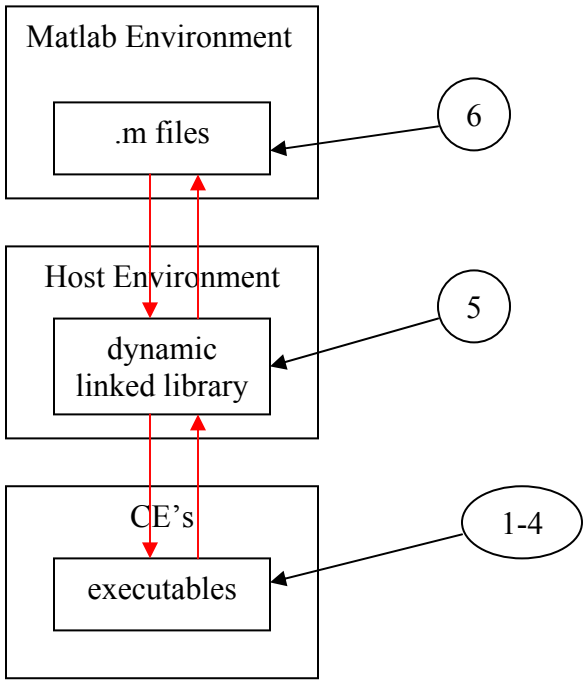


Figure 12 Steps for Building Multiprocessor Application

Program Operations:

Open: Loading Program
Select: Selection by mouse click, double-click or keyboard input
→ : Submenu
Enter: Keyboard input
Replace: Change text using keyboard or paste method

5.3.1 Step 1. Create Executables Using Multi Tool

Refer to Table 30 below for the steps required to create an executable using the Multi Tool.

Action	Result/Comments
1. Open Multi Program	Starts a GUI as shown in figure 2
2. Select "File" → "New Project" from Multi's Menu Bar	Opens a "New Project Wizard Window"
3. Select "Power PC" from Wizard Window, Processor Family List	Select the appropriate processor type.
4. Select "Generic Power PC" → "PowerPC 603" from Wizard Window, Board Name List	Select the appropriate platform
5. Select "Little Endian" from Wizard Window, Endian List	Selects between Little Endian and Big Endian Format
6. Press "Next" Button of Wizard Options	Change the window. Ask for programming language settings, project name, project type and directories.
7. Select "C" from Wizard Options Language Box	Set C language for compiling the project
8. Select "Empty Executable" from Wizard Options, Project Type Box	Set project as an empty project.
9. Enter *path* in Wizard Options, New Project Directory Box	Set project directory.
10. Enter *name* in Wizard Options, Project Name Box	Set project name.
11. Press "Next" Button of Wizard Options	Creates a new Defauld.bld project. This project contains default settings that can be modified by following steps 12-21
12. Select "Project" → "File Options" from Multi's Menu Bar	Opens a multi-tab window called "File Options"

Action	Result/Comments
Editing Mode: Press "Edit" Button of File Options	Change to edit mode. The user can type values as allowed.
View Mode: Press "View" Button of File Options	Change to view mode. User is not allowed to change settings. Default settings are visible.
15. Enter *preprocessor definitions* in File Options, General Tab, Defines Box	Any global #define variable needs to be define here (e.g. TATL_TRACE_ENABLE).
16. Enter *library name* in File Options, General Tab, Library Box	Any libraries must be specified here (e.g. libtatl.appc_le).
17. Enter *path* in File Options, General Tab, Source Directories Box	Any library path must be specified here.
18. Enter *name* in File Options, Action Tab, Output File Box	Optional output file name is specified here.
19. Enter *path* in File Options, Action Tab Object Directory Box	The path of the user's source code must be specified here.
20. Enter *path* in File Options, Advance Tab, Temporary Directory Box	A temporary directory must be specified here.
21. Press "Close" in File Options	Be sure that the user entries have been accepted and leave the File Option Window. Multi creates a "default.bld" object.
22. Select "default.bld", Press Mouse Right Button, Select	Multi must know about the name of the compiler to be used. See step 23.
23. Select "Mercury_PPC_GHS" from Build Target Selection Window	Enter the name of the compiler and close the window.

Table 30 Create Executables

5.3.2 Step 2. Add Source Code into Project

Refer to Table 31 below for the steps required to add source code into a project.

Action	Result/Comments
1. Press "Add" Button of Multi's Tool Bar	Loads a window that creates and saves a file object.
2. Enter *name* in Save Window, File Name Box	Enter the name of the file and add a valid extension: .c or .h
3. Press "Save"	Creates the source object in "default.bld"
4. Select *source file* from "default.bld"	Opens the editor for writing code. Enter code and save.
5. Press "Go Back" Button	Returns to "default.bld"

Table 31 Steps for Adding Source Code into a Project

5.3.3 Step 3. Compile Code

Refer to Table 32 below for the steps to compile code.

Action	Result/Comments
1. Press "Build" Button of Multi's Tool Bar	Compiles a source code. Source code has to be selected. Note: Multi tool cannot run the executables.

Table 32 Steps for Compiling Code

5.3.4 Step 4. Running Code

Refer to Table 33 for the steps on running code.

Action	Result/Comments
1. Open KornShell Console	Mercury Code can be run manually or programmatically. Using KornShell can be considered as a manual process.
2. Enter "config –cf *path\configuration_file_name* init"	Initializes the PCI-Boards using a configuration file. See Mercury documentation for details. Do not use quotation marks.
3. Enter "sysmc –ce 2 –bsc=0 int 2-9"	Initializes the CEs. CE 2 is selected as a server. CEs 2-9 are initialized. See configuration file of step 2 for CE numbering.
4. Enter "runmc –ce 2-9 *executable.ext*"	Loads and execute the program *executable.extension* in the indicated CEs.

Table 33 Steps for Running Code

5.3.5 Step 5. MS C++ Settings for Creating DLL to Run in Host

Refer to Table 34 for the steps required to MS C++ settings needed to create a DLL that will provide the capability to run in a host.

Action	Result/Comments
1. Open MicroSoft Visual C++	The following procedure will allow to edit, a dynamic linked library (DLL).
2. Select "File" → "New"	Opens the New Window
3. Enter *path* in New, Location Box	Set path of DLL.
4. Select "Win32 Dynamic-Linked Library" from New Project Window	Specify a DLL project.

Action	Result/Comments
5. Press “OK” Button	Moves to the next options in New Window.
6. Select “Empty Library”	Visual C++ offers other options.
7. Press “Finish”	Creates a project for the dynamic linked library
8. Select “Project” → “Project Settings” from Visual C++ Menu Bar	Opens the Project Settings Window
9. Enter *program* in Project Settings Window, Debug Tab, Executable for Debug Section Box	Supply the path and name of the program that is going to be used for debugging. (e.g. C:/MATLAB6/bin/win32/matlab.exe)
10. Replace “/MT_d” with “/MD” Option in Project Settings Window, C/C++ Tab, Category set to Preprocessor, Project Options Box	Required setting. Will generate warnings after compilation if it is not set properly.
10.1 Add MC_HOST, TATL_TRACE_ENABLE defines in Project Settings Window, C/C++ Tab, Category set to Preprocessor, Project Definitions Box	Add user-specific defines
10.2 Add path to *include* files in Project Settings Window, C/C++ Tab, Category set to Preprocessor, Include box	Visual C++ must know location of the Mercury includes, such as mcos.h. (e.g. C:/MercurySoftware/mercury/include) (can replace C:/MercurySoftware with $MC_ROOT_DIR)
11. Enter *libraries* in Project Settings Window, Link Tab, Category set to General, Object Library Modules Box	Visual C++ must know the names and location of the Mercury Libraries. (e.g. libmc.lib wsock32.lib setupapi.lib libtatl.lib (if using tatl))
12. Enter *libraries paths* in Project Settings Window, Link Tab, Category set to Input, Additional	Visual C++ must know the names and location of the Mercury Libraries. (e.g. C:/MercurySoftware/mercury/lib/winnt-

Action	Result/Comments
Library Path Box	5_0-ix86) (can replace C:/MercurySoftware with $MC_ROOT_DIR)
13. Edit and Compile	DLL may be loaded to Matlab or C applications.

Table 34 MS C++ Settings for Creating DLL to Run in Host

6.0 Testbed Host-to-CE Remote Procedure Call Design Issues

6.1 Shared Memory Buffers

The Host-to-CE Remote Procedure Call capability is in the host file master_mc_rpc.c (the host process is the master) and the target file slave_mc_rpc.c (the target process is the slave). The communication between the host and target utilizes three shared memory buffers (SMBs) and 2 POSIX semaphores. The host function mc_rpc_init() calls init_ipc() which creates the SMBs on the host. The target function ipc_init() creates the SMBs on the target. The first SMB contains the command to be sent from master to slave. This command informs the slave of the type of command (i.e. algorithm) to be executed on the slave, and the number, size, and type of the inputs and outputs. The inputs and outputs are not sent in the command. The command establishes the input and output data for both the master and the slave, so that both sides know how many transfers will be performed to move the inputs to the slave and to move the outputs back to the master. This command SMB is created on the master (host) side and attached-to on the slave side. The second SMB contains master-side data. This SMB can contain algorithm input data which is to be sent from the master to the slave, or algorithm output data which has been received back from the slave. The data transfer method is PIO for small data sizes or DMA for large data sizes. This master-data SMB is created on the master side and attached to on the slave side. The third SMB contains slave-side data. This SMB can contain algorithm input data which has been received by the slave side, or algorithm output data which is to be sent back to the master. This SMB is created on the slave side and attached to on the master side. The host function mc_rpc_call() implements a synchronous (program execution waits until the outputs are received back before continuing) remote procedure call. This remote procedure call causes the transfer of input data to the target, the execution of the algorithm on the target, and the transfer of the data back from the target to the host. The host function mc_rpc_call_nowait() implements an asynchronous (program execution continues after the command and input data is sent to the slave, but before the output data is received back) remote procedure call. The host function mc_rpc_wait_for_results() is used in conjunction with mc_rpc_call_nowait() to test when the outputs are received back from the algorithm. The mc_rpc_call_nowait() and mc_rpc_wait_for_results() are used for the multiprocessor

algorithms so that multiple slave algorithms can be started without waiting for the results of the earlier slave algorithms.

6.2 POSIX Semaphores

Two POSIX semaphores are used to provide synchronized access to the SMBs. One semaphore signifies that the master is done with its operations, and the other signifies that the slave is done with its operations. Only two semaphores are needed, since the sequence of transfers is defined by the command which is sent from the master to the slave. The command defines the number of inputs that will be sent from master to slave, and the number of outputs that will be sent from slave to master. The two semaphores insure that neither side writes or reads until the other side has performed its appropriate operation.

6.3 Data Transfers

The command SMB is transferred using PIO since it is a small data transfer. The data transfers between master and slave data SMBs use PIO for small (scalar variables like number_of_rows) data buffers and DMA for large transfers (vectors). For each algorithm there is a data transfer for the command and multiple data transfers for each input and output. Each transfer is controlled by a handshake using the POSIX semaphores master-done and slave-done.

6.4 Tradeoffs

This design was chosen for its flexibility. One data SMB is created for master and slave. The size of these SMBs must be as large as the largest input vector. An algorithm can have as many vector inputs and outputs as desired, as long as each vector does not exceed the allocated SMB size. This design offers the most flexibility for different types of algorithms. It is especially attractive since a master data SMB must be created on the host for each of the slave processes (8 in the case of AdapDev). This design establishes a limit on the size of the host SMBs which insures that the total for all 8 SMBs will not exceed the available memory for host-side SMBs. However it does not offer the greatest throughput. The semaphore handshaking which must be done around each data transfer, and the overhead for each DMA transfer detracts from the overall throughput. If all the data was sent in one data transfer, then this overhead would be limited. If data transfer time is found to be inadequate in the future this design could be modified to transfer all inputs together in one SMB (and, similarly, all outputs in one SMB).

6.5 Constraints

This remote procedure call interface was originally designed to be used with a master on either the host or CE. However all the testing was performed using the host for a master. Testing will be needed to insure that this interface also works when a CE performs the master functions (mc_rpc_call()).

7.0 Conclusions

A testbed environment was developed that provides a user with the capability to easily migrate a MATLAB algorithm to the Mercury AdapDev real-time architecture. The infrastructure software to configure and initialize the real-time hardware architecture was developed. A remote procedure call interface was developed. Timing analysis instrumentation tools was integrated into this testbed.

The testbed capability was demonstrated by rapidly prototyping the Polyphase Filter Channelization and Perfect Reconstruction algorithms. Parallel implementations of these algorithms were produced. Low-level signal processing functions required by these algorithms were implemented. The parallel algorithm results were validated against the MATLAB algorithm results. Initial performance estimates were obtained for single and multiple processors.

The capability for streaming real-time data into MATLAB was investigated. The Matlab library was procured from ICS to allow a MATLAB interface. Limited testing of streaming data collection via MATLAB was performed. A max data rate to RAID via MATLAB of ~6M samples/sec x 2 channels was obtained.