

Module 30 - Lab A: Executable Requirements & Specifications – Veribest Version

Executable Requirements Modeling using VHDL Tutorial

Copyright 1995-1999 SCRA

All rights reserved. This information is copyrighted by the SCRA, through its Advanced Technology Institute (ATI), and may only be used for non-commercial educational purposes. Any other use of this information without the express written permission of the ATI is prohibited. Certain parts of this work belong to other copyright holders and are used with their permission. All information contained, may be duplicated for non-commercial educational use only provided this copyright notice and the copyright acknowledgements herein are included. No warranty of any kind is provided or implied, nor is any liability accepted regardless of use.

The United States Government holds “Unlimited Rights” in all data contained herein under Contract F33615-94-C-1457. Such data may be liberally reproduced and disseminated by the Government, in whole or in part, without restriction except as follows: Certain parts of this work to other copyright holders and are used with their permission; This information contained herein may be duplicated only for non-commercial educational use. Any vehicle, in which part or all of this data is incorporated into, shall carry this notice .

See the [RASSP Disclaimer file](#) for additional RASSP Disclaimer, Warranty and Limitation of Liability Information concerning the material, VHDL code and software developed under the RASSP programs or incorporated in RASSP material.

1. Overview

In this lab experiment, you will be given a set of written design requirements and you will be asked to capture these requirements in an executable form. The goal of this exercise is to highlight a methodology for describing requirements in an executable form and the role it plays in the topdown design process. We will also differentiate between requirements and specifications and show how the requirements help drive the design specifications. The laboratory design will consist of a data source (represents the sensor system), a fast Fourier transform (FFT) signal processor, and a data sink (represents a display) as shown in Figure 1.

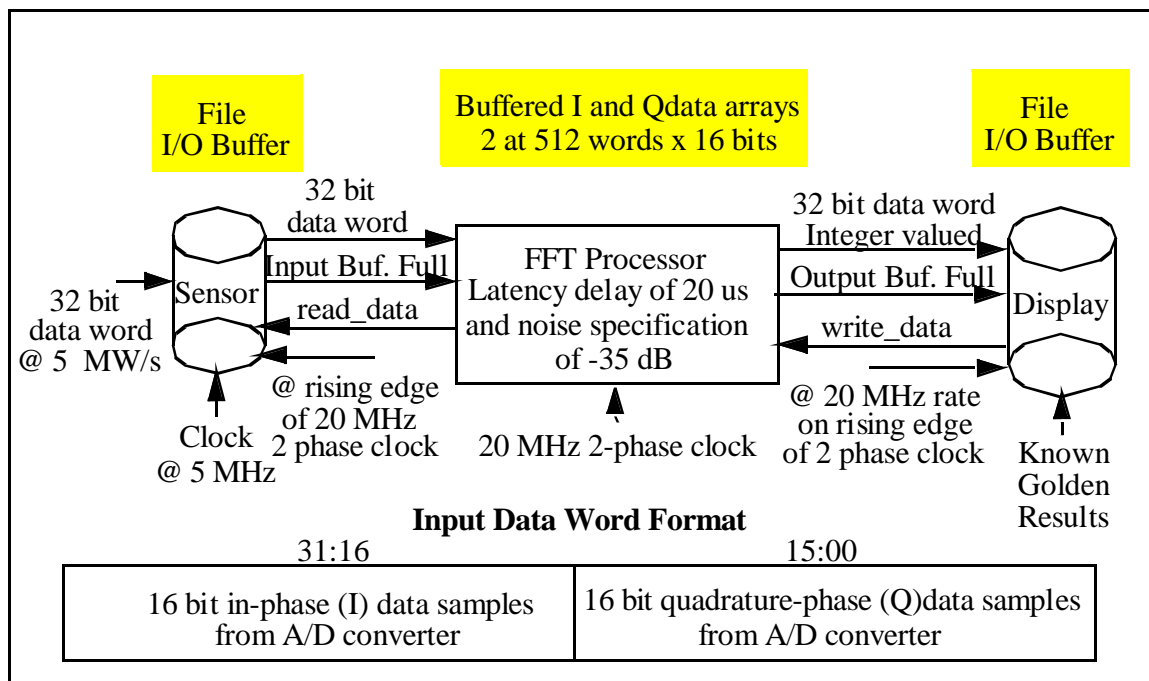


Figure 1 : Requirements for FFT Processor System

2. Design Requirements

The design requirements for this hypothetical signal processing system are listed below and are also shown in Figure 1 above.

- 1) Sampled complex data arrive from a sensor system as 16 bit signed integer values representing the data's real and imaginary parts. A 32-bit data word is created from the A/D sampled input signal and contains the 2 16-bit I/Q (real and complex parts) data samples received at the specified rate (in this case, the rate is 5 MW/s where the word size is 32 bits). The format of the data word is shown in Figure 1 where the lower 16 bits represent the complex part and the upper 16 bits represent the real part.

- 2) The sensor collects data and buffers it until it receives 512 complex samples. At this point, it must interrupt the processor to transfer data across the sensor/processor interface. The sensor/processor interface must send data to the processor in the same 32-bit format at an internal clock rate of 20 MHz. The data transfer begins when the I/O buffer of the sensor notifies the processor it is full (512 samples obtained from sensor). There is a clock available to the system that must be utilized. The clocking mechanism within the system is two-phase with a frequency of 20 MHz as shown in Figure 2. The data must be placed on the bus on the rising edge of the phase 1 clock and the processor will read the data on the rising edge of the phase 2 clock.

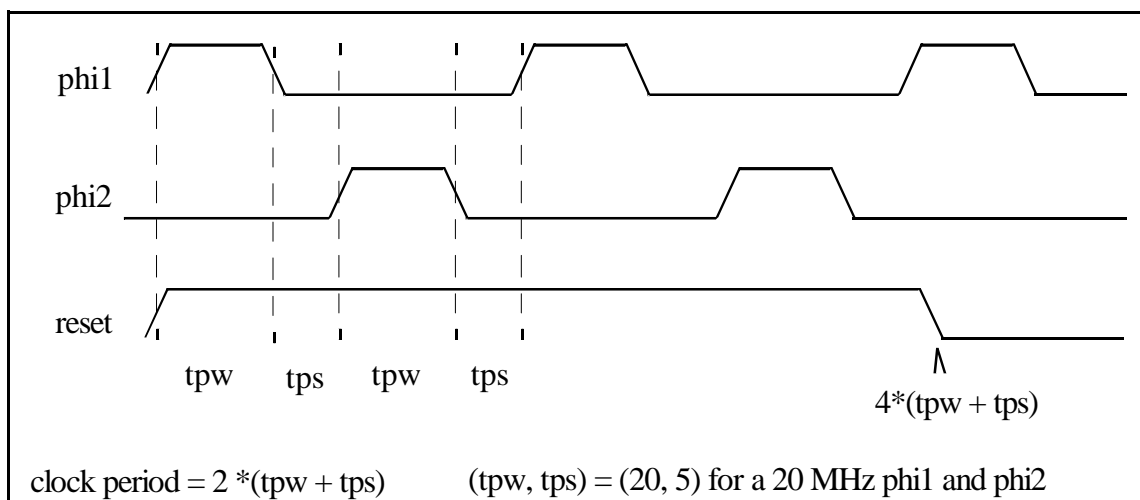


Figure 2 : Internal two-phase clocking system requirements

- 3) The sensor continuously collects data and when its buffer is full it sends the buffer full signal to the processor via an interrupt mechanism. The processor will respond (notify the sensor: read_data signal) when it is ready to accept data from the sensor. This signal from the processor should remain active high during the entire transfer of data from sensor to processor and return low again after the processor has filled its input buffers.
- 4) The system's processor block must buffer the data from the sensor into two 512 input data fifos, process the data using an FFT (in this case, only move the data to the output buffers after a specified processing latency), and finally send the processed data to the display unit.
- 5) The FFT processing must be completed within a delay latency of 20 us.

- 6) The data interface from the processor to the display unit operates at a 20 MHz rate. In this system, the output data format is 32-bit integers where alternating output samples from the processor represent the real and imaginary parts of the FFT result.
- 7) The processor must send an output buffer full signal to the display unit when it has enough samples to send and the display unit must reply with an active high acknowledge signal. The acknowledge signal must remain high during the entire data transfer and become inactive when the display unit's input buffers are full. The processor will send data while the acknowledge signal is high or until its output buffer is empty, where in this case, that implies 512 complex samples. The processor must place the data on the data output bus on the rising edge of the phase 1 clock signal and the display must read the data on the rising edge of the phase 2 clock signal. This must occur on the first rising edge of the phase 1 clock after the acknowledge signal is received from the display.
- 8) We require the processing noise specification to be no worse than -35 dB, which will not be important for this experiment but will be used in the executable specifications laboratory, M30_Lab_B.
- 9) A comparison mechanism must be designed into the display unit of the test bench so that the output results can be compared with known good results. This requires the use of file I/O to read the known good data from a file prior to comparison.
- 10) The behavior of the system must be specified on reset. In this system, the following output signals should be driven to the following states on reset:

Sensor entity:

buf_full = '0' and data = tri-state 'Z'

Processor entity:

read_data_ack = '0' and buf_full_out = '0' and data_out = tri-state 'Z'

Display entity:

write_data_ack = '0'

This design requirement experiment will be used to explore, (1) how executable requirements are captured, (2) the advantages of developing a executable requirement at the top level, and (3) how these flow down to lower levels in the design process.

3. What you will learn

- 3.1. What distinguishes executable requirements from executable specifications.
- 3.2. How executable requirements can be used in a top-down design methodology.
- 3.3. How to transform written design requirements into executable VHDL form.
- 3.4. How to specify the requirements of an embedded system design (I/O, processing latency, throughput, etc.).
- 3.5. Why executable requirements are primarily a system testbench design problem.
- 3.6. How to develop an effective executable requirement's testbench.

4. Create the directory structure and component libraries

- 4.1. Create a working directory in your home directory (*<home_dir>* in Figure 3) with the name “m30_lab_a” and go to that directory to start working. For Windows systems, create a new file folder as the directory. In this write-up, we assume you are using the Veribest simulator on the PC.

```
WINDOWS>> File->New->Folder m30_lab_a
```

- 4.2. Copy the executable requirements lab files to that directory from the CD-ROM. For Windows systems, select the file from the CD-ROM directory and drag it into the working folder just created, “m30_lab_a”.
- 4.3. Setup your VHDL environment correctly so that you have access to the simulator's executables. For Veribest, ensure the tools have been installed correctly on your machine and that the license.dat file has been placed in the c:\flexlm directory if it is required for the restricted version of the simulator.
- 4.4. Copy the zipped file containing the component models, m30_lab_a.zip, to your home directory. Using the *winzip* or *pkzip* utilities on your PC, unzip the file, and look at the directory structure. It should have the form shown in the Figure 3. If *winzip* or *pkzip* are not available on your machine, then copy the files directly from the CD-ROM.

```
<home_dir>>> winzip m30_lab_a.zip or  
<home_dir>>> pkzip m30_lab_a.zip
```

- 4.5. The files in the *<src>* directory contain the entity/architecture pairs for the components of the system. These include the sensor (source), FFT processor (processor), display (sink), clock (clock), special conversion routines (conversions), and system files (system). The *<libs>* directory will contain the compiled files after you finish with the build process. For the PC platform using

Veribest, we will build these libraries in the next section. To compile the VHDL files for this example into separate libraries, the following order of compilation is required for each of the component elements.

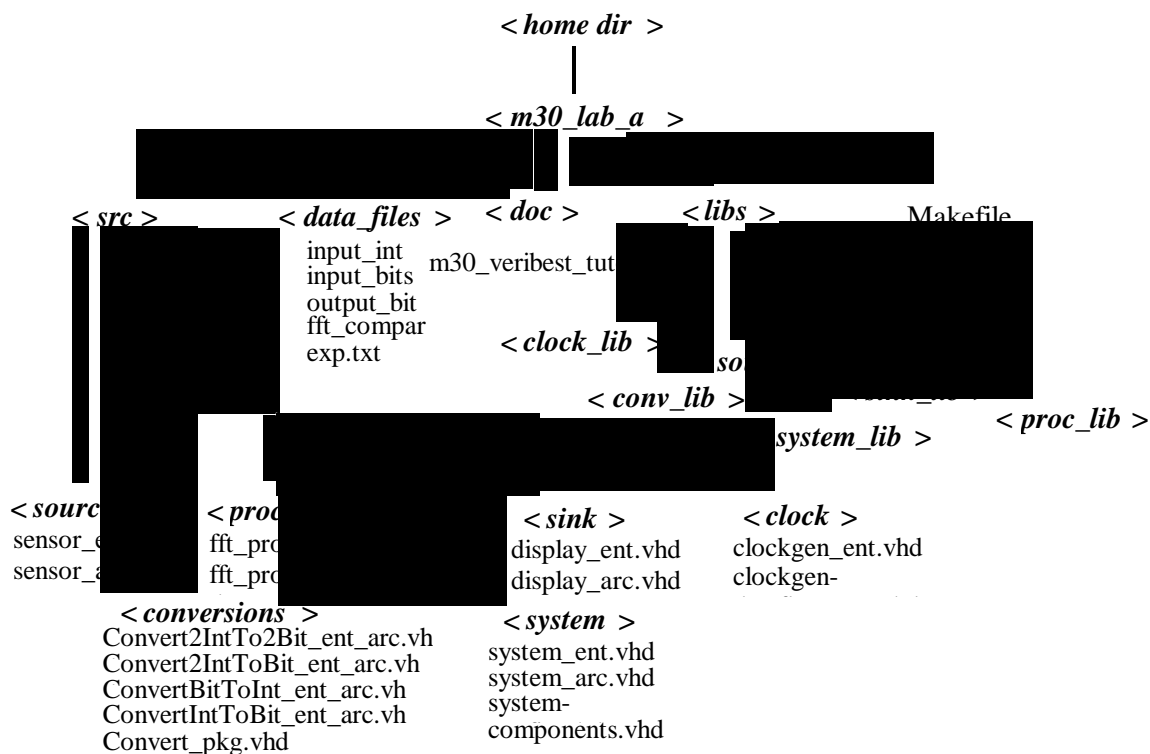


Figure 3 : Experiment File Hierarchy

- A. The files and compilation order for the clock (clock directory) model are the following:
 1. clockgen_ent.vhd
 2. clockgen-dataflow_arc.vhd
- B. The files and compilation order for the conversion (conversions directory) models are the following:
 1. Convert_pkg.vhd
 2. Convert2IntTo2Bit_ent_arc.vhd
 3. Convert2IntToBit_ent_arc.vhd
 4. ConvertIntToBit_ent_arc.vhd
 5. ConvertBitToInt_ent_arc.vhd

- C. The files and compilation order for the sensor (source directory) model used in this example are:
 - 1. sensor_ent.vhd
 - 2. sensor_arc.vhd
- D. The files and compilation order for the processor (processor directory) model used in this example are:
 - 1. fft_proc_ent.vhd
 - 2. fft_proc_arc.vhd
- E. The files and compilation order for the display (sink directory) model are the following:
 - 1. display_ent.vhd
 - 2. display_arc.vhd
- F. The files and compilation order for the system (system directory) models are the following:
 - 1. components.vhd
 - 2. system_ent.vhd
 - 3. system_arc.vhd
 - 4. system-config.vhd

The makefile provided with the distribution is designed for use with Mentor Graphics QuickHDL tool set and uses the *make* utility on UNIX systems. If you want to use the Mentor tool suite on a UNIX platform, look at the m30_lab_a Mentor documentation for this laboratory. There is no equivalent makefile for the Veribest simulator provided with the CD-ROM.

On the PC platform using Veribest, do the following to build the necessary libraries. First start the Veribest simulator.

```
WINDOWS>> Start->Programs->Veribest
```

Open a new workspace to build the clock library. Build the clock library in the M30_Lab_A\libs\clock_lib directory. In the Veribest simulator, select the following.

```
VERIBEST>> File->New Select Workspace and click ok
```

Create the workspace with the name *clock* and the workspace path to be the M30_lab_a\libs\clock_lib directory.

```
VERIBEST>> Create Workspace Name Path and click create
```

A window appears as shown in Figure 4. This also creates the appropriate work library within the directory. We next must add the clock specific entity and architecture files to the clock workspace. To add the files, click on the plus (+)

button in the workspace window and a new window will appear asking you to add the appropriate files to the workspace. Go to the m30_lab_a\src\clock directory and select the clock entity and architecture files.



Figure 4: Clock Workspace

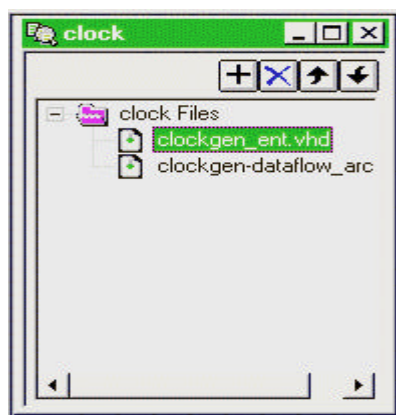


Figure 5: Clock Workspace with Entity and Architecture Files

After loading the files into the workspace, the screen should now appear as in Figure 5. The compilation order is from top to bottom within the workspace window, so the clockgen_ent.vhd file should be listed before the architecture file as shown in Figure 5. If the files are not in the correct order, then move them appropriately using the up and down arrows within the workspace window so that the compilation order is correct.

Next, we would like to compile the design files into our working library. First, set the compilation settings by doing the following from the pull-down menu or by clicking on the quick-select buttons below the pull-down menu options.

```
VERIBEST>> Workspace->Settings
```


At this point, a new window appears allowing the user to set the proper settings for compilation. Select the compilation library to be WORKLIB and the debugging switch to be turned on. Now we can proceed with the compilation of the files within the workspace. Use the menu or the quick-select buttons to do the compilation of both files in the workspace.

```
VERIBEST>> Workspace->Compile All
```

After successful compilation of the clock library files, we can now proceed to building the remaining libraries for the overall system design. Start by saving and closing the current clock workspace and selecting a new workspace. Develop the workspaces in the order mentioned above, i.e. clock_lib, conv_lib, source_lib, proc_lib, sink_lib, and finally system_lib. Build these in their associated directories and name the workspaces clock, conv, source, proc, sink, and system respectively. When you reach the development of source_lib, there is another setting that must be applied. This requires the addition of a library mapping to conv_lib. The mapping is required because conv_lib is included within the source design files (USE VHDL statement). To create a library mapping for the current workspace, select the pull-down menu Library and select the Add Lib Mapping option.

```
VERIBEST>> Library->Add Lib Mapping
```

A window appears for the user to enter the appropriate information. The physical and logical names of the library must be entered. To add the library mapping, traverse through the file pathnames until you reach the M30_lab_a\libs\conv_lib directory. When you select this directory, the library name WORKLIB appears under the physical names section. Select this physical name and then type in the appropriate logical name. In this case, we need the logical name *conv_lib* to be mapped to the physical library WORKLIB in this section. Click OK when you are finished. Once this is done, compile the files into the source working library. Proceed in building all the remaining libraries, i.e. proc_lib, sink_lib, and system_lib. For proc_lib and sink_lib the library mapping to conv_lib is required as done in source_lib. For system_lib, the library mappings to clock_lib, conv_lib, source_lib, proc_lib, and sink_lib are all required so these must be added prior to correct compilation.

5. Capturing the written requirements in an executable form

In the following sections, we will show how to capture the written requirements of this system in an executable VHDL form.

5.1. Requirement 1: The system clock will operate with 2 phases and at a 20 MHz frequency and reset will be active for 2 clock periods on system initialization.

Using the clock entity and architecture files in the *m30_lab_a\src\clock* directory, create a clock that will meet this system requirement. Edit the file *clockgen_ent.vhd* and modify the generic parameters found within the entity description to create the clock waveforms shown in Figure 6. This requires changes to the *tps* and *tpw* parameters. *Reset* must also be set to match that shown in the figure. For all references to editors within this document, use the editor within the Veribest tool to make the changes to the files. You can choose your own editor as a replacement in each case. First, open the clock workspace to begin the editing.

VERIBEST>> File->Open Workspace \libs\clock_lib\clock
Double click on the file that you need to edit in the workspace window.

VERIBEST>> Double click on *clockgen_ent.vhd*
After the file opens in the working area, edit using the Note Pad-style editor provided with the Veribest tool.

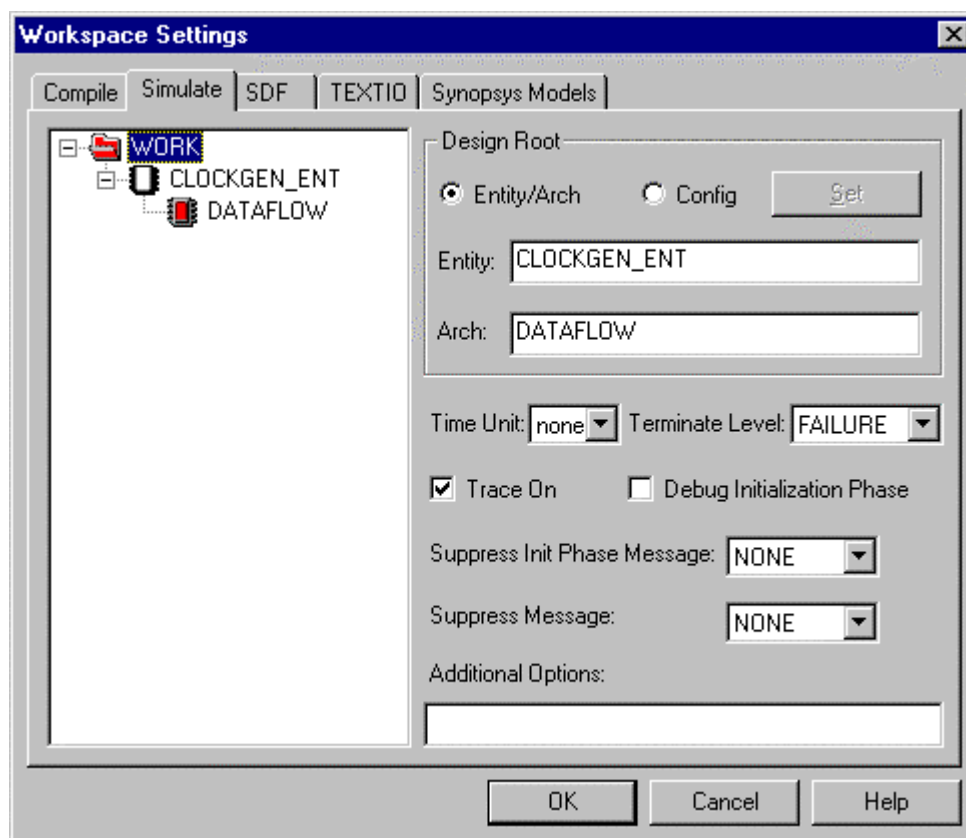


Figure 6: Workspace Settings to Select the Entity and Architecture

editor>> search for tpw and tps and modify to create the correct waveforms in Figure 6

editor>> save the file (ctrl-s while it is selected)

Now that the file has been modified, recompile both the entity and architecture files for the clock circuit and simulate the design.

VERIBEST>> Workspace->Compile All

You should now have an update of the clock and reset functionality. Next, simulate the design and check the timing for correctness. To simulate the design, first configure the simulator settings.

VERIBEST>> Workspace->Settings

In the Workspace Settings window, select the simulator option. Expand the work library to show the items contained within it. At this point, you should only have the clock entity and the dataflow architecture. Select these as shown in Figure 6 and click on the OK button. You are now ready to simulate.

To simulate the design, click on the simulate hot button or choose from the pull-down menu.

VERIBEST>> Workspace->Execute Simulator

To view waveforms for the clock signal, once the simulator has started, select the waveform viewer from the tools menu.

VERIBEST>> Tools->New Waveform Window

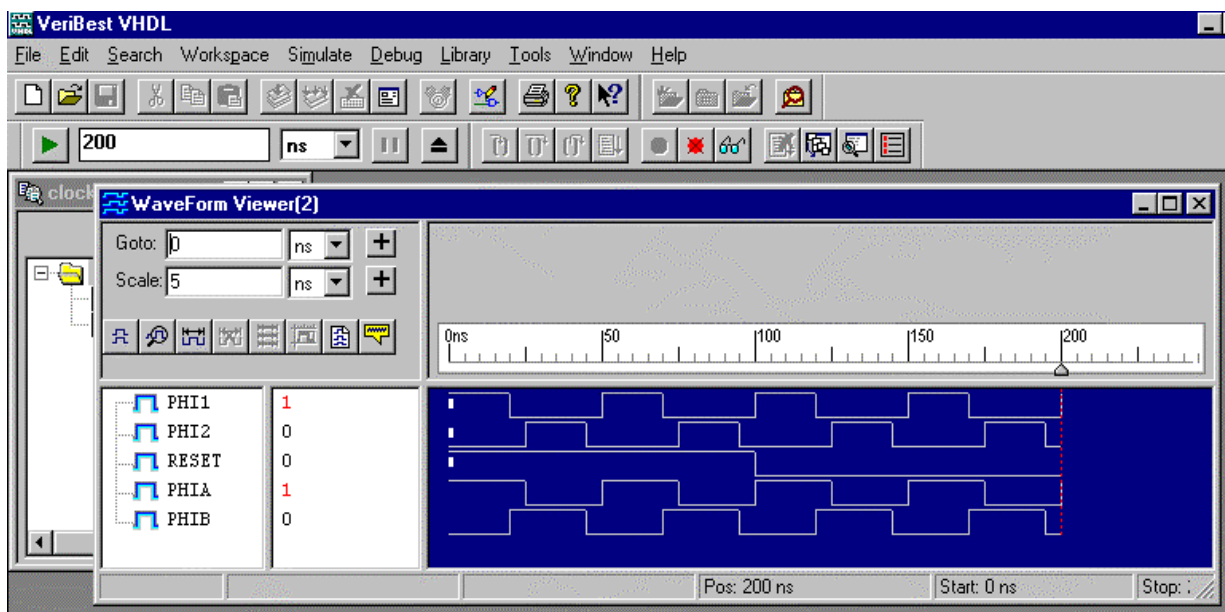


Figure 7: Clock Waveform with tpw = 20 ns and tps = 5 ns

Once the waveform viewer appears, add signals to the waveform viewer using the left-most button from the buttons options.

```
WAVEFORM_VIEWER>> click on the left-most button
WAVEFORM_VIEWER>> Select the 'Add All' option to view
all signals in the design and 'close' the selection
tool
```

You are now ready to run the simulation. The length of the simulation runtime is set in the upper left hand corner (default is 100 units). Run for 200 ns and view the waveform. You should see the results shown in Figure 7.

Verify that your clock waveform has a period of 50 ns for both phases of the clock and that the reset is active high for 2 clock periods as shown in Figure 7.

5.2. *Requirement 2: On reset the following output signals shall be set to the values listed below*

Sensor entity:

buf_full = '0' and data = tri-state 'Z'

Processor entity:

read_data_ack = '0' and buf_full_out = '0' and data_out = tri-state 'Z'

Display entity:

write_data_ack = '0'

To model this requirement, each functional architecture of the three main components (sensor, processor, and display) must have a section of code sensitive to reset = '1'. Open the following files and go to the lines listed below to observe where this is modeled in the code. For *sensor_arc.vhd*

```
VERIBEST>> File->Open \src\source\sensor_arc.vhd
```

Observe line numbers 116-124 in this section where buf_full is assigned '0' and data is assigned 'Z'.

```
VERIBEST>> File->Open \src\processor\fft_proc_arc.vhd
```

Observe line numbers 112-132, 216-219, and 249-251 in this code where the signals listed above are set appropriately. Also in these sections, various internal signals are set to inactive values.

```
VERIBEST>> File->Open \src\sink\display_arc.vhd
```

Observe line numbers 119-123 in this code where the signal *write_data_ack* is set to '0'.

Next, we will run the full system for two clock periods to observe the output waveform resulting from a system reset. Since we have modified the code for the clock, recompile the system configuration files.

```
VERIBEST>> File->Open Workspace \libs\system_lib\system
```

```
VERIBEST>> Workspace->Compile All
```

You should now be ready to run the simulator.

```
VERIBEST>> Workspace->Settings
```

In the Workspace Settings window, select the simulator section. Expand the work library to show the items contained within it. At this point, you should select the system configuration file (SYSTEM-CONFIG) rather the entity-architecture pairs. Select these as shown in Figure 8 and click on the OK button. You are now ready to simulate. Execute the simulator.

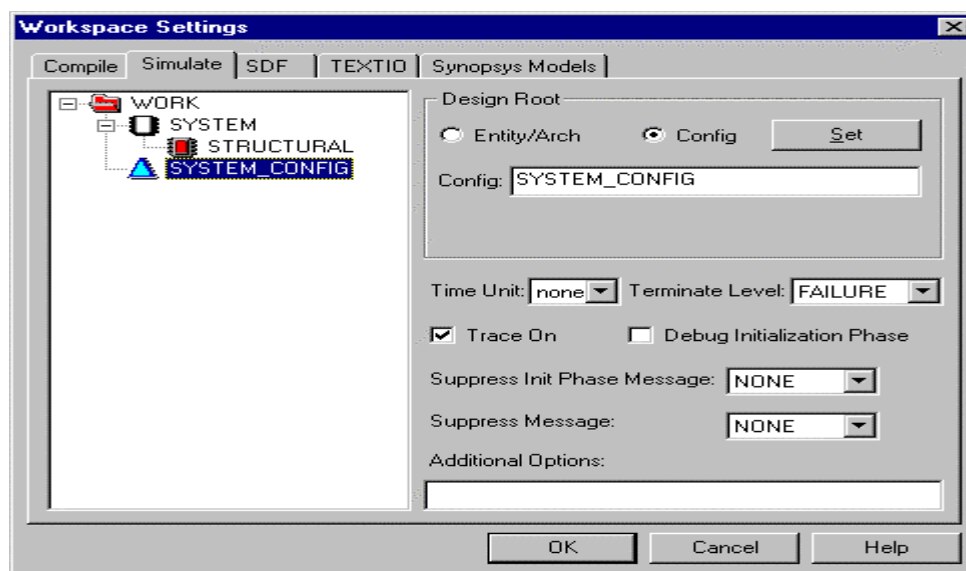


Figure 8: Workspace Settings to Select the System Configuration

```
VERIBEST>> Workspace->Execute Simulator
```

```
VERIBEST>> Tools->New Waveform Window
```

Once the waveform viewer appears, add signals to the waveform viewer using the left-most button from the buttons options.

```
WAVEFORM_VIEWER>> click on the left-most button
```

```
WAVEFORM_VIEWER>> Add the appropriate signals as  
mentioned early to verify the values.
```

You are now ready to run the simulation. Run for 110 ns and view the waveform resulting from a system reset.

Verify that the output signals of each of the three main components have been properly set. Placing a cursor at 105 ns and observe the signal values in the left-hand window of the waveform viewer. Also see Figure 9 for waveform viewer results.

- 5.3. *Requirement 3: Complex sensor data is continuously sampled at a 5MHz frequency and is stored in an input buffer. The buffer shall be large enough to store 512 samples plus any input samples that arrive during the send operation to the processor.*

Since this is a sensor requirement, open the *sensor_arc.vhd* file if it is not already open. First, there is a requirement for data to be sampled at a 5 MHz rate. This implies the need for a 5 MHz clock signal. Since the system clock is 20 MHz, we can divide the system clock by 4 and obtain the 5 MHz sampling clock. This can be achieved by using a special process running inside the sensor architecture that is sensitive to phase one of the system clock. Observe lines 189-215 of this file to see how this can be accomplished. The generic parameters *divide_count* and *half_clock_period* can be set inside the entity file *sensor_ent.vhd* and can be overridden by setting them in the system configuration file, *system-config.vhd*.

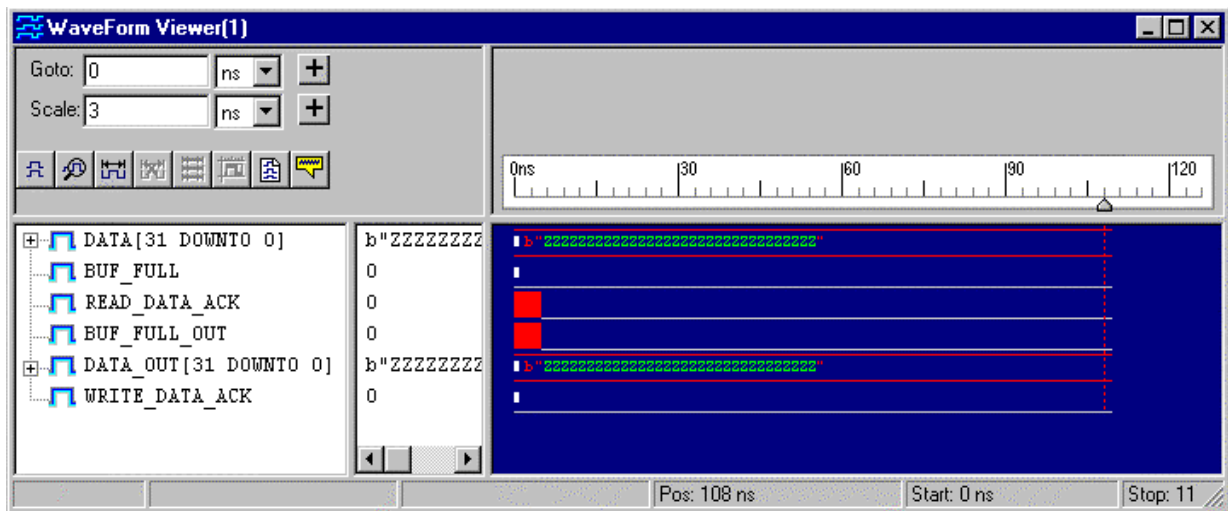


Figure 9: Waveform after System Reset

There is also the requirement for buffering the input data. This can be modeled using array types that represent buffers. Observe the declaration lines 104-108 in the *sensor_arc.vhd* file. Here a buffer type is defined that has a length specified by the generic parameter *buffer_size* and two buffers, one each for the real and imaginary parts of the sampled data (*i_buffer_array* and *q_buffer_array*). Also,

the buffer length is defined to be 1024 samples so that when 512 samples have been collected, the sensor can send data to the processor from half the buffer while continuously collecting more data samples in the second half of the buffer. Sensor hardware is modeled using file I/O where data is taken from the file specified by the generic parameter *input_sensor_data*. Those samples coming from an actual sensor are, in our model, coming from a file. Observe lines 126-142 where on each rising edge of the 5 MHz clock, data is read from a file, placed into the I and Q buffers, and buffer pointers are incremented.

We will next run the model and observe the behavior of this requirement. Restart the simulation by selecting 'restore' from the Simulate pull-down menu.

```
VERIBEST>> Choose Simulate->Restore and select 0 ns
```

Close the old waveform viewer and open a new one.

```
VERIBEST>> Tools->New Waveform Window
```

Once the waveform viewer appears, add signals to the waveform viewer using the left-most button from the buttons options.

```
WAVEFORM_VIEWER>> click on the left-most button
```

```
WAVEFORM_VIEWER>> Add all the signals contained in the  
system interface.
```

You are now ready to run the simulation. Run for 50000 ns.

```
VERIBEST>> Run for 50000 ns
```

Next, set a breakpoint in the sensor architecture at line number 129 so that we can observe the internal buffer arrays. First, open the file *sensor_arc.vhd* in the working area. The file should open with line numbers on the left-hand side. If it does not, then you may not have set the debugging switch on initial compilation. If this is the case, go back to the 'source' workspace and recompile the sensor with the debug switch turned on. Once the file is open and line numbers exist within it, go to line 129 with the cursor. Click on the red button or use the pull-down menu to set a breakpoint at this location in the code.

```
VERIBEST>> Debug->Insert/Remove Breakpoint (line 129)
```

Continue running the simulation from this point and wait until it reaches the breakpoint.

```
VERIBEST>> Run for 150 ns
```

View the I internal buffer variable (*i_buffer_array*) within the simulation's sensor architecture by selecting (highlighting) the variable within the code and clicking on the examine hot button (eye glasses).

```
VERIBEST>> Click on eye glasses hot button
```

Expand the array within the examine window as shown in Figure 10.

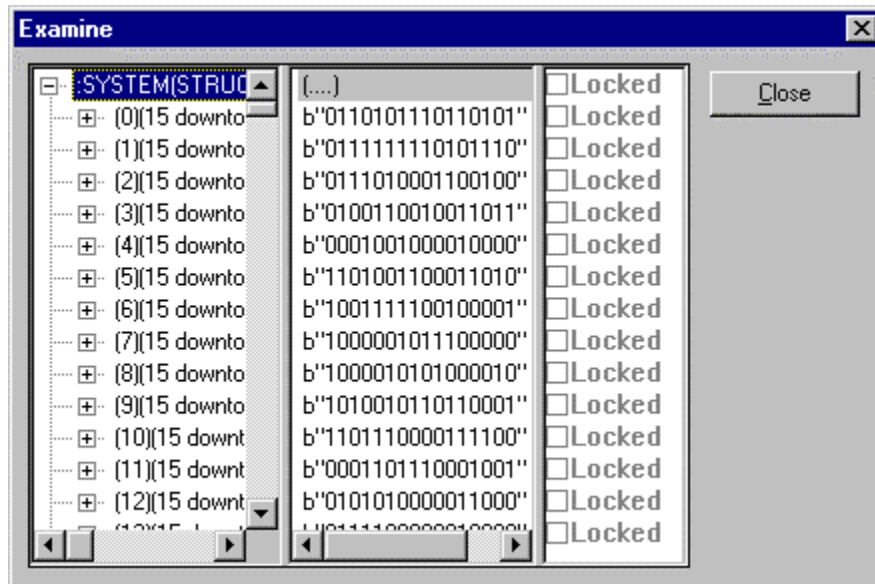


Figure 10: I Internal Buffer Array Contents

At this point you should see 250 nonzero values within the I and Q internal buffers of the sensor as shown in Figure 10. Data continues to fill the buffer until it has reached a point where it is ready to send data to the processor. This occurs when 512 samples are read into the buffer (the number of samples is specified by the generic parameter 'send_amount' within the sensor entity). Remove this breakpoint and run the simulator for another 52400 ns and observe the waveform results when the sensor buffer becomes full. This occurs at time 102400 ns.

Click on the red button with the X through it or use the pull-down menu to remove the breakpoint at this location in the code.

```
VERIBEST>> Debug->Insert/Remove Breakpoint (line 129)
VERIBEST>> Run for 52400 ns
VERIBEST>> Select the 'GOTO' location in the waveform
viewer to be 102320 ns and the 'scale' to be 5 ns and
view the waveform
```


The *buf_full_sig* signal becomes active and waits for a response from the processor unit acknowledging the beginning of data transfer (Figure 11).

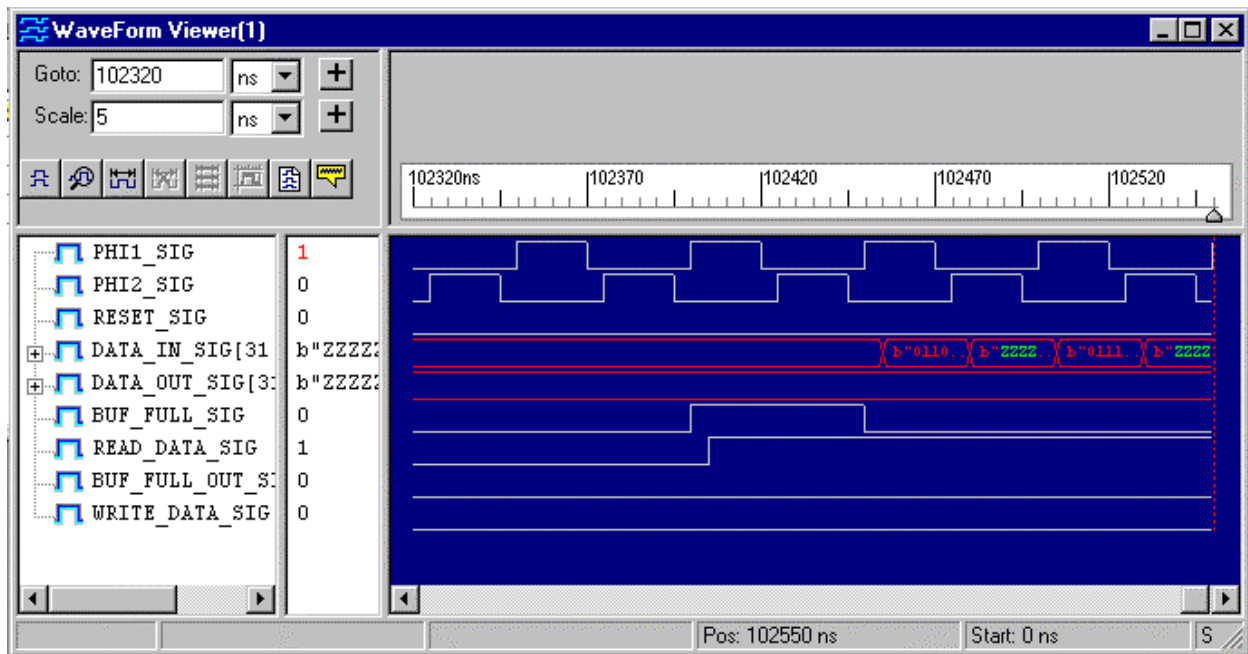


Figure 11: Data Transfer Begins from Sensor to Processor Entity

5.4. Requirement 4: The data word format shall be 32 bits containing both the real and imaginary parts of the complex data (16 bits signed integer for each) where the upper 16 bits shall contain the *I* samples and the lower 16 bits shall contain the *Q* samples.

For this requirement, set the range of viewing the waveform data as follows.

```
VERIBEST>> Select the 'GOTO' location in the waveform
viewer to be 102440 ns and the 'scale' to be 1 ns and
view the waveform
```

Observe the values on the sensor data output signals and compare them with those values contained within the *I* and *Q* sensor buffers. Since this is the first value sent to the processor the upper 16 bits should match the first element in the *I* buffer and the lower 16 bits should match the first element in the *Q* buffer.

This requirement can also be observed in the code of the sensor architecture model by observing lines 160-162 of *sensor_arc.vhd*. In this code segment we see that data values are assigned to their outputs by concatenating the *I* and *Q* buffer values. Figure 12 shows the results captured on the waveform viewer.

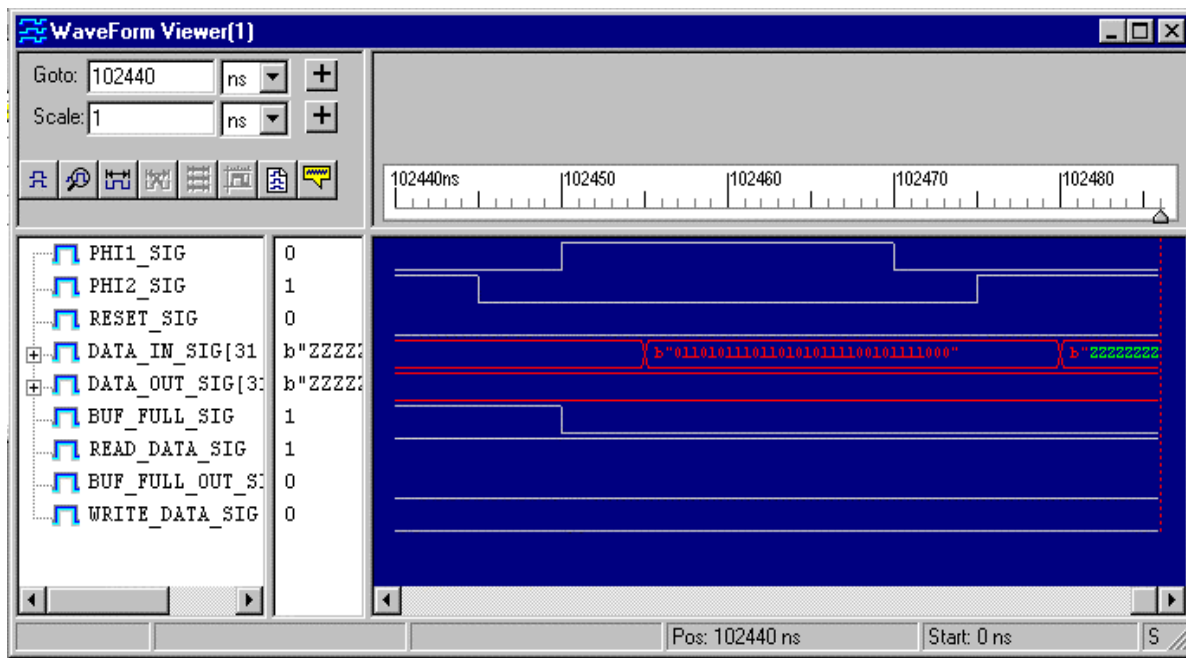


Figure 12: First Sensor Data Output to the Processor

5.5. Requirement 5: Sensor data must be placed on the bus on the rising edge of the phase one clock and read by the processor on the rising edge of the phase 2 clock. The transfer rate shall be 20 MW/s (32 bits/word).

This requirement can be viewed on the waveform plot by zooming into the following range and observing that data is placed on the bus 5 ns after the rising edge of the clock and driven until 5 ns after the rising edge of the phase 2 clock. as shown in Figure 13.

VERIBEST>> Select the 'GOTO' location in the waveform viewer to be 102380 ns and the 'scale' to be 3 ns and view the waveform

5.6. Requirement 6: The processor shall respond with a acknowledge signal when it is ready to accept data from the sensor after the sensor sends it a notification that its buffer is full. It shall hold this signal active high until it has received enough data to process.

This can be observed in Figure 13 by noticing that when the *buf_full_sig* signal from the sensor goes active high, the FFT processor signal, *read_data_ack*, goes high 5 ns later. Next, data is sent to the processor until its input buffer is full, at which point the processor deactivates the acknowledge signal. The return of the acknowledge can be observed in the file *fft_proc_arc.vhd* at lines 223-226. The process is sensitive to the buffer full signal from the sensor. Data is

placed into the internal buffer of the processor by observing the code lines 134-149 of the same file. In this section of code, the *read_data* signal from the processor and phase two of the clock both must be active in order for data to be taken off the bus. When the input buffer becomes full (lines 151-164 of *fft_proc_arc.vhd*), processing can begin and the data acknowledge signal to the sensor is disabled.

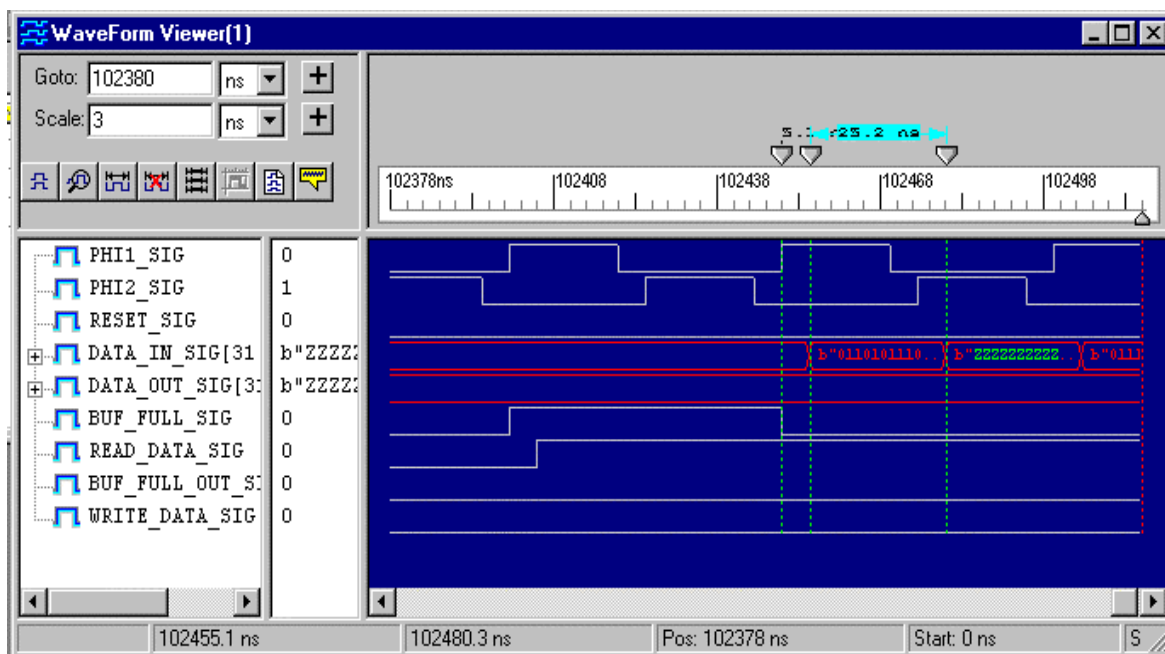


Figure 13: Waveform capturing requirements 5 and 6

5.7. Requirement 7: The processor shall buffer the input data into 2-512 element buffers and process the data with a maximum latency of 20 us at which point it shall send the processed data to the display unit.

To observe and model this behavior, restart the simulation and run it for 302 us. Remove all breakpoints if this has not already been done.

VERIBEST>> Choose Simulate->Restore and select 0 ns

Close the old waveform viewer and open a new one.

VERIBEST>> Tools->New Waveform Window

Once the waveform viewer appears, add signals to the waveform viewer using the left-most button from the buttons options.

WAVEFORM_VIEWER>> click on the left-most button

WAVEFORM_VIEWER>> Add all the signals contained in the system interface.

You are now ready to run the simulation. Run for 302 us.

```
VERIBEST>> Run for 302 us
```

Zoom into the region that represents the processing. The processing begins when the `read_data_ack` signal from the processor returns inactive.

```
VERIBEST>> Select the 'GOTO' location in the waveform  
viewer to be 124000 ns and the 'scale' to be 800 ns and  
view the waveform
```

Observe the point where the processor receives the last data item and when the first data item is sent to the display. This can be measured by looking at the time difference between when the processor sets `read_data_ack` inactive low (128030 ns) and when it sends a output buffer full signal to the display (148050 ns). It is shown in Figure 14. The latency is modeled in the code at lines 151-164 of `fft_proc_arc.vhd`. This code segment is entered when the receive data counter value has reached the input buffer size and it responds with a signal to stop reading data. The signal to write data to the display is not assigned its value until after a delay of 'latency' time units. When the process sets this signal, the data is transferred from the input buffer to the output buffer (lines 166-175), it sets the `buf_full_out` signal (lines 255-258), and then monitors the transfer acknowledge signal from the display (line 181).

5.8. Requirement 8: The processor to display interface shall operate at a 20 MHz rate passing 32 bit real and imaginary integer data values where the real and imaginary values are alternately sent on the bus (real is sent first). Data is sent when the acknowledge signal from the display is active high. This signal remains high during the entire transfer. As in the sensor to processor interface, data is placed on the bus on the rising edge of the phase 1 clock and read by the display on the rising edge of the phase 2 clock.

Since both real and imaginary data must be sent totaling 1024 data values, the time required to send all samples will be double that of the sensor to processor interface. Observe the values being placed on the data output bus of the FFT

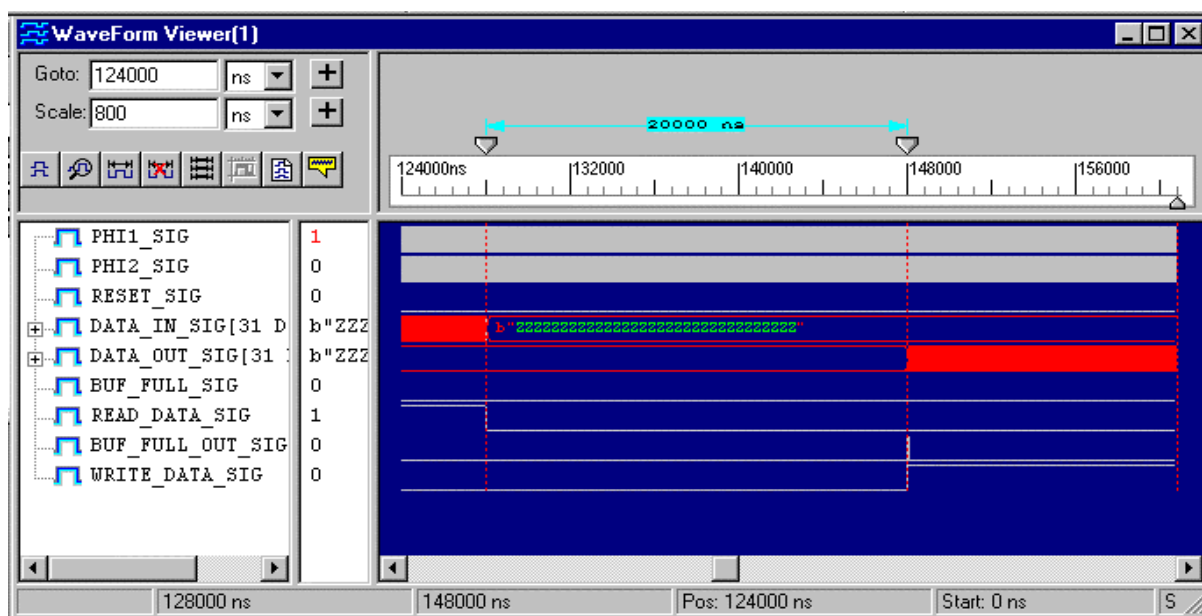


Figure 14: Model of Processor Latency

processor and compare them with the values in its output buffer as well as the original values in its input buffer. All values should match, however the values in the output buffer should be sign extended 32 bit versions of the original values in the input buffer. Restart the design and run it for 150 us to observe this behavior.

VERIBEST>> Choose Simulate->Restore and select 0 ns
Close the old waveform viewer and open a new one.

VERIBEST>> Tools->New Waveform Window

Once the waveform viewer appears, add signals to the waveform viewer using the left-most button from the buttons options.

WAVEFORM_VIEWER>> click on the left-most button

WAVEFORM_VIEWER>> Add all the signals contained in the system interface.

You are now ready to run the simulation. Run for 150 us.

VERIBEST>> Run for 150 us

Open the file \src\processor\fft_proc_arc.vhd so that the debugging information is contained within it. Set a breakpoint at line 166 of this file.

VERIBEST>> Debug->Insert/Remove Breakpoint (line 166)

Continue running the simulation from this point and wait until it reaches the breakpoint.

VERIBEST>> Run for 150 ns

Examine the contents of the *i_buffer_array*, *q_buffer_array*, and *output_buffer* and compare them with the values being placed on the *data_out* bus of the processor. Use the call stack functionality of the simulator to obtain this data as shown in Figure 15.

VERIBEST>> Debug->Call Stack

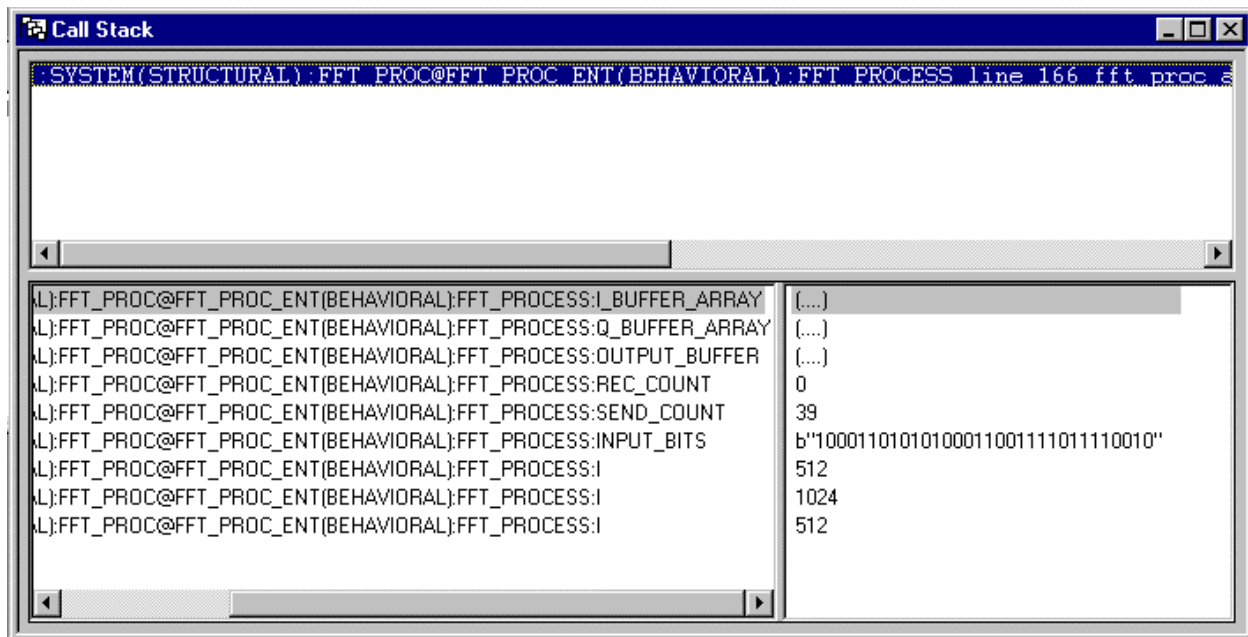


Figure 15: Call Stack Variables in the FFT Architecture

Observe the code segments that are modeling these operations. The data is being placed on the bus in the *fft_proc_arc.vhd* file (lines 181-193) only when the acknowledge signal is high from the display and on the rising edge of the phase 1 clock. The display unit file (*display_arc.vhd*) receives data from the bus on the rising edge of the phase 2 clock (lines 170-187) and places it into two input buffers named *fft_i_result_buffer* and *fft_q_result_buffer*. You can observe the contents of buffers using the call stack function from the pull-down menu.

- 5.9. *Requirement 9: A comparison mechanism must be designed into the display unit of the test bench so that the output results can be compared with known good results.*

This mechanism should read known good results from a file and compare them with the data received in the input buffers of the display unit. The comparison only should take place when the input buffers are full and data is not in the

process of being sent from the processor. This mechanism can be seen in the file *display_arc.vhd* at lines 192-224. Assertion statements are used if the comparison results in an error. The name of the input file to use for the comparison is passed to the display unit via generic parameters and can be found in *display_ent.vhd* at line 65. The parameter is called *fft_comparison_data* and is a string type. At this point, we have successfully modeled all the key parameters of the system requirements.