

# **Module 30 - Lab A: Executable Requirements & Specifications – Mentor Graphics Version**

## **Executable Requirements Modeling using VHDL Tutorial**

**Copyright 1995-1999 SCRA**

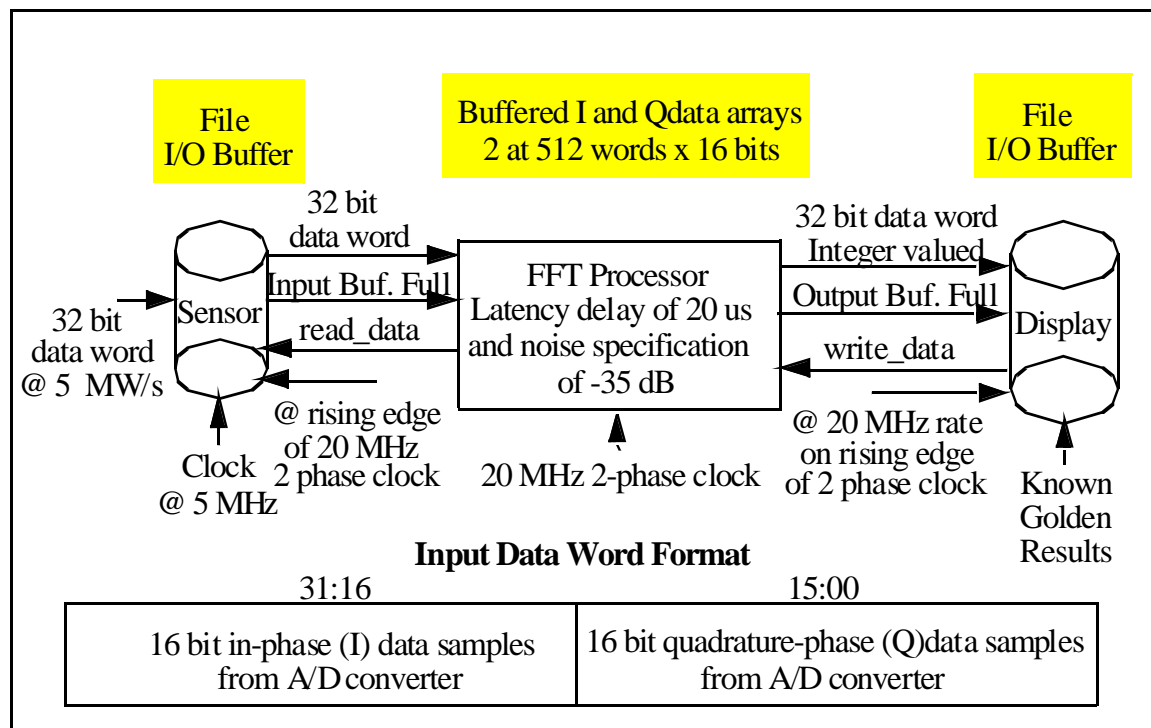
**All rights reserved. This information is copyrighted by the SCRA, through its Advanced Technology Institute (ATI), and may only be used for non-commercial educational purposes. Any other use of this information without the express written permission of the ATI is prohibited. Certain parts of this work 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**

**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.



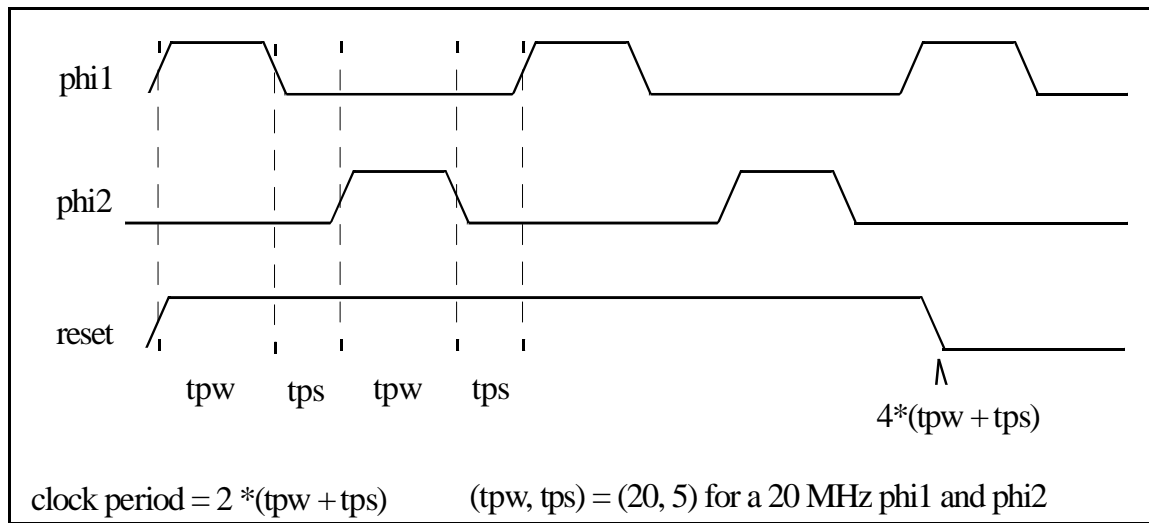
**Figure 1 : Requirements for FFT Processor System**

## 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 wordsize 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 will begin 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 specification 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.

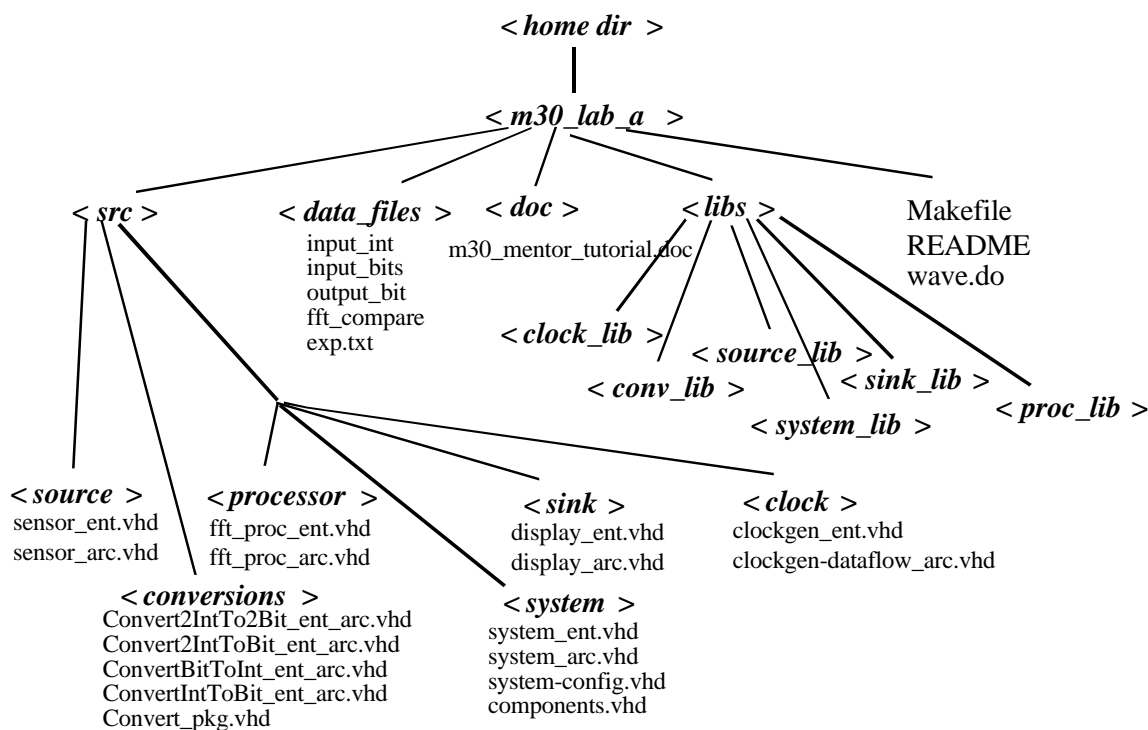
### 3. What you will learn

- 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 begin executing the laboratory. For UNIX systems, use the *mkdir* command.



**Figure 3 : Experiment File Hierarchy**

- 4.2. Copy the executable requirements lab files to that directory from the CD-ROM. For UNIX systems, use the *cp* command and copy the lab files into the directory just created, i.e. *m30\_lab\_a*.
- 4.3. Setup your VHDL environment correctly so that you have access to the simulator's executables on the UNIX system. For Mentor, the environment variable *MGC\_HOME* must point to the top-level directory of the Mentor tools.

- 4.4. Using the *tar* utility in UNIX, copy the file containing the component models, *m30\_lab\_a.tar*, to your home directory. Untar the file, and look at the directory structure. It should have the form shown in the Figure 3. If this utility is not available on your machine, then copy the files directly from the CD-ROM.

```
<home_dir> >> tar xvf m30_lab_a.tar
```

- 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 contains the compiled libraries after *make* has been executed. To compile the VHDL files for this example into separate libraries, the following order of compilation is required for each of the component elements.
- 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 is 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 have Mentor on your UNIX system and want to check if all environment variables are currently configured, type the following at the UNIX prompt.

```
UNIX>> cd m30_lab_a
UNIX>> make check
```

If the above command line operation returns 'Proceed with compilation', then make all the files using the following.

```
UNIX>> make initial
```

Upon completion of the make, six libraries now exist in the */m30\_lab\_a/libs* directory with the names, *conv\_lib*, *source\_lib*, *proc\_lib*, *sink\_lib*, *clock\_lib*, and *system\_lib*. As these libraries are being generated, proceed to the next section to acquaint yourself with the methodology for capturing design requirements in an executable form.

## 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 */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 2. 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, *emacs* has been selected as the shell command line input. You can choose your own editor as a replacement in each case.

```
UNIX>> emacs ./src/clock/clockgen_ent.vhd &
```

```
editor>> search for tpw and tps and modify to create
the correct waveforms in Figure 2 (ctrl-s tpw) (ctrl-s
tps)
```

```
editor>> save the file (ctrl-x, ctrl-s)
```

Now that the file has been modified, recompile both the entity and architecture files for the clock circuit and simulate. Execute 'make' operations from the */m30\_lab\_a* directory where the makefile is located.

```
UNIX>> make clock_lib
```

You should now have an update of the clock and reset functionality. Next, simulate the design and check the timing for correctness. Listed below are the command line inputs to the version of the tool used to execute the models during the laboratory development. The names, qhmap and qhsim, may change in later versions of the simulator and the user must make modifications to the makefile where necessary.

```
UNIX>> qhmap work ./libs/clock_lib/
```

```
UNIX>> qhsim &
```

You should now be entering the simulator. Select the *clockgen\_ent* entity during elaboration. Within the simulator follow the commands below.

```
QuickHDL>> select 'clockgen_ent' entity from the library
```

```
QuickHDL>> select 'dataflow' architecture
```

```
QHSIM>> wave /*
```

```
QHSIM>> run 200 ns
```

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 4.

*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'*

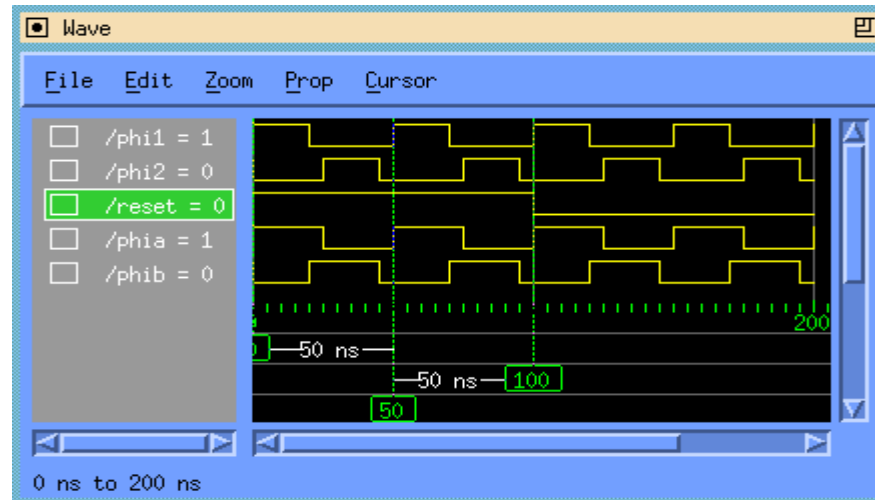
*Processor entity:*

*read\_data\_ack = '0' and buf\_full\_out = '0' and data\_out = tri-state 'Z'*

*Display entity:*

*write\_data\_ack = 0'*





**Figure 4 : Clock Waveform with  $tpw = 20\text{ ns}$  and  $tps = 5\text{ ns}$**

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*

```
editor>> emacs ./src/source/sensor_arc.vhd &
```

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

```
editor>> emacs ./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.

```
editor>> emacs ./src/sink/display_arc.vhd &
```

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

Next, run the full system for two clock periods to observe the output waveform resulting from a system reset. Since the for the clock has been modified, recompile the system configuration files by typing the following.

```
UNIX>> make system_lib
```

You should now be ready to run the simulator.

```
UNIX>> make simulate
```

Load the *system-config* configuration into the design and at the QHSIM prompt execute the *wave.do* file and run the simulator for 110 ns.

```
QuickHDL>> select the 'system-config' configuration
QuickHDL>> select File->Execute command file from the menu
QuickHDL>> select the file 'wave.do' from FILE listing
and execute
QHSIM>> run 110 ns
```

At this point you should now be able to observe the waveform resulting from the system reset. Verify that the output signals of each of the three main components has been properly set. This can be done by placing a cursor at 100 ns and observing the signal values in the left hand window of the waveform viewer.

*5.3. Requirement 3: Complex sensor data is continuously sampled at a 5MHz rate 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-related 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 through the use of 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 within the entity file *sensor\_ent.vhd* and can be overridden by setting these parameters in the system configuration file, *system-config.vhd*.

There is also the requirement for buffering the input data. This can be modeled using array types to represent the buffers. Observe the declaration lines 104-108. A buffer type is defined that has a length specified by the generic parameter *buffer\_size* as well as two buffers, one each for the real and imaginary parts of the sampled data (*i\_buffer\_array* and *q\_buffer\_array*). The buffer size is defined to have a length of 1024 samples. It is defined in this manner 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 'restart design' from the File' pull down menu.

```
QuickHDL>> Choose FILE->Restart Design from the menu
QuickHDL>> Push the button Restart and keep all settings
QHSIM>> run 50000 ns
```

Next we will set a breakpoint in the sensor architecture at line number 129 so that we can observe internal buffer arrays. This can be done by selecting 'view' from the pull down menu and selecting structure within the View'menu. Also do the same for viewing the source code. When the structure of the system appears, select the sensor element.

```
QuickHDL>> Select View->Structure... from the menu
QuickHDL>> Select View->Source... from the same menu
QuickHDL>> Click on sensor: sensor_ent(behavioral) in the
structure window
```

Now the breakpoint can be set in the source window by either clicking on the line number on the left-hand side or by typing the following within the QHSIM window.

```
QHSIM>> bp ./src/source/sensor_arc.vhd 129
```

Continue running the simulation from this point and wait until it reaches the breakpoint. Type run within the QHSIM window.

```
QHSIM>> run 150 ns
```

View the variables within the simulations sensor architecture by selecting the View pull down menu and the Variables... option.

```
QuickHDL>> Select View->Variables... from the menu
```

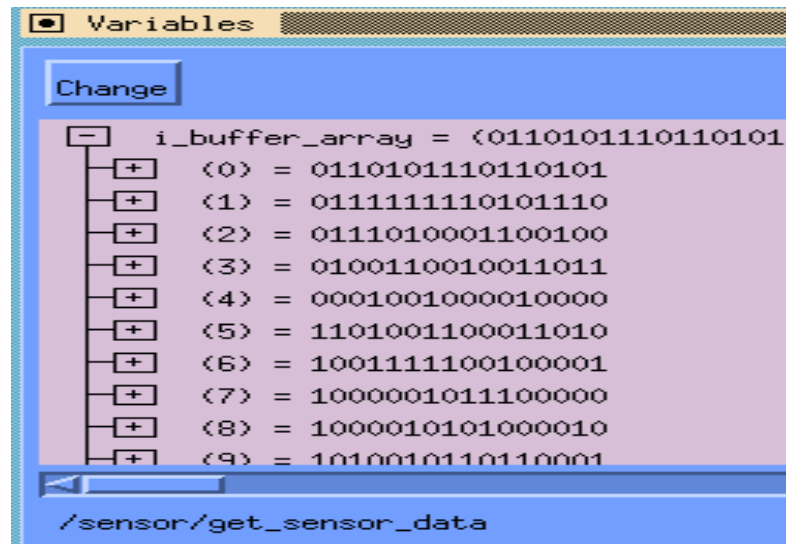
At this point you should see 250 nonzero values withing the I and Q internal buffers of the sensor as shown in Figure 5. Data continues to fill the buffer until it has reached a point where it is ready to send data to the processor. This occurs when 512 samples are read into the buffer (the number of 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.

```
QHSIM>> bd ./src/source/sensor_arc.vhd 129
```

```

QHSIM>> run 52400 ns
Wave>> select Zoom->Range... in the Wave window
Wave>> Start = 102300 ns
Wave>> Stop = 102557 ns

```



**Figure 5 : Sample values contained within the sensor's I input buffer array**

The *buf\_full\_sig* signal becomes active and waits for a response from the processor unit to acknowledge the beginning of data transfer.

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.

```

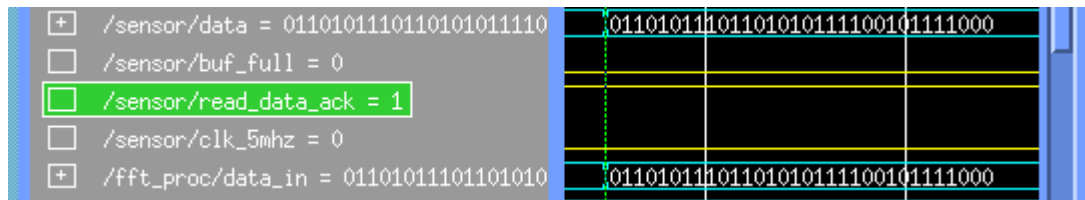
Wave>> select Zoom->Range...
Wave>> Start = 102440 ns
Wave>> Stop = 102480 ns

```

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 6 shows the results that should be seen on the waveform viewer.



**Figure 6 : 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 6.

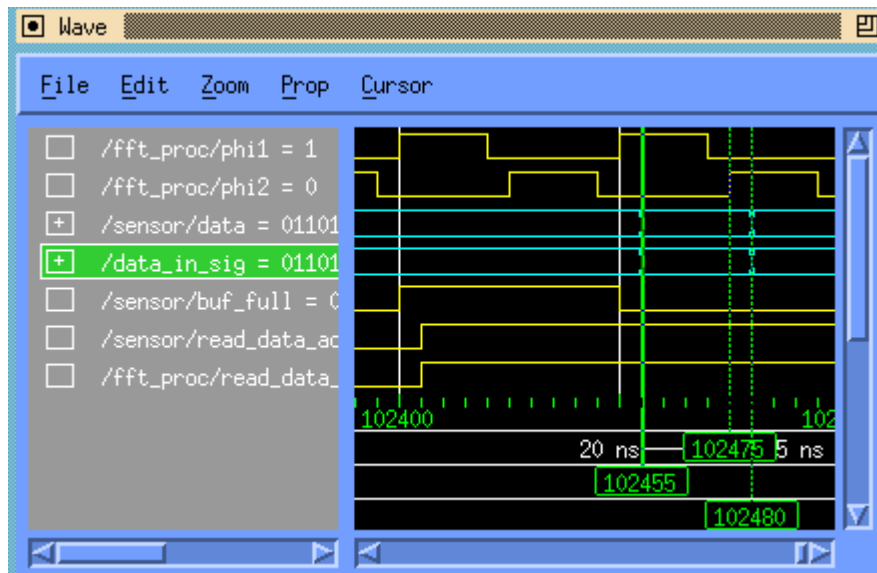
Wave>> select **Zoom->Range...**

Wave>> Start = **102380 ns**

Wave>> Stop = **102560 ns**

*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 7 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 until the input buffer of the processor is full, at which point the processor deactivates the acknowledge signal. This 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 must be active and phase two of the clock 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 will be disabled.



**Figure 7 : 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.

```
QuickHDL>> Choose FILE->Restart Design from the menu
QuickHDL>> Push the button Restart and keep all settings
QHSIM>> run 302 us
```

Zoom into the region that represents the processing. The processing begins when the read\_data\_ack signal from the processor becomes inactive.

```
Wave>> select Zoom->Range...
Wave>> Start = 127000 ns
Wave>> Stop = 149000 ns
```

Observe the point where the last data item is received by the processor 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). How the latency is modeled in the code can be observed by looking 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 frequency 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 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.

```
QuickHDL>> Choose FILE->Restart Design from the menu
QuickHDL>> Push the button Restart and keep all settings
QHSIM>> run 150 us
QuickHDL>> Click on fft_proc: fft_proc_ent(behavioral) in
the structure window
QHSIM>> bp ./src/processor/fft_proc_arc.vhd 166
QuickHDL>> run 150 ns
Wave>> select Zoom->Range...
Wave>> Start = 148000 ns
Wave>> Stop = 148200 ns
```

In the 'variables' window of the simulator look at *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. Observe the code segments that are doing 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*.

- 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.