Digital Signal Processing for Analog Input

Arnav Agharwal Saurabh Gupta

April 25, 2009

Final Report

1 Objective

The object of the project was to implement a *Fast Fourier Transform*. We implemented the *Radix 2 FFT Butterfly algorithm* which works in order $n * log(n)$ time.

We choose this particular problem as calculation of the Frequency Spectrum would allow us to build *Signal Processing Utilities*. Moreover, the project also invloved working outside the FPGA board, in the form of *ADC and DAC interfacing* which provided us a taste of hardware design outside the FPGA board.

2 Detailed Module Description

2.1 Overall Input Output Module

Input: $clk\_process, clk\_sample, reset, writeEnbl, Data\_from\_ADC[7:0]$, $Read\_Address\_from\_FFT[2:0], Write\_Address\_from\_FFT[2:0]$, $Write\_Data\_from\_FFT[7:0]$

Output: $Data\_to\_DAC[7:0], Read\_Data\_for\_FFT[7:0], countVal[2:0]$.

We implement a Buffered input output system, wherein we have 2 buffers at both the input and the output. The module writes the input received from the ADC in Buffer 1, the data already contained in Buffer 2 is given to the FFT module for processing. Once this data ion Buffer 2 is processed, the FFT module writes it into Buffer 3 while the Buffer 4 is being displayed by the DAC onto the output. Once this cycle is over the Buffers 1 and 2 are switched and Buffers 3 and 4 are switched. This module does this switching.

The module samples the ADC input at every sample clock edge.

Since we implement a 8 pt FFT, the switching happens after every 8 sample clocks. Thus, the module is implemented using a 4 bit counter clokced by the sampling clock. The most significant bit of this counter is used for switching and the 3 least significant bits are used to specify which address to read and write to in the Buffers handling input Output with the ADC and the DAC.

The block diagram for this module is shown in Figure 1

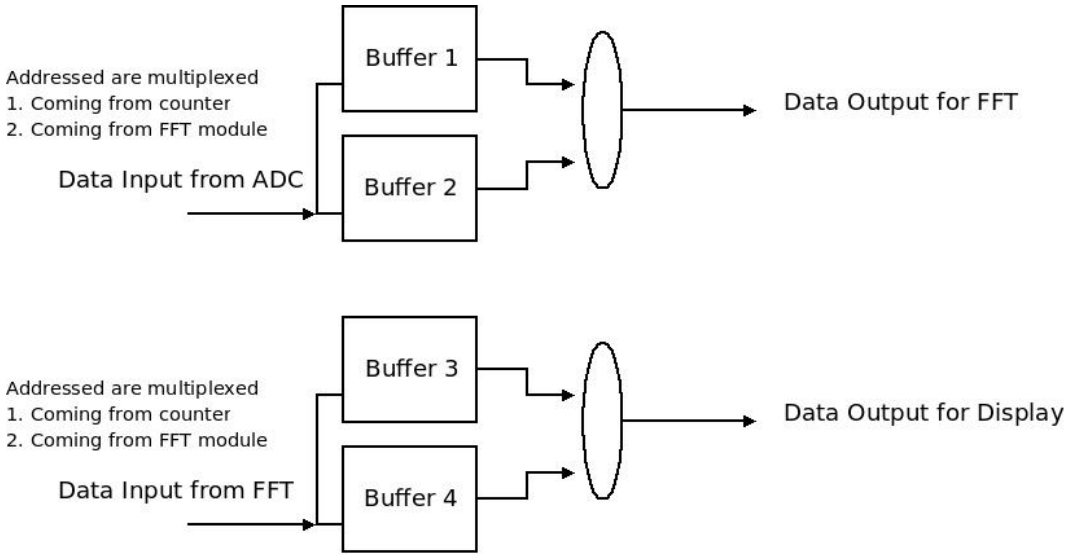

Figure 1: Figure Showing Block Diagram for Overall Input Output Module

2.2 Internal Control FSM

Input: $start\_FFT, eoc\_DE, eol, ready\_Butterfly, reset, clock$

Output: $ready\_FFT, src, reset\_DE, start\_DE, freeze\_Loop, reset\_Loop, le\_Data, N[7:0], start\_Butterfly$

This FSM is the controller co-ordinating the process of data exchange between the internal modules *Data Exchange*, *Loop* and *Butterfly*. The state machine is triggered when the $start\_FFT$ signal goes high. It cycles throught 8 internal states, performing specific functions at every state. The 8 states are as follows:

Idle This is the Idle state for the controller. It waits for the $start_FFT$ signal to go high and resets all internal modules (excluding *Internal Regs* which are reset externally when the overall machine is reset).

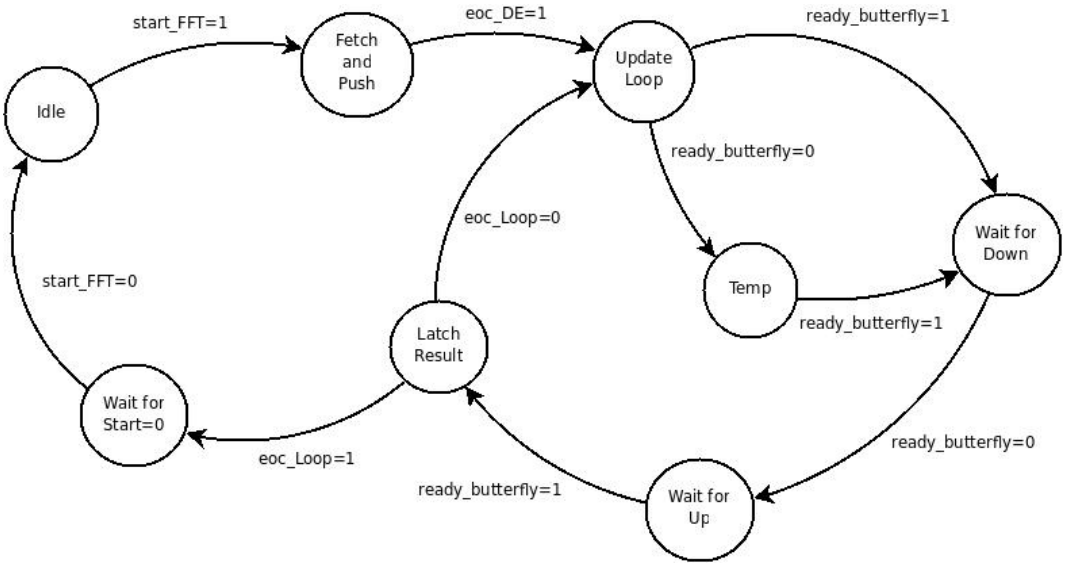

Figure 2: Figure showing Finite State Machine for the Internal Control

Fetch and Push The *Data Exchange module* is triggered in thie state. Data is fetched from the external register set andand data from the previous calculation is pushed into the second external register set.

Update Loop In this state, one iteration of the loop takes place.

Temp This is a temporary waiting state for the $ready\_Butterfly$ signal to go high in case the *Butterfly Module* is busy.

Wait for 0 This state triggers the *Butterfly Module* and waits for the $ready\_Butterfly$ signal to go low.

Wait for 1 This waits for the $ready\_Butterfly$ signal to go high which indicates eoc for the *Butterfly Module*.

Latch Result This state latches the result of computation of the *Butterfly Module* into the *Internal Registers* and checks for the end of the loop.

Wait for start0 Waits for the $start\_FFT$ signal to go low (spec requirement of the topmost FSM).

2.3 Data Exchange Module

Input: $Reset\_DE, Start\_DE, clk$

Output: $addr[2:0], eoc\_DE, write\_enbl, ce\_out\_DE(dummy)$

The purpose of this module is to store the data input from the *ADC output* into registers for calculating their FFT and pushing the results of the calculation for the last input sequence. Internally, it consists of an elementary *state machine* implemented using a 3-bit counter that controls *Internal Registers*, which is a data storage module. The process of writing inputs to the *Internal Registers* and writing the calculated results to output registers is done concurrently taking *1 clock cycle per register* and overall *8 clock cycles* in order to minimise the total number of clock cycles involved in the computation of the FFT.

The inputs $Start\_DE$ and $Reset\_DE$ trigger the data exchange process and reset the module respectively. The *write_enbl* signal remains high so long as the data exchange is in progress. It acts as a Write Enable signal for the external registers that store the final calculated value so long as the exchange is in progress. The eoc_DE signal becomes high when the data exchange process is over and goes low again when the module is reset or exchange is in progress. The $addr[2:0]$ signal addresses both the *Internal Registers* and the external registers.

Like any other state machine, this module waits for the $Start\_DE$ signal to go high before it begins computation.

2.4 Loop Module

Input: $freeze\_Loop, N[7:0], reset\_Loop$

Output: $EOL, i[7:0], B[7:0], addr0[7:0], addr1[7:0]$

This module is responsible for running the loop for calculating the FFT. This module has been designed in a manner such that the loop takes exactly *n log(n) clock cycles* which evaluates to *24 clock cycles* for the current 8-point FFT implementation. It is essentially an automated datapath with very little controlling FSM involved.

It has the following three major submodules:

Address offset (i) This cycles through 0 to the current block size B, getting incremented by 1 every time.

Block size (B) This cycles from 1 to N/2, getting doubled every time.

Base address (addr) This cycles from 0 to B-1, getting incremented by 1 every time. It has 2 multi-bit outputs, $addr0$ and $addr1$, which form the base addresses for the 2 registers on which a *Butterfly operation* needs to be applied.

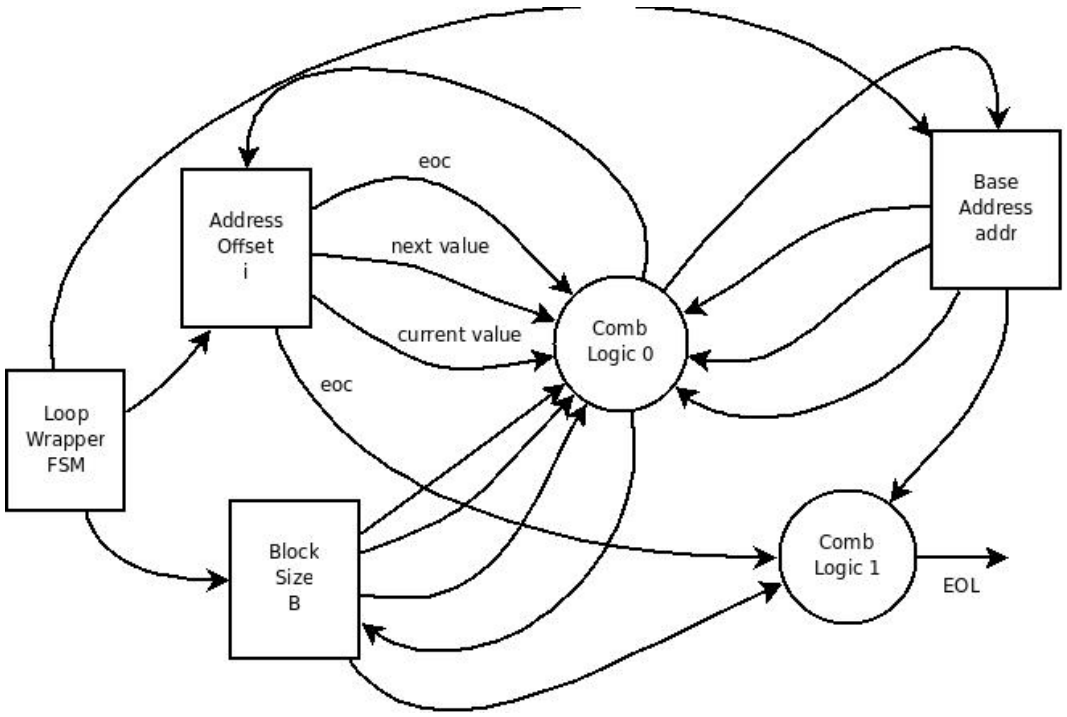

Figure 3: Figure showing internal block diagram of the Loop Module

The sum of i with $addr0$ and $addr1$ gives the address of the 2 above-mentioned registers.

These modules interact with each other using simple combinational logic. It also has a trivial internal FSM by the name of Loop Wrapper that essentially introduces a delay of one clock cycle to correct the timing of the loop.

The complexity of the design arises so that the loop terminates in exactly $nlog(n)$ clock cycles, which requires some amount of look-ahead for the internal variables of the loop. An alternate (and possibly simpler) implementation would have involved a simple Datapath and Controller approach which would have introduced a multiplication factor of 2 in the clock cycles required. This would have considerably slowed down the overall implementation as the number of clock cycles for calculating a FFT would have required twice the number of cycles, hampering the scalability of the design.

The machine is reset when the $reset\_Loop$ signal is high. The execution of the loop begins automatically when this signal goes down. The loop can be halted by making the $freeze\_Loop$ signal high.

The input $N[7:0]$ is the value of N for an N-point FFT operation. This makes the entire loop scalable upto implementing a 64-point FFT.

The EOL signal goes high with the final iteration of the loop indicating the end of computation. The signals i and B are required to lookup the *sine* and *cosine* values. Signals $addr0$ and $addr1$ are used for addressing the internal registers for carrying out one *Butterfly operation*.

2.5 Butterfly module

Input: $StartSignal, Re\_a[15:0], Im\_a[15:0], Re\_b[15:0], Im\_b[15:0], Re\_w[15:0], Im\_w[15:0], clk\_process, reset$

Output: $ReadySignal, Re\_ans1[15:0], Im\_ans1[15:0], Re\_ans2[15:0], Im\_ans2[15:0], overflow$

This module takes in as input the Complex Numbers A, B and W and produces as output the values of $Ans1$ and $Ans2$ where these are defined as $Ans1 = A + W * B$ and $Ans2 = A - W * B$. Since A, B and W are all complex numbers, there are a total of 4 multiplications and 8 additions that need to be carried out.

The implementation uses one *18 bit multiplier* and one *16 bit multiplier* to produce the output in 9 cycles (One cycle for each of the additions). The multiplier and the adder form the *critical path* of this module. They may have been pipelined, but that would have increased the number of overall cycles, and since the FPGA boards offer only a frequency upto 2Mhz, the overall design would have been slower, if the number of cycles increased as is the case with a pipelined design.

The module implements a simple state machine which waits in an idle state for a start signal and on receiveing a start signal transits through a series of states doing one elementary operation in each state and finally ending in the idle state again.

The state diagram and block diagram are shown in Figure 4 and Figure 5.

2.6 Internal Regs Module

Input: $addr0\_data\_in[31:0], addr1\_data\_in[31:0], addr0[2:0], addr1[2:0], CE, reset, clk$

Output: $addr0\_data\_out[31:0], addr1\_data\_out[31:0]$

This is purely a *data storage* module. It has 8 internal modules, each of which stores a 16-bit * 16-bit complex number. The first 16 bits of the input and output terminals store the Real part and the last 16 store the

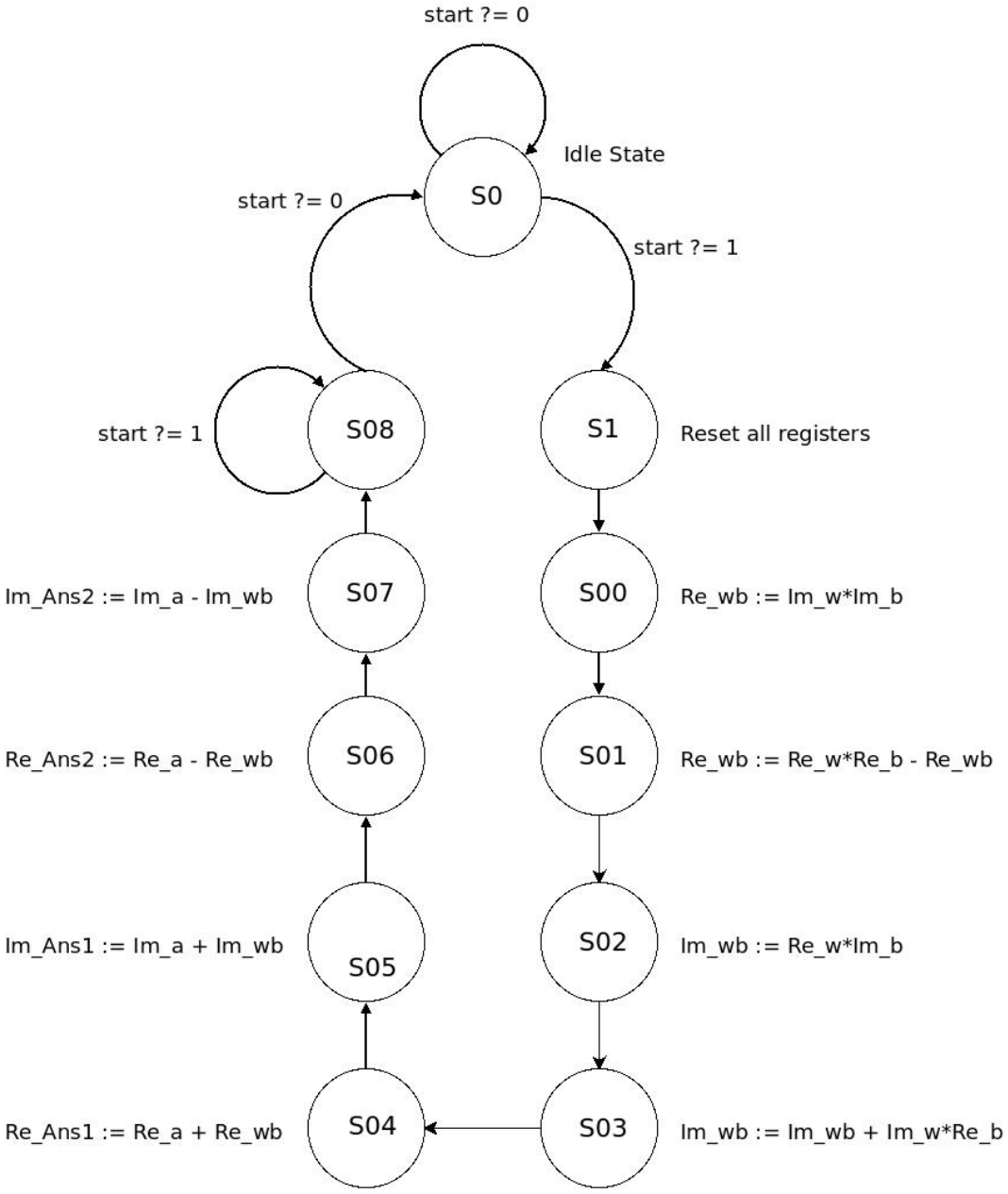

Figure 4: Figure Showing State Machine for Butterfly Module

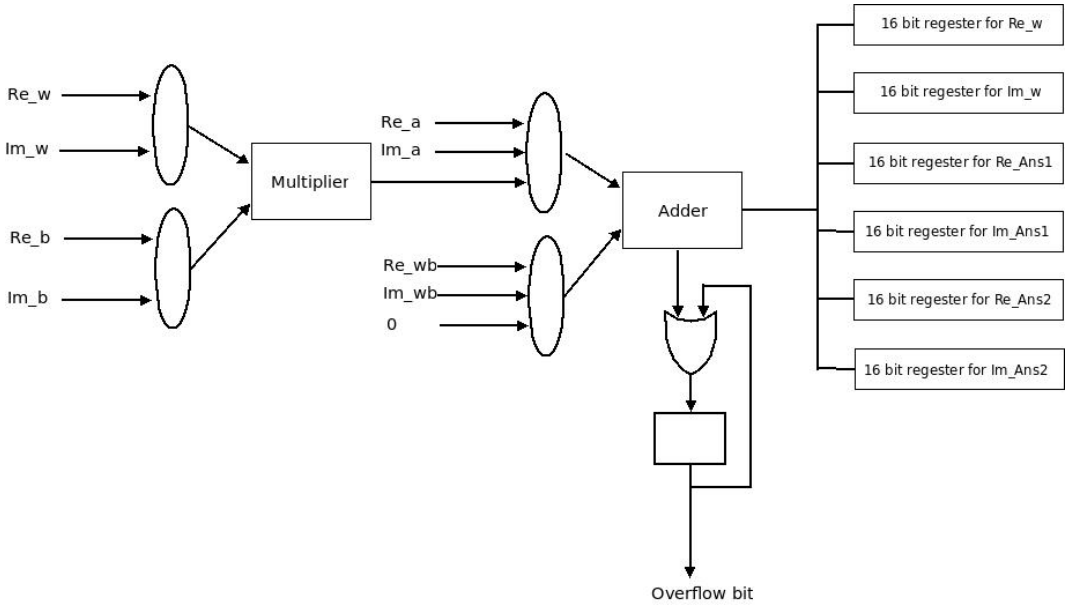

Figure 5: Figure Showing Block Diagram for Butterfly Module

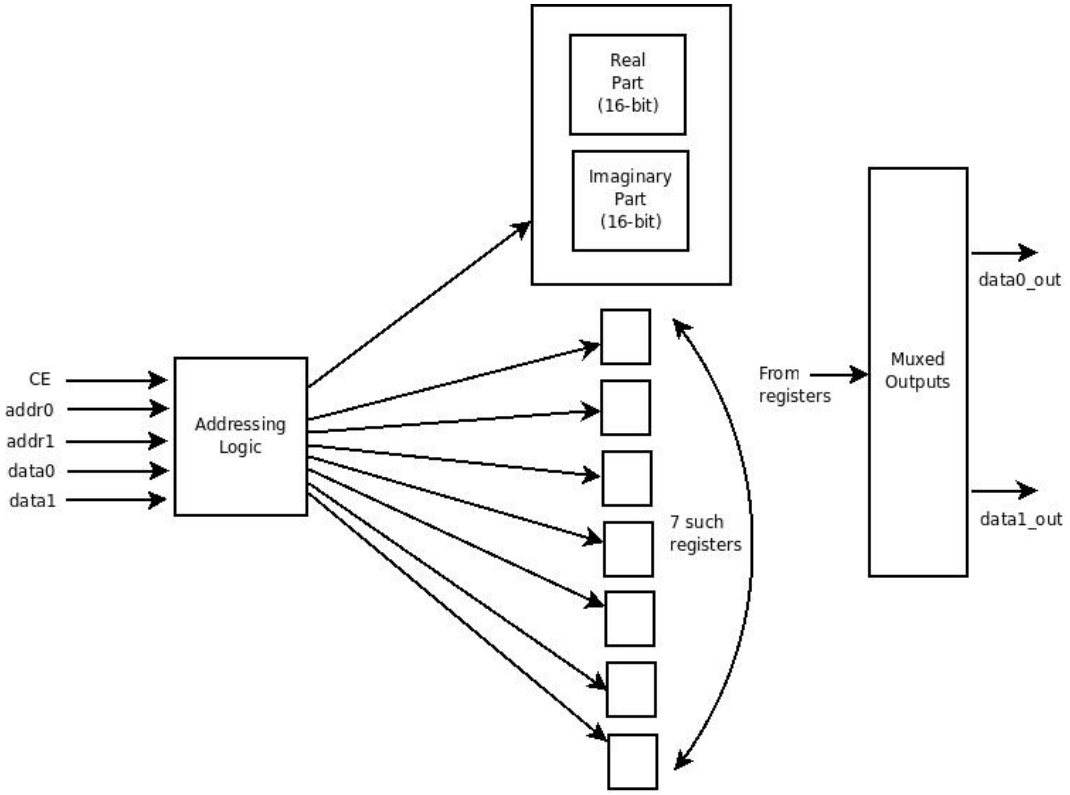

Figure 6: Figure showing internal block diagram of the Internal Regs Module

Imaginary part. It is specially designed to allow addressing of 2 registers at the same time for providing the inputs to the *Butterfly Module*. The 2 output registers are addressed by $addr0$ and $addr1$ and both registers can be written to simultaneously in the same clock cycle.

Data is written into the registers synchronously only if the CE signal is high. All interal registers are synchronously reset to 0 when the $reset$ signal goes high.

Data path for this module is shown in Figure 6

2.7 Display Module (for Display on LED Matrix)

Input: $clk\_sample, clk\_display, DAC\_data, countVal$

Output: $Row\_output\_for\_the\_LED\_matrix$,
$Coloumn\_output\_for\_the\_LED\_Matrix$

The module takes in the input from the Overall Input Output Module and displays it on the LED matrix as a histogram of frequencies.

Since sampling may be done at a sufficiently high rate and 8 pts may not be able to capture the entire frequency content of the signal, so this module latches to the frequency output after a fixed time interval so as to produce a stable output on the LED matrix.

The module has three sub modules.

Module 1 Module 1 converts the input 8 bit integer value into an appropriate histogram bits. (eg 0100 0000 gets converted to 11110000)

Module 2 Module 2 latches to these values after a fixed duration of time, and output 64 bit values to be given to Module 3 for display on the matrix

Module 3 Module 3 takes in as input the 64 bit values from Module 2 and displays them on the LED matrix, by appropriately multiplexing the output by scanning the coloumns and exciting the row inputs of the LED matrix.

3 The Testbench

3.1 Inputs

$clock, reset$

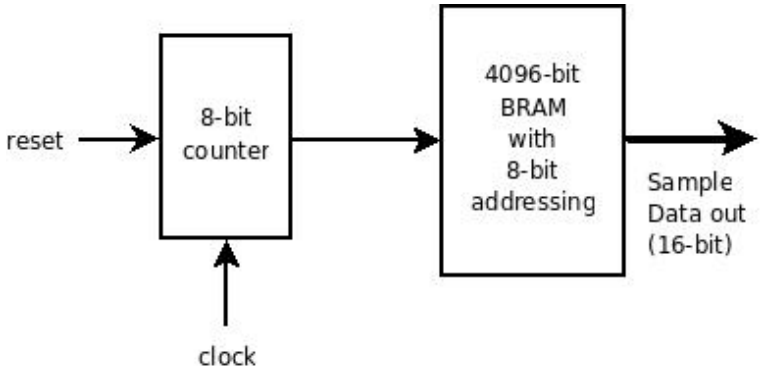

Figure 7: Figure showing the internals of the Testbench

3.2 Outputs

$sampled\_data[15:0]$

3.3 Description

The Testbench is utilised for providing a set of test vectors to the overall FFT module in the absence of the ADC interfacing. It is in the form of a 4096-bit BRAM with 8-bit addressing, addressed by an 8-bit counter. It is initialised with 8-bit digitised samples of any desired mathematical function sampled at 256 equally spaced intervals of time. The initialisation of the BRAM contents is achieved by using the *data2mem* tool packaged with the Xilinx ISE. The memory files for this purpose were generated using JAVA and SML code.

3.4 Functioning

The *reset* signal, when high, resets the BRAM *data_out* address to 0. This signal is useful to synchronize the testbench with the FFT module for testing purposes. When the *reset* signal is low, the testbench cycles through the prefed values updating the output at every clock cycle.

The lowest 8 bits of the $sampled\_data[15:0]$ signal form the desired sample data which emulate outputs of the 8-bit ADC.

4 Utilisation Report

4.1 Device Utilisation Summary

```
Selected Device : v600ehq240-6
```

Number of Slices:	949	out of	6912	13%
Number of Slice Flip Flops:	504	out of	13824	3%
Number of 4 input LUTs:	1314	out of	13824	9%
Number used as logic:	1282			
Number used as RAMs:	32			
Number of IOs:	34			
Number of bonded IOBs:	34	out of	158	21%
Number of BRAMs:	1	out of	72	1%
Number of GCLKs:	1	out of	4	25%

4.2 Timing Summary

```
Speed Grade: -6

   Minimum period: 33.411ns (Maximum Frequency: 29.930MHz)
   Minimum input arrival time before clock: 11.067ns
   Maximum output required time after clock: 11.249ns
   Maximum combinational path delay: No path found
```

Critical Path is the *Combinational Multiplier* that has been used in the *butterfly module.*

5 Implementation Status

5.1 Interfacing the DAC

Interfacing DAC with the FPGA board. We spent 4 hours figuring out detasils from the scratchy documentation that was available for the DAC interface, but could not figure out anything substantial. Whatever little that we figured and tried did not behave as expected. **Failed**

5.2 Design of Individual Modules

We split the implementation into two parts which could be pursued independently. The modules were independently developed and were tested successfully on the simulator. **Successful**.

5.3 Interfacing

The individual models developed were interfaced. There were some issues with the interfacing in the sense of bus widths, eoc signals, whcih were resolved and the modules were properly interfaced. **Successful**.

5.4 Testing on Simulator

It was difficult to provide input as the testbench would provide and hence we used some trivial input to test the FFT and butterfly module and they worked correctly. **Successful**.

5.5 On Board Testing

On board testing with test benches with various types of inputs. The output was observed on the LED matrix. For some of the inputs for which output was expected to be nice and uniform were tested. After some struggle with faulty boards, the output obtained was correct. **Partially Successful**.

5.6 Interfacing with ADC

We are currently trying interfacing with of the FPGA board with the ADC. We have figured out most things but some details still remain to be sorted out. **In Progress**.

5.7 Remaining Work

1. Interfacing with the ADC, and testing on real input.