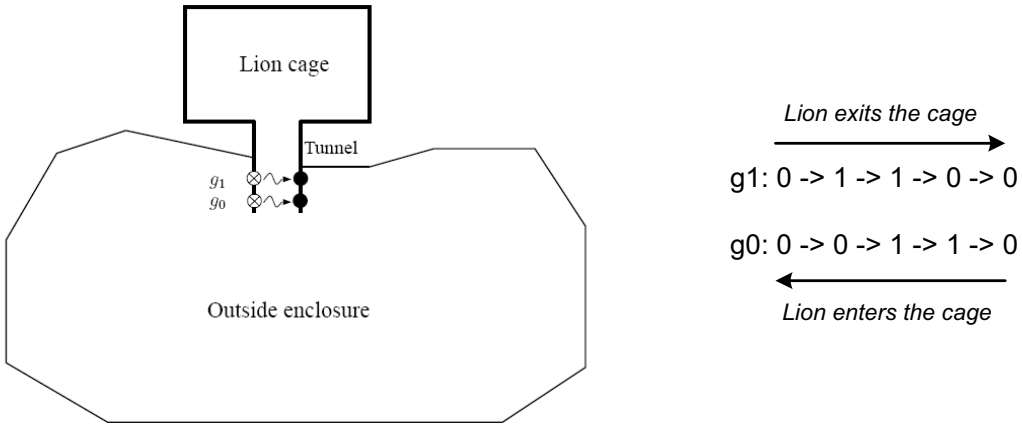
Table of contents

1. Assignment 1: The Lion Cage

1.1. Assignment specification



(a) *Lion Cage physical construction and the placement of two photo sensors*

(b) *Corresponding sensors' output sequence upon events*

Figure 1. The Lion Cage system specification

Figure 1(a) above shows the system specification for this assignment. A smart system is supposed to be designed with using the installed photo sensors in the lion cage tunnel to detect the movements of lions between the lion cage and outside enclosure. A danger light as a system output indicator should turn on when at least one lion enters the tunnel from the cage side, and lights off when both lions are inside the cage.

1.2. System implementation

The entire system is controlled by using a finite state machine (Mealy machine) as shown in Figure 2. During the system development, two basic properties had been considered into the design:

(1) Generic: The system structure should not be restricted with the number of lions under detection.

(2) Safety: With the aid of initial state, system will be properly initialized upon system reset. The only assumption here is made that all lions are staying together (does not matter where they are).

Overall, the finite state machine consists of 7 states with one 2-bit counter involved. The counter is needed here to watch out the number of lions entering into the cage in order to determine the status of output light.

State 0 – "lionOut_ini": This is an initial state listens to the system reset. State transitions are made based on the different sensor value patterns. With "g1 = 0 & g0 = 1", the lion's moving direction - towards lion cage is detected, so the case of all lions

are initially outside enclosure can be deduced, consequently the lion counter value is set to 0. The opposite case happens when "g1 = 1 & g0 = 0", a moving direction of exiting lion cage is sensed, and all lions are initially inside the lion cage is deduced, where the corresponding lion counter value is then set to 2.

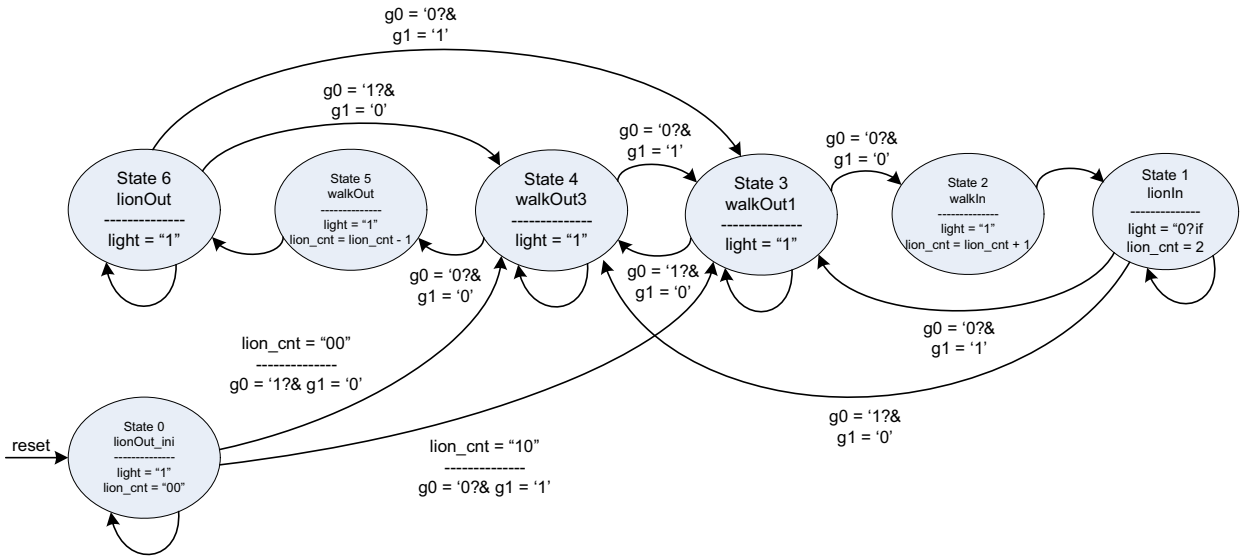


Figure 2. Finite State Machine of the Lion Cage System

State 1 & 6 – "lionIn" & "lionOut": These are two ending transition states, which indicate a lion has passed through the tunnel, enters either to the lion cage side or outside enclosure. Two different state transitions could be triggered there, since a lion may move back in direction or a new lion may follow it moving in the same direction.

State 2 & 5 – "walkIn" & "walkOut": These are two un-conditional states, which are only used to update the lion counter values. In the direction of entering lion cage – state 2, counter value is incremented; in contrast, counter value decrements when lion enters the outside enclosure – state 5.

State 3 & 4 – "walkOut1" & "walkOut3": These are two intermediate transition states. State transition is made upon the sensor value changes.

Output danger light indicator turns off only when the lion counter is equal to 2 and the current state is in state 1 – "lionIn".

1.3. Conclusion

Finally, system works as expected. The captured ModelSim simulation result is shown in Figure 3.

During the lab period of this assignment, the only concerning I had was the VHDL code translation from the conventional programming style to the use of structured VHDL programming style. In fact it is not hard to do the code changing, and there wouldn't be any difference in terms of functionality. However, it is always hard to do something which you do not see the benefits by doing it.

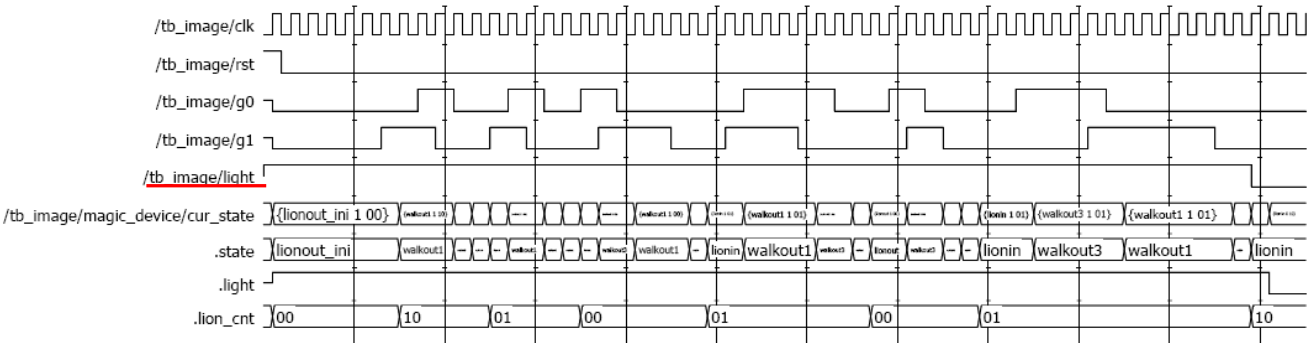
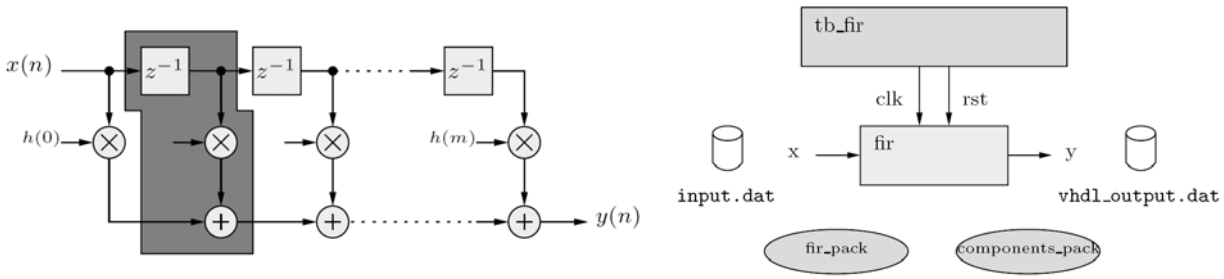


Figure 3. ModelSim simulation

2. Assignment 2: The Direct Mapped FIR filter

2.1. Assignment specification



(a) *FIR filter in direct form* *(b) VHDL file structure and test environment*

Figure 4. The direct mapped FIR filter assignment specification

In this assignment a small 7th order FIR filter in direct mapped form is going to be implemented in VHDL. Under this construction, the FIR filter is highly scalable due to the property of similarity, as illustrated in the shaded area in Figure 4(a).

2.2. Wordlength consideration

There are two ways of implementing the direct mapped FIR filter. One is to keep the value precisions during the system calculation in order to avoid data overflow. And the other way is to introduce data truncation inside the system.

Obviously, there would not be any harm to data results in the first scheme, as all the value details are kept through the system. However, this increases the required wordlength, which is further costly in hardware. Referring to Figure 4(a), wordlength needed after multiplications are fixed, but increases through the following adders by the factor of $log_2(N)$, where N is the number of taps (stages) in the system. Considering the FIR filter in this assignment, the full precision output wordlength is,

$$W_{Out} = W_{In} + W_h + \lceil \log_2(N) \rceil = W_{In} + W_h + \lceil \log_2(M+1) \rceil = 8 + 6 + \lceil \log_2(7+1) \rceil = 17$$

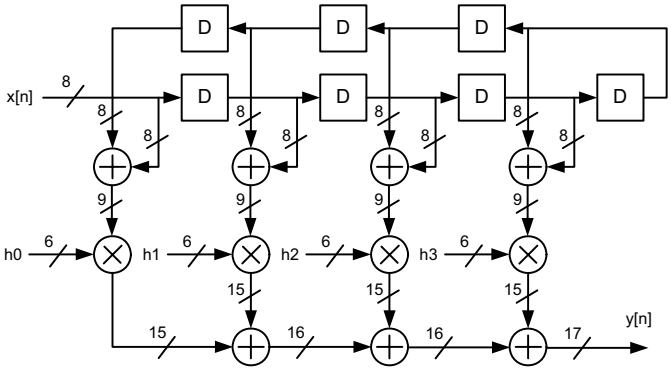
where M is the filter order.

In second scheme, data truncation can be further implemented in two ways, either to do data truncation on the final result only, or truncate data after each multiplication. Again the first method gives better SNR than the second one, due to the only place of loosing data precision. Whereas the second method gains in the required hardware area, since all the adders are smaller in size.

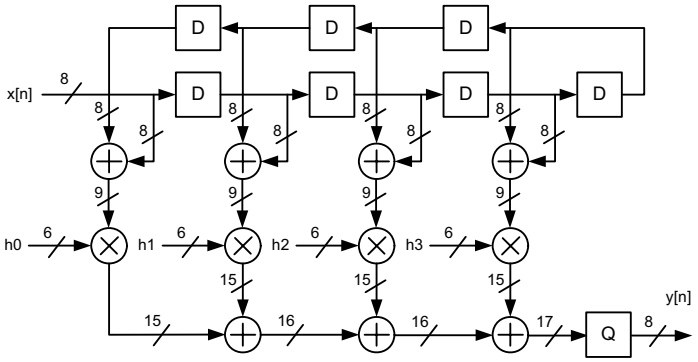
To do the data truncation, two different schemes can be used, data quantization (floor) and rounding. Since rounding data is the same as finding its closest value, so it is expected to have better SNR than data quantization. However, it is easier to achieve data quantization than rounding, since there is no extra computation needed.

2.3. System implementation

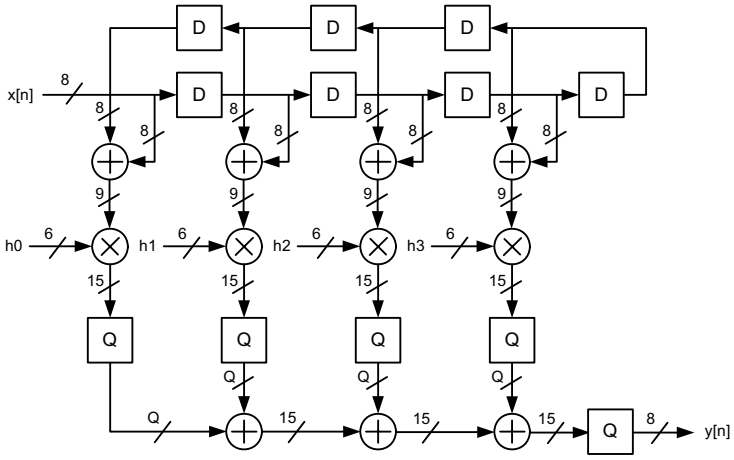
All the implementation schemes mentioned in the previous section have been experimented in this assignment. By investigating the filter impulse response coefficients, linear phase property is utilized in the filter construction, which saves a considerable number of multipliers in hardware.



(a) Method 1 – Full precision computation



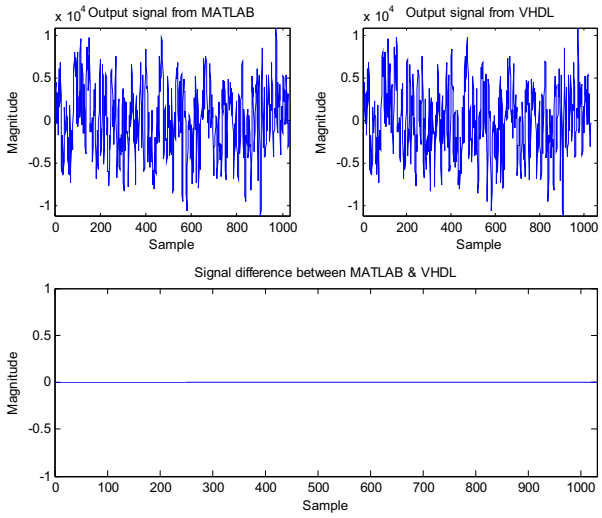
(b) Method 2 – Data truncation on the output



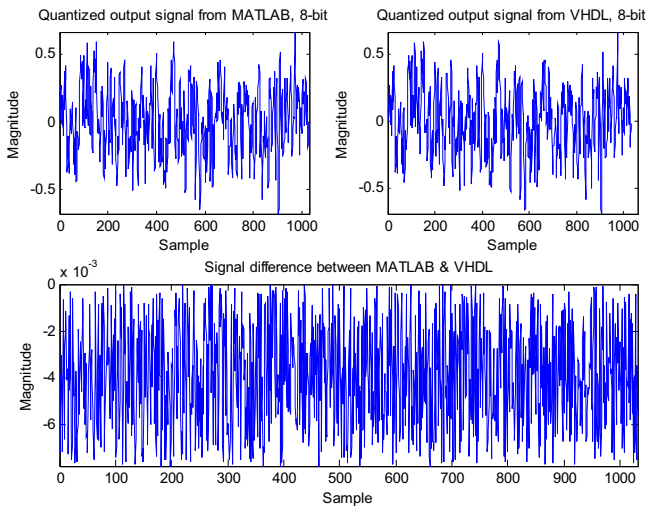
(c) Method 3 – Data truncations after each multiplication & on the final result

Figure 5. The direct mapped FIR filter structures

2.4. Output accuracy verification



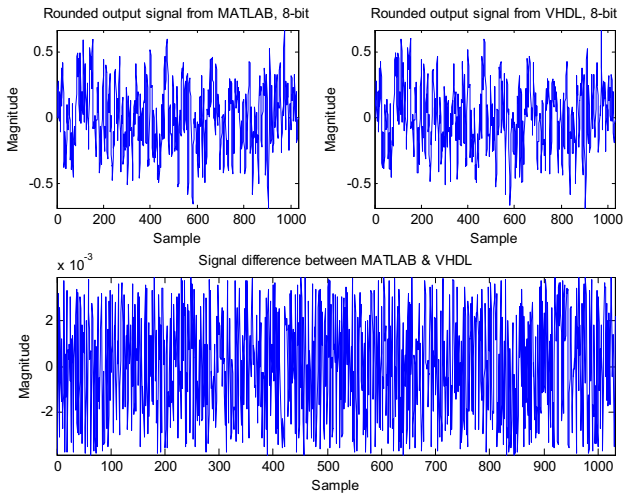
(a) Result from method 1 – Full precision computation



(b) Result from method 2a – 8-bit quantized calculation

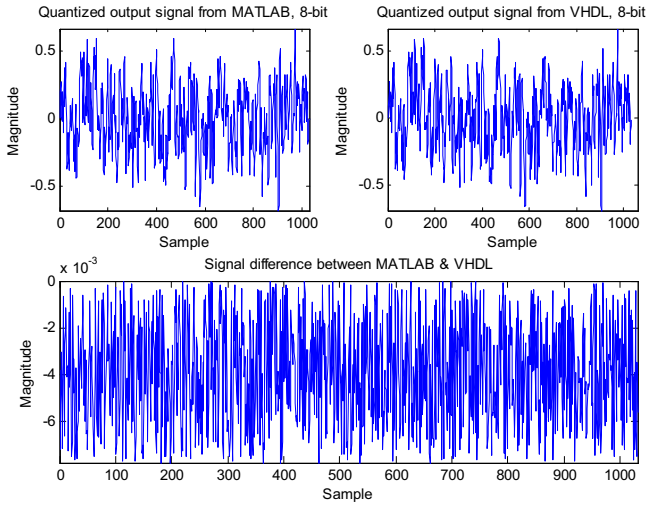
$$SQNR = 10 \cdot \log_{10}\left(\frac{P_{signal}}{P_{noise}}\right)$$

$$= 10 \cdot \log_{10}\left(\frac{sum(y.\hat{}\,2)}{sum(y\_diff.\hat{}\,2)}\right) \approx 34.7444[dB]$$



(b) Result from method 2b – 8-bit rounded calculation (Common rounding scheme)

$$SQNR \approx 40.507[dB]$$



(c) Result from method 3 – 10-bit quantization after each multiplication

$$SQNR \approx 34.7444[dB]$$

Figure 6. Calculation accuracy comparisons between MATLAB calculation and filtering in VHDL

There are several points can be exploited from records shown in Figure 6. First of all, results from method 2a and 2b clearly show that, a higher SNR is achieved by

using data rounding than the simple data quantization. Secondly, figure (c) exhibits a similar result as obtained by using method 2a, which means reducing data precision inside the system after each multiplication from 15-bit down to 10-bit, will not reduce the output accuracy in this case. This is because all the filter impulse response coefficients are small here, so removing the redundant data bits during the calculation will not harm the system.

In order to experience more on data truncations, two series of simulations are carried out in this assignment. Firstly, by using the filter structure in method 2a, output SNR respect to output data bits is investigated, as shown in Table 1.

Table 1. SNR respect to output data bits in method 2a – Quantization on the output.

	6-bit	***8-bit***	***10-bit***	***14-bit***
SNR [dB]	22.6774	34.7444	46.9254	75.085

Result verifies the famous SNR expression stated that,

$$\therefore \underline{\underline{SQNR}} = 20 \cdot \log_{10}(2) \cdot n + 1.76dB \underline{\underline{\cong 6.02 \cdot n + 1.76[dB]}} \qquad \textit{eq 1.}$$

Which represents a 6dB SQNR (SNR) change will be introduced into the system whenever an additional bit devoted to or removed from quantization.

Secondly, output SNR influenced by different quantization resolutions after each multiplication is examined by using filter structure in method 3, as shown in Table 2.

Table 2. SNR respect to quantization resolutions during system calculation in method 3 – Quantization following by each multiplication.

	7-bit	***8-bit***	***9-bit***	***10-bit***	***15-bit***
SNR [dB]	23.6721	27.6785	30.7054	34.7444	34.7444

2.5. Conclusion

Data truncation investigation was a main focus in this assignment. Different data truncation places and several truncation schemes are carried out during the FIR filter implementation.

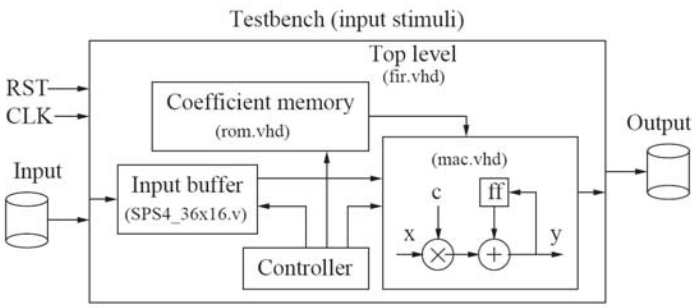
One careless mistake during the assignment was the computation of SNR in MATLAB. Since the input data series has the DC bias value different from 0, so,

$$SQNR = 10 \cdot \log_{10}\left(\frac{P_{signal}}{P_{noise}}\right) \neq 10 \cdot \log_{10}\left(\frac{\sigma^2_{signal}}{\sigma^2_{noise}}\right) \qquad \textit{eq 2}$$

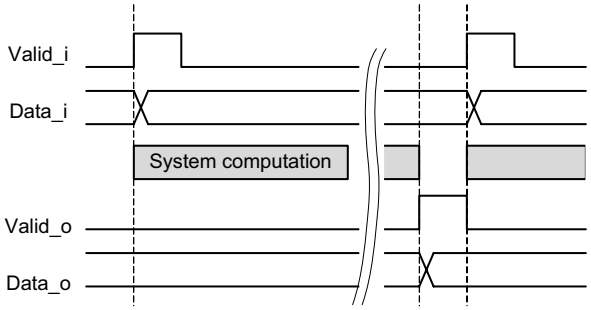
Due to the wrong expression used in the beginning, SNR in quantization was higher than the data rounding; therefore I was confused for a long time. But this is a good point to remind.

3. Assignment 3: The Time Multiplexed FIR filter

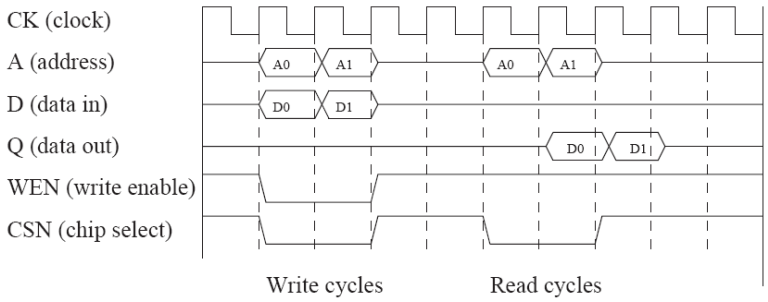
3.1. Assignment specification



(a) Block diagram of time Multiplexed FIR filter



(b) Input/Output synchronization protocol



(c) SRAM control signals and timing chart

Figure 7. The time multiplexed FIR filter assignment specification

To achieve the same filter function, FIR filter can be implemented by using the minimum hardware resources, where the reduced area comes at the cost of system computation time, therefore power dissipation. In this assignment, a 35th order time multiplexed FIR filter is going to be implemented in VHDL.

Figure 7(a) shows the block diagram and data flow of the time-multiplexed FIR filter. Input data are sequentially clocked into the system, where they are buffered by a SRAM module. All 36 filter impulse response coefficients are stored inside the ROM, and a small ALU is used to do the necessary computations. Controller is responsible for processing synchronizations and administrates the collaborations between different sub-modules.

Figure 7(b) describes the engaged communication protocol, which is used to synchronize data input/output and system computation.

3.2. Wordlength consideration

No data truncation is considered in this assignment, so precision calculation is kept through the system. By using the same equation applied in previous assignment, output data wordlength is obtained as,

$$W_{Out} = W_{In} + W_h + \log_2(N) = 16 + 16 + \lceil \log_2(36) \rceil = 38$$

Table 3. Wordlength requirements

	W_{Input}	$W_{Coeff.}$	$W_{Mult.}$	$W_{Add.}$	$W_{Mac\_Reg.}$	W_{Output}
Wordlenth	16	16	32	38	38	$38-15=23$

Since the filter coefficients are 2's complement represented, so the output data should be scaled down by amount of 2^{15}, which is achieved by right shifting 15-bit on the output data.

3.3. System implementation

Since all the corresponding data samples have to be present at the right moment in order to do the necessary MAC arithmetic, so one of the challenges in this assignment is to control the data flow inside system. To ease the data handling, the SRAM module is configured as a circular buffer, as illustrated in Figure 8.

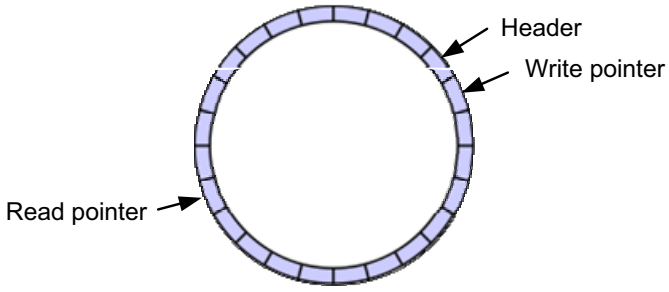


Figure 8. Circular buffer with its pointers

Three pointers are accompanied there. "Write pointer" is used to remember where the new incoming data should be stored at. Besides, it is also used as a tail pointer to indicate the end of processing elements for current iteration. "Header", as the name implies, points at the beginning of the circular buffer. "Read pointer" is used for reading data out from buffer, starts from the "Header" until tail point indicated by the "Write pointer". In transient state, where the circular buffer has not been circulated back for the initial 36 data samples, the "Header" will stay at the beginning without any movements. In steady state, "Header" is incremented by one position at a time whenever a new data sample comes in.

Figure 9 on next page depicts the architecture of time multiplexed FIR filter, where all the sub-modules are central controlled by a FSM. "valid" signal on MAC module is used for enabling the MAC register to store new data; "load" signal controls the output register in MAC unit to clock out new result in the end of each processing iteration, which avoids the intermediate results being present on the output data bus while each filtering iteration is still in progress; "clear" signal is used to initialize the MAC register in the beginning of each processing iteration.

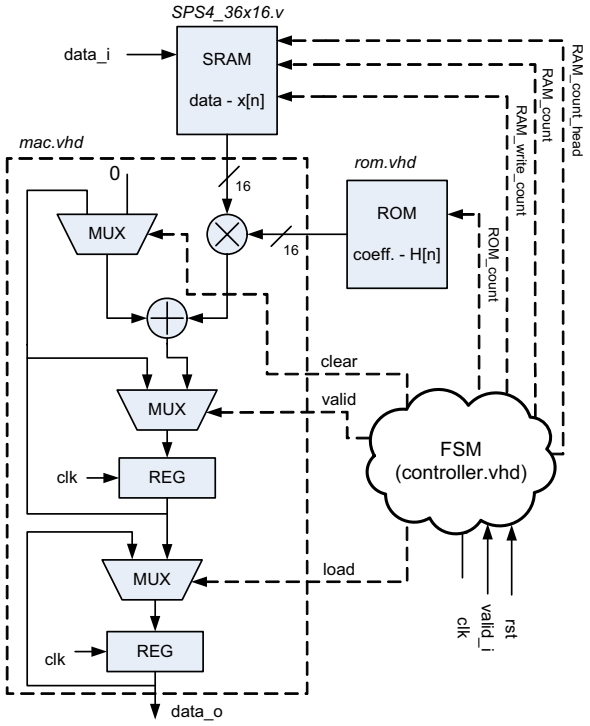


Figure 9. Architecture of time multiplexed FIR filter

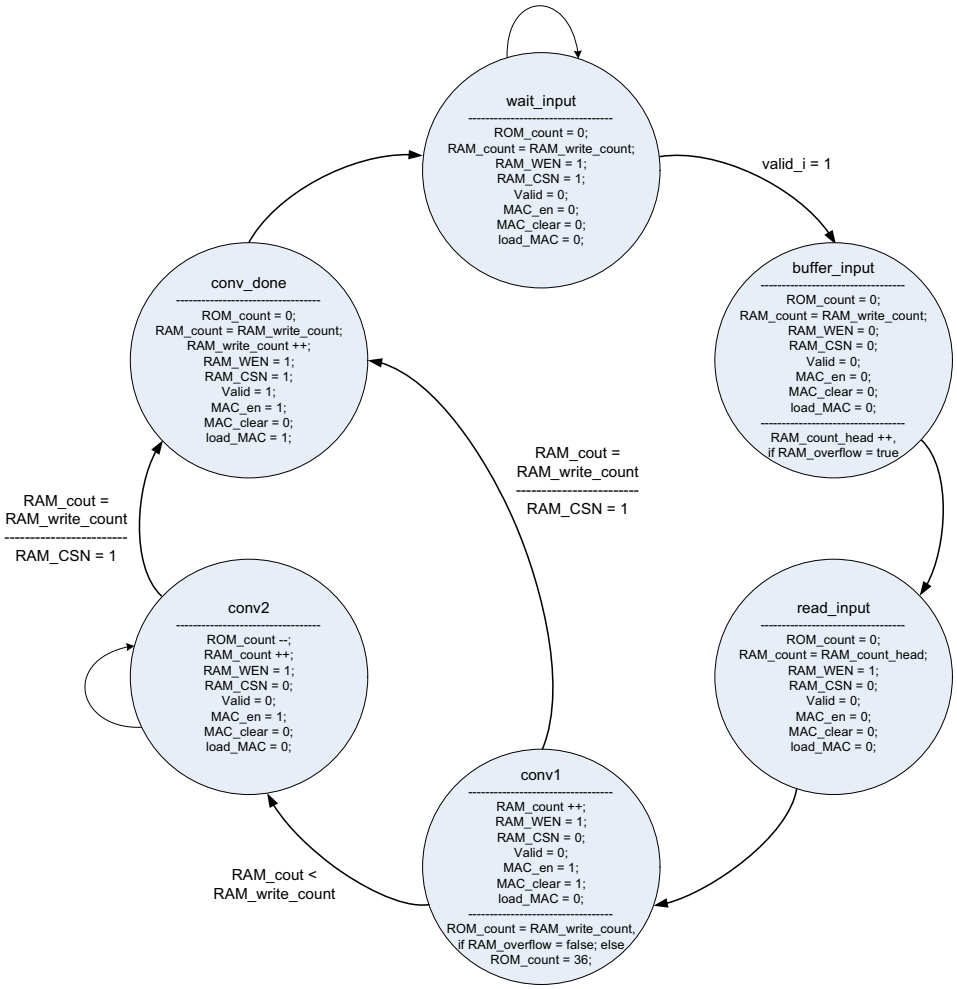


Figure 10. FSM of the central controller

FSM in the central controller contains 6 states, as shown in Figure 10.

"wait_input": This is an idle state. FSM listens to the starting signal – "valid_i". Upon the event of "valid_i" signal going high, FSM jumps to the input data buffering state.

"buffer_input": Incoming data is stored in the SRAM at the position pointed by "RAM_write_count".

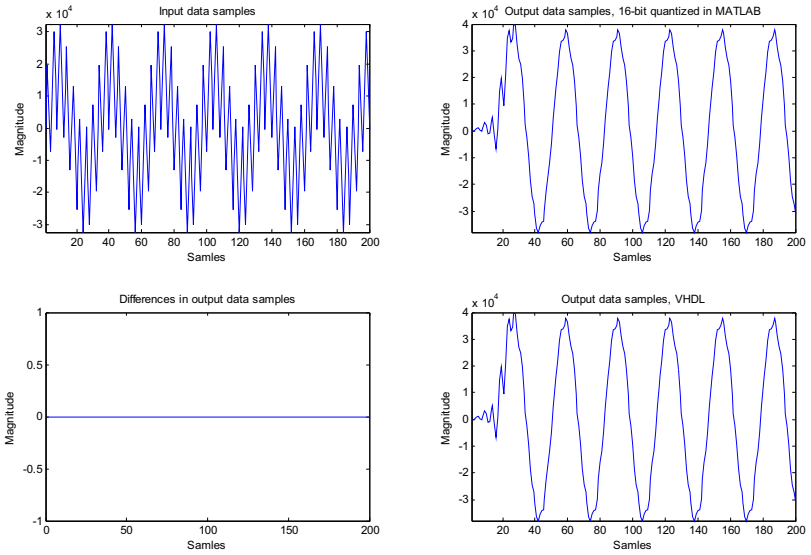
"Read_input": Since reading data out from SRAM has 1 clock cycle latency, so this state initializes the control signals and reading address for the SRAM.

"conv1": ALU starts processing the input data read out from SRAM. Internal register in the MAC unit is initialized in this state.

"conv2": ALU processing state. FSM moves on when the SRAM reading pointer reaches the tail pointer in the circular buffer.

"conv_done": ALU computation finished. Final result is clocked into the output register in MAC unit and being present on the output data bus. The flag of calculation done is toggled.

3.4. Output accuracy verification



(a) (b)
(c) (d)

Figure 11. Calculation accuracy comparison between MATLAB and system designed by VHDL. (a) Original input data samples. (b) Output data samples filtered by using full precision calculations in MATLAB. (c) Output data sample differences between MATLAB calculation and Time Multiplexed FIR calculation. (d) Output data samples filtered by using full precision calculations in Time Multiplexed FIR (VHDL).

Since there is no data truncation involved during the filtering process, so no value difference is observed from the output data plot when comparing with the full precision calculation done in MATLAB.

3.5. Conclusion

Time multiplexed FIR filter works as expected. Implementation on the circular buffer inside SRAM was the main concern in this assignment. By expanding the theoretical expression of data convolution, relations on input data samples and filter coefficients are evident, and the rest of experiments are straightforward.

By comparing the direct mapped FIR filter implemented in assignment 2 and the time multiplexed FIR filter, we can conclude as following:

Direct mapped FIR filter:

(1) Filter processes incoming data samples in parallel, where each output is calculated in one clock cycle. Therefore the throughput of system is high and independent with the filter order.

(2) High system throughput is paid out by the required hardware resources and therefore areas: m+1 fixed multiplier, m adders and m registers.

Time multiplexed FIR filter:

(1) Filter processes incoming data samples sequentially, where each output is done in multi-cycles. In this assignment, since a SRAM module is engaged for buffering input data, so some extra clock cycles are required: 1 clock cycle data writing into SRAM, 1 clock cycle reading latency from SRAM and 1 clock cycle for entering into the processing state in FSM, so overall $3+m+1=3+35+1=39$ clock cycles are needed to produce one data output. Due to this multi-cycle data processing, system throughput is mainly dependent on the filter order.

(2) Hardware resources are fixed and independent with the filter order: 1 flexible multiplier, 1 adder and 1 register.

The choice of proper filter architecture is dependent on the applications. For instance, the direct mapped filter can be used for high-end circumstances, where speed is crucial and cost is careless. Time multiplexed filter architecture is nice for lower-end applications, since it occupies less hardware area and flexible in functionality, but being paid by the required processing time.

4. Assignment 4a: Synthesis of assignment 3

4.1. What constraints were set in Design Compiler?

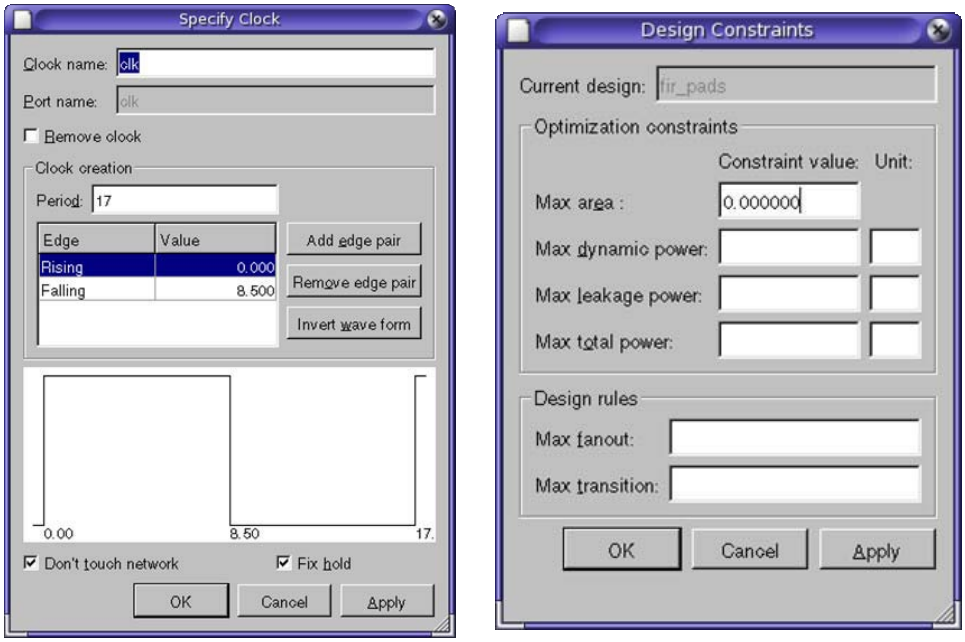
4.1.1. Maximum speed

Two phases are involved on synthesis in pursuit of maximum system speed. In phases 1, the expected clock period was set up to a non-reachable value, for instance, 1nS in my case. The synthesis tool will then try it's best to optimize design to reach the timing constraint. But since we know the dreaming clock frequency is nonrealistic,

so system clock delay violation was expected to see from the synthesis report. In phase 2, the slack clock delay from the synthesis result in previous iteration was added to the initial clock period, which results in a minimum achievable system clock delay for my design, therefore the maximum system speed. Figure 12(a) shows the visualization of the final timing constraint used in the design synthesis of phase 2.

4.1.2. Minimum area

In order to get a minimum design in terms of silicon area, design synthesis was slacked on the timing constraint, which to avoid the influences from system speed being as the crucial criterion. Instead, the maximum area expectation was set to 0 as the design synthesis constraint, as shown in Figure 12(b).



(a) Timing constraint *(b) Area constraint*

Figure 12. Synthesis constraints

4.2. Report the area for the sub-blocks.

Table 4. Report of system areas for two different synthesis aspects

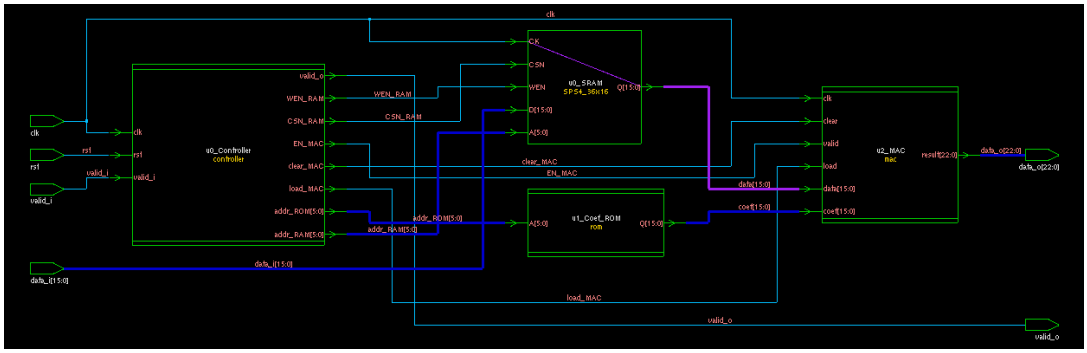
	Max. speed [μm²]	***Percentage***	***Min. area [μm²]***	***Percentage***
fir_pads (Wrapper)	1,513,956.972636	100%	1,458,395.4042	100%
fir	217,514.6622	14.4%	161,954.3862	11.1%
controller	16,319.8746	1.1%	16,088.2146	1.1%
mac	149,533.938	9.9%	94,205.4192	6.5%
rom	6,678.72	0.4%	6,678.72	0.5%
SPS4_36x16 (SRAM)				

Table 5. Report of MAC areas for two different synthesis aspects

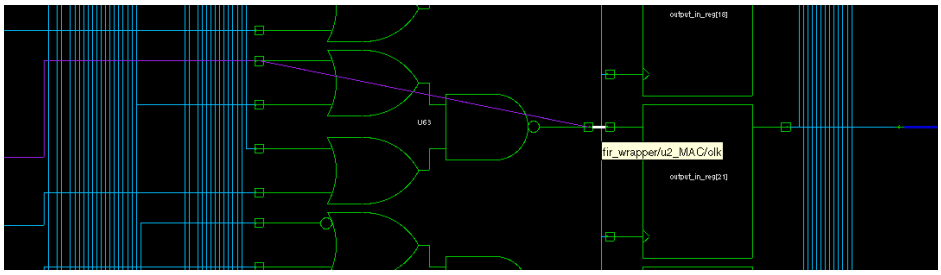
	Max. speed [μm²]	***Percentage***		***Min. area [μm²]***	***Percentage***
mac	149,533.938	100%	***mac***	94,205.4192	100%
DW01_add_1	23,357.673	15.62%	***DW01_add_0***	9,528.3	10.11%
DW_mult_tc_1	97,278.6348	65.05%	***DW_mult_tc_0***	61,458.9606	65.24%
registers	28,897.6302	19.33%	***registers***	23,218.1586	24.65%

4.3. What is the critical path for the design?

4.3.1. Maximum speed



(a) Overview of the critical path



(b) Critical path in MAC module

Figure 13. Critical path in maximum speed design

4.3.2. Minimum area

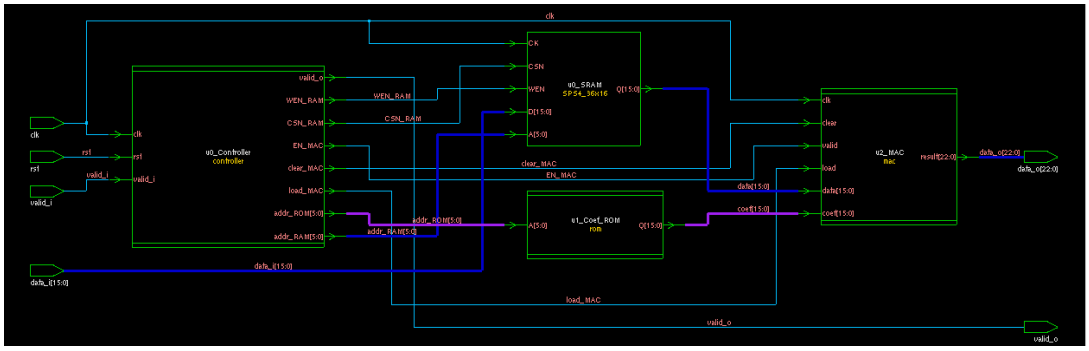


Figure 14. Critical path in minimum area design

4.4. Which blocks are in the critical path and what are the individual delays for each block?

4.4.1. Maximum speed

Refer to Figure 13, the critical path in maximum speed design is the input data bus going from SRAM to the MAC module.

Table 6. Signal delays in the critical path

	Path delay [nS]	***Percentage***
SRAM	5	29.62%
Multiplier	5.28	31.28%
Adder	5.91	35.01%
Output register	0.69	4.09%
Total	16.88	100%

4.4.2. Minimum area

Refer to Figure 14, the critical path in minimum area design is the filter coefficients data bus starting from the controller until the MAC module.

Table 7. Path delays in the critical path

	Path delay [nS]	***Percentage***
Controller	0.72	2.27%
ROM	4.96	15.64%
Multiplier	7.1	22.38%
Adder	17.92	56.49%
Output register	1.02	3.22%
Total	31.72	100%

4.5. Report maximal clock frequency and suggest how you could increase it at the register transfer level.

It is obvious to see from Table 6 and Table 7, “Adder” unit inside MAC module is the heaviest part in the critical path, which is unusual when comparing the strength of adder with multiplier that located beside. But this is the case in my design, because the reported delay of the “Adder” unit also includes signal delays on the input and output MUXs accompanied with the real addition unit, as shown in Figure 15. So in order to speed up system, pipeline registers might be considered to partition the adder away from the MUXs.

In addition, another crucial path in both designs is the data bus between MAC module and the data source, which is the SRAM module for speed version and ROM

module in the area design. One way to solve this problem is to insert the pipeline register in between of them.

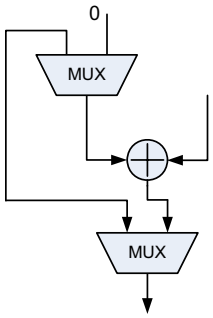


Figure 15. "Adder" unit in the MAC module

4.6. What are the differences between the high speed and low area synthesis results?

Table 4 verifies that different design constraints result in different synthesis output. Comparing two designs, the main difference comes from the different area contribution of the MAC module. Numerically, system optimized for speed uses 3.4% more silicon area than the area version in this case. This is because different adder and multiplier will be elaborately selected according to the different design constraints. Table 5 summarizes this in details.

So when considering a system design, there will always be a trade-off between silicon area and processing speed.

5. Assignment 4b: Place-and-route of assignment 3

5.1. Compare the final area with the estimated area from the synthesis tool. Only compare the core area.

Table 8. Core area comparisons between synthesis and place & route for maximum speed design.

	Max. speed [μm²]	***Percentage***	***Min. area [μm²]***	***Core utilization***
fir_pads (Wrapper)	1,513,956.972636	100%	3,735,437.522	58.97%

Apparently, the final core area after place and route is much larger than the one got from Synopsys synthesis tool. Here the main reason is that my design is pad-limited, therefore the actual core utilization is very little, as we can see both from the numerical value 58.97% and the design layout shown in Figure 16. So by using this information to calculate the actual core area, it becomes,

$$A_{Core} = 3,735,437.522 \cdot 58.97\% = 2,202,787.5067234\left[\mu\mathrm{m}^2\right]$$

Which is then close to the synthesis result.

5.2. Show layout.

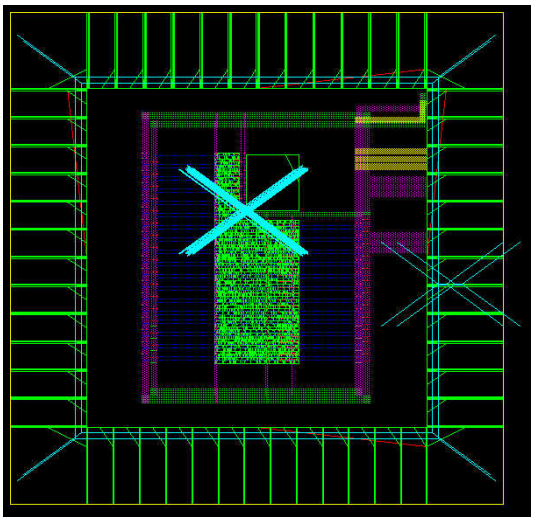


Figure 16. Design layout

5.3. Show script-files produced during P&R.

Table 9. List of script files used in Place & Route

Script file	*Description*
1_verilogin.mac	Import design libraries (.lef), design files and netlists.
2_floorplan.mac	Floor plan; Place IO pads; Place Blocks; Place Cells.
3_pads.mac	Place IO pads based on the design constraints.
4_powerroute.mac	Construct power rings around core and memory block.
5_fillperi.mac	Place filler IO cells.
6_connectrings.mac	Connect power rings.
7_import.mac	Import design files with the constructed clock tree.
8_fillercells.mac	Place filler cells.
9_route.mac	Route signals.

All of these 9 script files are enclosed in the file attachments.

5.4. Present and discuss violations.

There are two clusters of violations, one is around the IO pins of SRAM module, and the other one is around the system IO pins. But both of them can be ignored in this assignment. The detailed violation report is enclosed in the file attachments.

5.5. Conclusion

It was a nice tutorial in assignments 4 and 5. I have got the main idea on how an ASIC chip is performed from the RTL level down to the gate level. I have also experimented on how design constraints influenced the final chip layout in terms of silicon area.

Comparing with 3 previous assignments (VHDL designs), the design synthesis and place & route are more trouble like. To me, I think the tasks themselves are straightforward, but the doubts came from how to use the provided tools correctly. It is great of using better software that has better graphical user interface, but it would be nice if we can keep the lab menu updated simultaneously, because personally I got a lot of problems due to the performed wrong lab procedures or wrong operations.

Overall, I was really enjoyed in the trip provided from assignment 1 to 5.