IP - COR-ASP

GROUP4_4

SHAARAN ELANGO

OBJECTIVE

The main objective of this project is to design and test our reactive and concurrent processor utilising both Quartus Prime and modelsim. We have to ensure that any new instructions that are being passed through the processor are executed correctly within certain timing constraints. By using modelsim, we can check whether or not the correct instructions have been executed by checking our values at the register transfer level.

The main objective of this individual project is to create a new application specific processor (*ASP*) that will be used to calculate the correlation between points within a window of ADC samples.
This ASP will need to obtain values representing the averages from previous ADC samples, and calculate the correlation value within large windows using these average values. Once calculated, this information will be passed on to a peak detecting ASP (*PD-ASP*), where the values will be used to determine whether or not a peak is within the given correlation window.. By utilising modelsim, the validity of the correlation ASP can be easily checked and verified, ensuring that the ASP is functioning as expected and producing the correct results, such that we could then determine when peaks occur within an ADC signal.

BACKGROUND INFORMATION

Autocorrelation is a mathematical representation of the similarities and differences of a signal when compared to a lagged previous iteration of itself, as specified in [1] . This is done by taking periodic samples, called windows, within a signal, then using an algorithm to calculate the correlation value of that window. There are many different variants to this equation, however the most common variation is the variation that was best fitting for the project in hand.

$$Correlation = f(x) * f(x-1) + f(x+1) * f(x-2) + \dots f(k+n-1).f(k-n)$$

Equation One: Correlation value calculation

Where *n* is half the size of the correlation window, and *k* is the centre of the correlation window.

The designed ASP will be utilising *Equation One* to calculate the correlation value, since this best fits the implementation of our project. This is because the correlation window will be easily variable, along with it making it significantly more simple to implement in an FPGA.

METHODOLOGY

When designing the ASP required to complete the functionality of determining the correlation values of various windows, I had to consider how data would be transmitted between each component that will be a part of the final project.

This includes having to communicate with the averaging ASP (*AVG-ASP*) to obtain the average values that will be used in the correlation equation, as well as transferring the correlation values to the peak detecting ASP (*PD-ASP*).

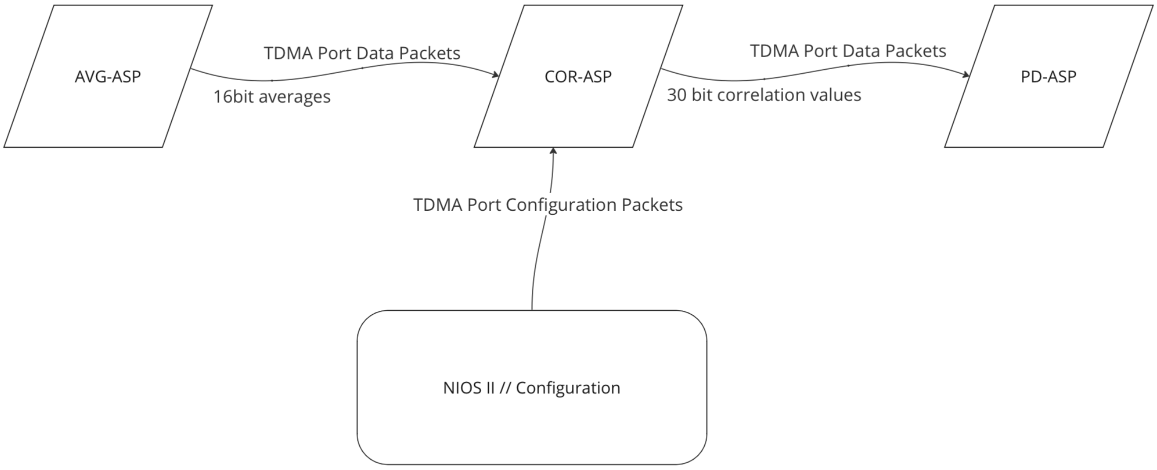


Figure One: Diagram of data flow to and from the COR-ASP

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15 downto 0
Data In	1	0	0	0													AVG Value
Conf-COR	1	1	0	1	Currently Unused				Next port, where to send data out to						Enable		
Data Out	1	V	CORRELATION VALUE – 30 BITS														

Figure Two: Table of TDMA port data bits and definitions

Figure One describes the dataflow that is involved in the COR-ASP. To configure the ASP, I assumed that the NIOS application that we will build in the latter stages of this project will send configuration packets to the COR-ASP. These configuration packets will contain information about enabling the ASP and where to route the outputs from this ASP. Next, the AVG-ASP component will be used to send data representing the latest average calculated through the TDMA-Min ports. Once this average is received inside of the COR-ASP, the average is stored within an internal buffer.

Storing within an internal buffer ensures that the information about the various averages taken are quickly available without any delay, allowing for the calculation of correlation to occur as swift as possible since the required values are already in reach of the ASP.

Currently, the correlation window is set to be equal to 20 averages, therefore, the internal buffer can only hold up to 20 averages before it is full.

Once the buffer is filled, *Equation One* is executed, where we set *n* to be equal to 10. The equation is completed within a VHDL process statement utilising variables, ensuring that the changes to values occur instantaneously. Once the calculation has been completed, a flag is set to clear the internal buffer to clear space for a new calculation to occur, as well as setting a bit to signal that the calculation has been completed.

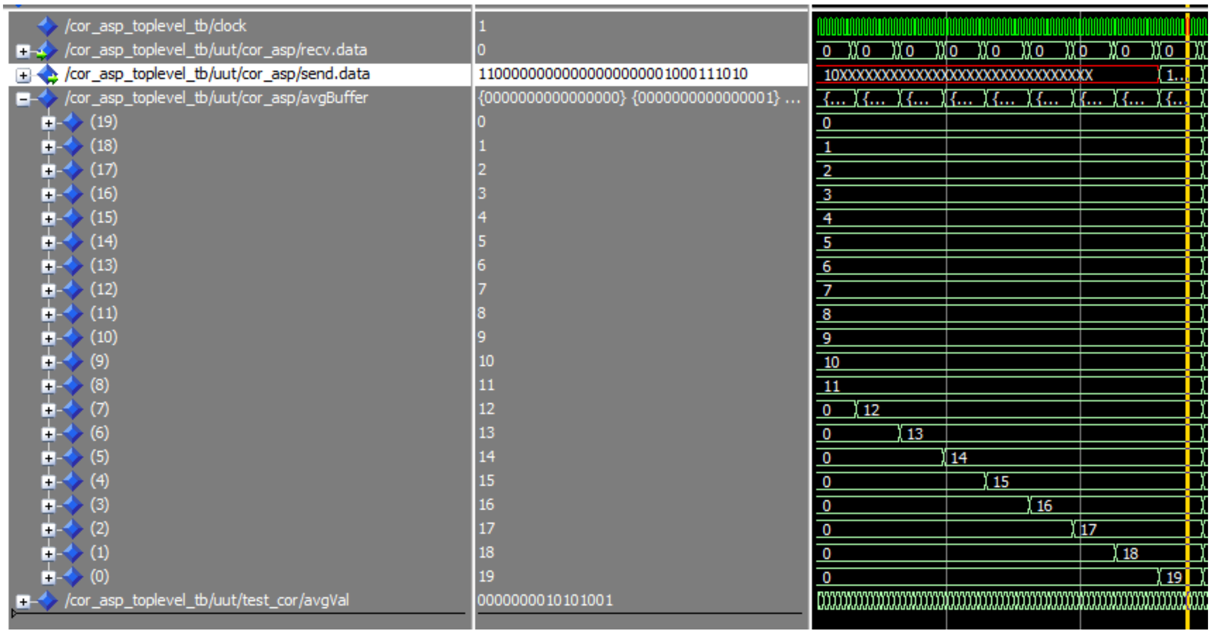
The TDMA-Min output port will then be sending the correlation value. As can be seen in Figure Two, the bottom 30 bits of the *Data Out* will represent the correlation value of the window, and the 30th bit will represent the validity of the correlation value. If V is equal to one, that means that the correlation value being outputted is a valid value, therefore should be used by the PD-ASP, if it is equal to 0 that indicates that the value is an old value, therefore should not be used in any calculations by the PD-ASP.

Due to the size of the average values obtained from the AVG-ASP, coupled with the correlation equation, the data size of the TDMA port was not large enough to contain the entire correlation value. This is because the value could exceed the 32 bit maximum that the TDMA transfers can handle.
There were multiple attempts made when trying to overcome this problem, including potentially sending the packets one after another, each with half of the correlation value. However this would have caused the COR-ASP to become very delayed, essentially doubling the latency of the ASP. Therefore I opted to use only 30 bits of the port for the data I need to transfer to PD-ASP, leaving two bits for PD-ASP to use to decipher what information is being received through its ports. This method causes two of the bits to become omitted, however this can be fixed by omitting only the bottom two bits of the correlation value, as these are the two least significant bits, therefore will be unlikely to cause large differences in values.

For testing purposes, instead of utilising the AVG-ASP and PD-ASP, a test ASP was utilised. This test file would simply send the configuration packet at the beginning, acting like the NIOS would, then continue to send new data packets as a replacement for the AVG-ASP component sending data. The COR-ASP also sends data right back to this component, to act like the receiving PD-ASP component. This was done for ease, as well as because I currently do not have access to the other two ASPs to test the correct functionality, therefore this will be further implemented in the full HMPSoC project.

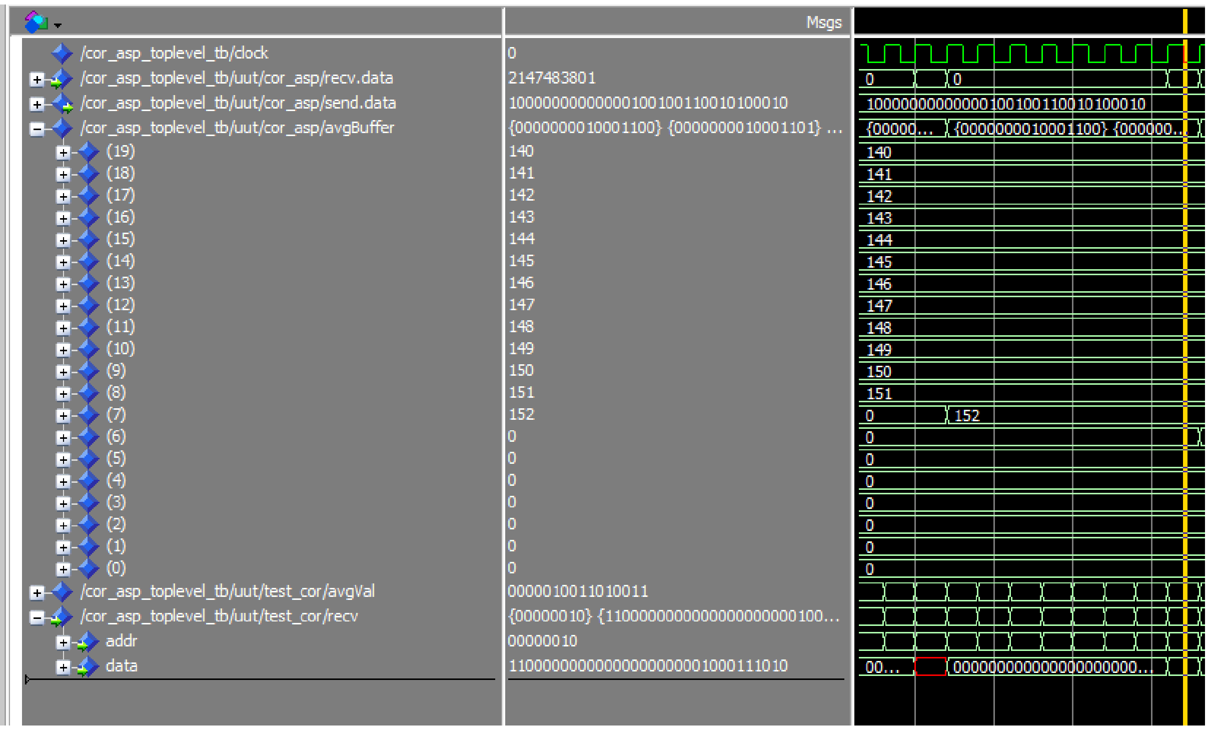
RESULTS

Through rigorous testing using the inbuilt waveform file in Quartus 18.1, testing was performed carefully by using the inbuilt Modelsim wave generator application.



/cor_asp_toplevel_tb/clock	1
/cor_asp_toplevel_tb/uut/cor_asp/recv.data	0
/cor_asp_toplevel_tb/uut/cor_asp/send.data	11000000000000000000001000111010
/cor_asp_toplevel_tb/uut/cor_asp/avgBuffer	{0000000000000000} {0000000000000001} ...
(19)	0
(18)	1
(17)	2
(16)	3
(15)	4
(14)	5
(13)	6
(12)	7
(11)	8
(10)	9
(9)	10
(8)	11
(7)	12
(6)	13
(5)	14
(4)	15
(3)	16
(2)	17
(1)	18
(0)	19
/cor_asp_toplevel_tb/uut/test_cor/avgVal	0000000010101001

Figure Three: Modelsim simulation results



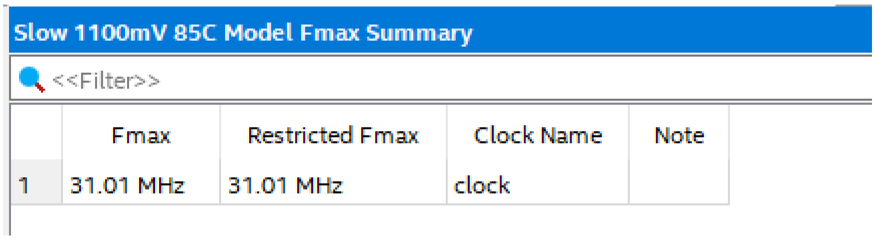
```
                                                   Msgs
/cor_asp_toplevel_tb/clock                         0
/cor_asp_toplevel_tb/uut/cor_asp/recv.data         2147483801
/cor_asp_toplevel_tb/uut/cor_asp/send.data         10000000000000100100110010100010
/cor_asp_toplevel_tb/uut/cor_asp/avgBuffer         {0000000010001100} {0000000010001101} ...
  (19)                                             140
  (18)                                             141
  (17)                                             142
  (16)                                             143
  (15)                                             144
  (14)                                             145
  (13)                                             146
  (12)                                             147
  (11)                                             148
  (10)                                             149
  (9)                                              150
  (8)                                              151
  (7)                                              152
  (6)                                              0
  (5)                                              0
  (4)                                              0
  (3)                                              0
  (2)                                              0
  (1)                                              0
  (0)                                              0
/cor_asp_toplevel_tb/uut/test_cor/avgVal           0000010011010011
/cor_asp_toplevel_tb/uut/test_cor/recv             {00000010} {1100000000000000000000000100...
  addr                                             00000010
  data                                             11000000000000000000000001000111010
```

Figure Four: Receive data from the test ASP

As can be seen in figure three, the modelsim results obtained show the buffer, which has been expanded to for clarity. The correlation calculation made is correct, as it follows the formula *Equation one*. When trying the formula by hand, we get the value 470, which corresponds to the bottom 30 bits of the *send.data* value. Referring to figure two, we know that the first two bits represent the validity of the data sent, whereas the bottom 30 bits represent the correlation value.

This shows that the calculation works as expected and is sending the data out through the TDMA-Min port associated with the ASP.
In Figure Four, it can be seen that the data finally is received by the testing ASP, however only after a very long delay. The first data was sent at approximately 300ns, whereas it was received almost 20,000 ns later. Parts of this could be due to the number of ports that are currently in the design, as I set it to 8 even though I only utilise two of them.

However, it can easily be seen through both Figure three and four that the correct calculations are made, as well as how the data being sent is also received on the other end, albeit at a delayed time.
This proves that the simulation works as expected, thereby ensuring that the COR-ASP functions as it should, meaning that the objective of converting the average values into a correlation representing a large correlation window has been achieved.

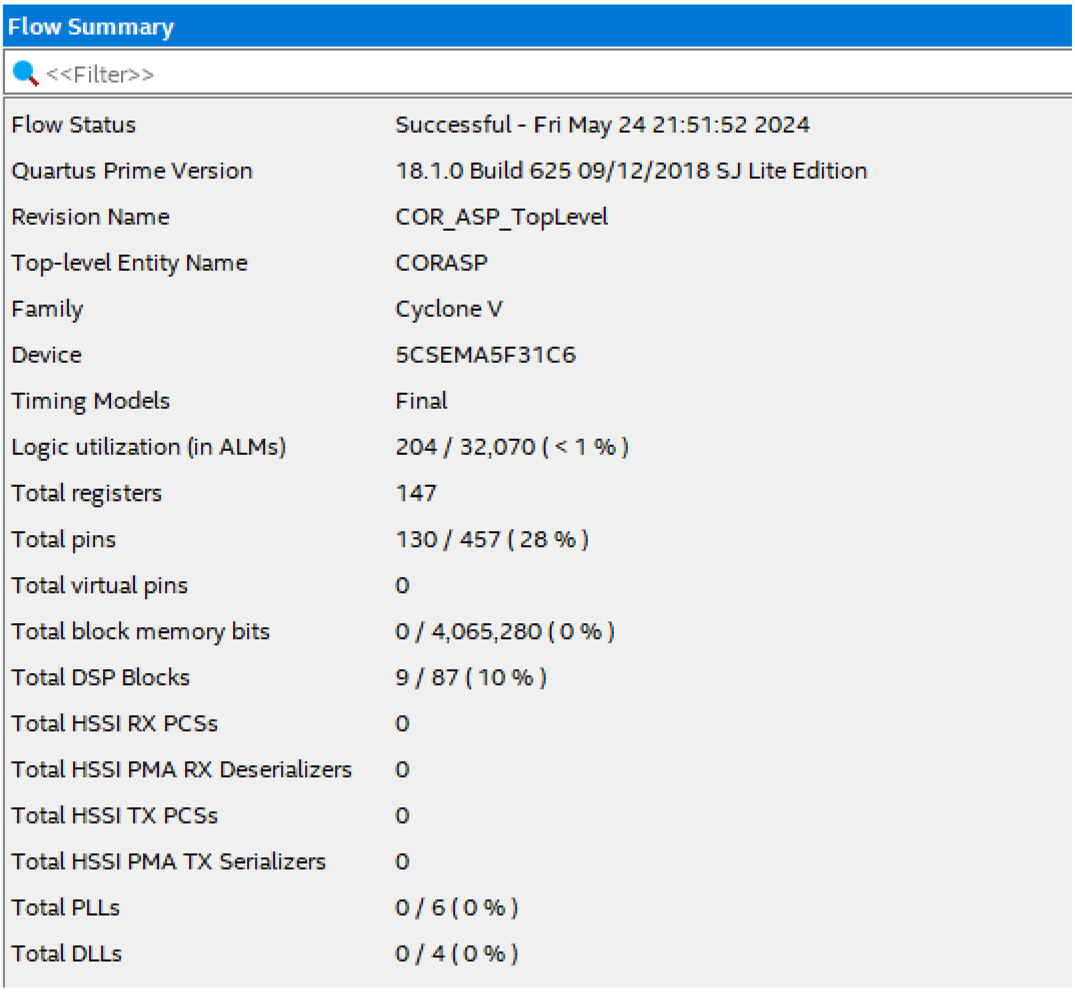


Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	31.01 MHz	31.01 MHz	clock	

Figure Five: Timing analysis and speed of the ASP



Flow Summary

<<Filter>>

Flow Status	Successful - Fri May 24 21:51:52 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	COR_ASP_TopLevel
Top-level Entity Name	CORASP
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	204 / 32,070 (< 1 %)
Total registers	147
Total pins	130 / 457 (28 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	9 / 87 (10 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure Six: Logic utilisation and resource usage of the ASP

Figure five and six represent the maximum clock frequency and resource usage of the ASP respectively. It should be noted that the maximum clock frequency is actually quite slow, this was fairly surprising.

DISCUSSION

Throughout the testing stages, it was determined that the COR-ASP meets its requirements and can accurately determine the correlation value for various windows using average values obtained from the AVG-ASP. However, the functionality comes with various limitations, the main one being the large delay when transferring data through the TDMA ports.

As stated in the *Results* section, the correlation value was only received at the test ASP's receive port after almost 20,000 ns. This is a very large delay that will likely need to be reduced in future.

I believe that this issue may be due to the number of TDMA ports that are linked up, however this will require further research to conclude the true reason for the large delay.

Another issue noted is how 2 bits of the final correlation value is being omitted when sending through TDMA. I believe that for the best functionality, the TDMA-Min should not be used, instead having buses route the data between the ASPs will likely result in a much quicker use case, especially for the sending of correlation data between the COR-ASP and the PD-ASP. As mentioned in the *Methodology* section, the large number of bits required for the correlation value would cause it to not be able to be transported in one packet using TDMA, requiring either some bits to be omitted or for a form of downscaling to be performed.

In hindsight, I believe that slightly downscaling such that I do not have to omit two bits would have been a greater approach. This is because instead of losing the bits entirely, the PD-ASP could then upscale back to the original value within the ASP, thereby meaning that we remain consistent. It may cause a slight inaccuracy, however it should be easier to accomplish and does not require drastic alterations to how the TDMA-Min data packets are distributed.

I also believe that it may have been beneficial to potentially use two different components to send and receive data, this is because it will represent the true implementation more closely, and it also may help us determine if the fact that data is being sent back and forth from two of the same ports could be causing the large delay that we experienced in the *Results* section.

CONCLUSION

The purpose of this project was to design and test a new application specific processor that will be utilised to determine the correlation of different segments of an incoming signal. This is so that a comprehensive final frequency calculating system can be created. This was accomplished using both Quartus Prime and ModelSim. During the testing stage, it was noted that the functionality of the ASP was perfect with no major faults. However it can be noted that there is a large delay when transferring data using the TDMA ports, further expanded on in the *Discussion* section of this report. Through careful design consideration of how to approach sending data through the TDMA, a respectable solution was accomplished. By being able to still send 30 bits instead of a capped 16 bits, the PD-ASP has a more reliable value to work with when making calculations and predictions on when a peak occurs.

By utilising the Modelsim RTL simulator through Quartus, the results confirmed how the COR-ASP can effectively execute correlation calculating when provided an input. Several challenges were faced along the way in line with concurrency issues which are common in parallel processing but through meticulous testing, we were able to overcome these issues and produce a successful outcome.

Moving forward, there are opportunities to enhance the COR-ASP by implementing a more robust and efficient technique for sending data to the PD-ASP to allow for a quicker data transfer without large delays. This is because the large delays may be detrimental to the final objective of obtaining accurate frequency readings.

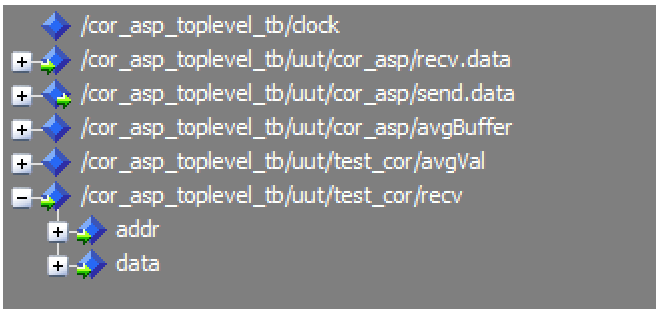
It is also required that the COR-ASP should be tested with the finalised AVG-ASP and PD-ASP to ensure that all members of the group have matching functionality and expectation from the other ASPs, such that there are no integration problems when combining the works.

In conclusion, the COR-ASP IP project was a successful endeavour that helped me demonstrate an ability to design and test an ASP for frequency calculating operations. The insights and experiences gained from this project provide a solid foundation for future work in developing efficient and reliable processors for embedded systems.

COMPENDIUM

Here are the instructions on how to run the COR-ASP and its test benches.

1. Open up the project using Quartus Prime
2. Once opened, select ‘COR_ASP_TopLevel’ to be the top level entity
3. Compile the top level.
4. Once compilation is complete, head to Tools > Run Simulation Tool > RTL Simulation to start up Modelsim
5. In modelsim, start compiling the various TDMA files, along with the CORASP.vhd, testCor, COR_ASP_TopLevel and COR_ASP_TopLevel_tb .vhd files.
6. After compiling, be sure to start the simulation, ensuring that you are simulating the COR_ASP_TopLevel_tb file
7. You can then start adding some waves, I would recommend the ones in the picture below



```
   /cor_asp_toplevel_tb/clock
+  /cor_asp_toplevel_tb/uut/cor_asp/recv.data
+  /cor_asp_toplevel_tb/uut/cor_asp/send.data
+  /cor_asp_toplevel_tb/uut/cor_asp/avgBuffer
+  /cor_asp_toplevel_tb/uut/test_cor/avgVal
-  /cor_asp_toplevel_tb/uut/test_cor/recv
   +  addr
   +  data
```

8. Start simulation, can choose any time you prefer.
9. If you would like to alter the simulation variables, feel free to do so. Notably testCor.vhd since it contains the values that the Cor-asp is receiving. You may also alter values through the COR_ASP_TopLevel_tb file as it also contains a nice Sine Wave that has been conveniently commented out.

REFERENCES

[1] M. Staudacher, V. Steixner, A. Griessner, and C. Zierhofer, “Fast fundamental frequency determination via adaptive autocorrelation,” *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2016, no. 1, Oct. 2016, doi: https://doi.org/10.1186/s13636-016-0095-8.