Hardware Software Co-Design (ECE 4530)
Codeisgn-Challenge

Vidur Kakar
vidurk@vt.edu

Correlation Engine

Given –

Digital correlation is a signal processing operation used to identify or locate signals. For example, correlation is used in the reception of GPS signals, in radar applications, as well for tracking of objects in real-time video data.

In other words, in correlation, we take a reference waveform in discrete-time data samples is used in locating a signal in the search waveform. The search waveform may also contain noise.

Thus correlation basically gives us the position of reference waveform in search waveform.

How is Correlation Done –

Correlation is done by repeatedly computing the dot product of the reference waveform with the search waveform.

Pseudo Code for Correlation –

For R samples of reference wave form and S samples of search waveform, the correlated value will be the following as shown below. Here 'corr' stores the sum of the multiplier of each search and reference signal.

```
for (j=0; j<S; j++) {
     for (i=0; i<R; i++) {
          corr[j] = corr[j] + reference[i] * search[(j+i) % S];
     }
}
```

Let us assume that the reference is like a window which slides over the search waveform, that is, acts as a sliding window to calculate the sum of correlation for each case.

But when the loop reaches the 113th iteration, beyond this iteration, the sliding window goes out of range, that is grater than 128 in the search waveform.

Thus to prevent this, a modulo operation is done to wrap around the signal. So after 128 sample's end, the window starts sliding back from 1 onwards.

We can imagine the search waveform as a circular structure which goes back to the start when one reaches at the end, that is, 128 samples.

Here the modulo of S (%S) is done for each signal to make sure the signal wraps around after reaching 128 values of the reference signal.

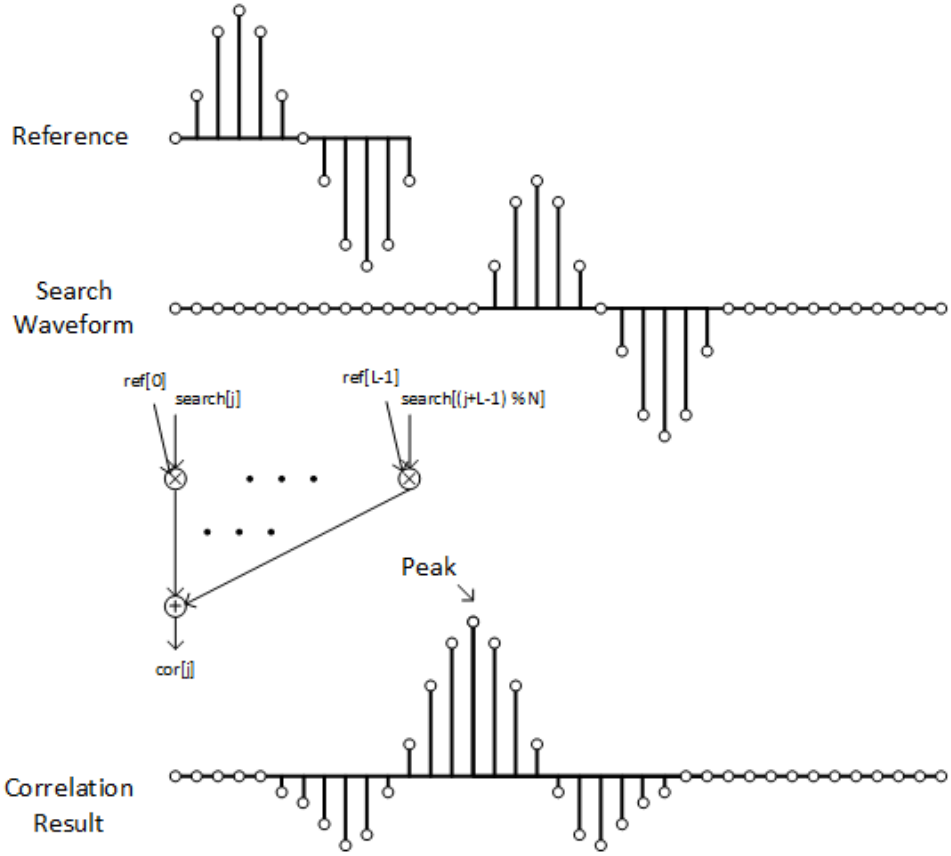


Figure 1: A visual example of Correlation

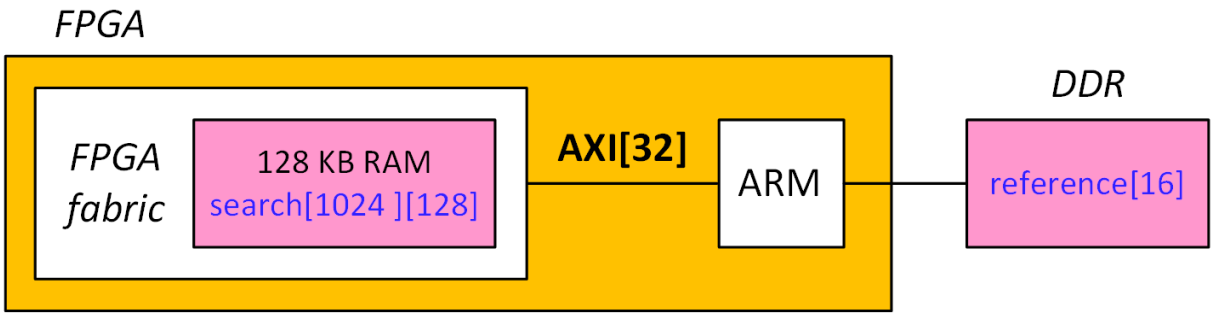
As shown in the image above, the peak indicates the location where there is maximum overlapping of the reference and the search waveform.

Numeric Data: -

No. of Samples in Search Waveform - 128
No. of Samples in Reference Waveform - 16
No. of Search Waveforms – 1024 (8 noise levels, thus 8 X 128 = 1024)

Resolution of Samples – 8 bit

Reference Architecture -



Design Strategy

From the given data, one figures out that multiplication and addition takes a lot of cycles. The iteration of loops takes the bulk of the clock cycles.

No. of iterations = 1024 (Outermost Loop) X 128 (No. of Search waveform samples) X 16 (No. of Reference waveform samples)

Total Number of iterations = 2,097,152

Also it is given that the execution time of the original reference design (460 million cycles).

Thus on an average no. of clock cycles for 1 loop iteration = 460,000,000 / 2,097,152
= 220 clock cycles

This means that a lot of clock cycles are being wasted for each computation. One has to reduce the number of iterations to reduce the clock cycle count.

Initial Strategies for speed up –

1.)

I copied the reference "bulkcorrelate" function inside the "hw_bulkcorrelate" function. In this function, I replaced the variables "waveforms" with 1024, "samples" with 128 and "templatesamples" with 16.

This saved a few clock cycles in each iteration of fetching the values of these variable. This helped achieve a slight speed up computation with 390 million clock cycles.

2.)

Optimum the code by putting the -O3 flag in the make file to make sure that the compiler compiles this program optimised for speed.

This helped me achieve another speed up of around 20 million cycles.

Next Strategy; Offloading computation to the Hardware –

To achieve a speed up, one has to offload computation as much as possible on the hardware to reduce the number of clock cycles for computation.

1.)

Testing initial setup by making a simple co-processor where I send the sig and template from the innermost calculation loop to the FPGA. The FPGA calculates the multiplication result of the two and returns the result of the same.

```
acc += sig[i * samples + (j + k) % samples] * template[k];
```

Now to achieve this, one had to define an intermediate memory space to link from the user memory space to kernel space to the I/O space. For this I define an mmap, from which one can use the virtual memory space to access the memory in the kernel space.

I then define a correlator base which is the sum of the virtual base and the defined memory space.

This design successfully runs but was not able to achieve any speed up as a lot of time goes to waste in sending and receiving of data to and fro from the FPGA.

Thus to achieve a good speed up, one has to do more operations together on the FPGA.

2.)

Now in my second strategy, I write a code in which I open the innermost loop of 16, through which the ARM sends complete reference waveform of 16 as well as the signal waveform of 16 to the FPGA.

Loop that is opened -

```
    for (k=0; k < templatesamples; k++) {
                        acc += sig[i * samples + (j + k) % samples] *
template[k];
      }
```

In short, my multiply and accumulate for each case is being done on the FPGA.

In this case, I achieve a marginal speed up to around 300 million clock cycles.

3.)

Now, to achieve a further speed up, I plan to do more number of multiply and accumulate operation in parallel.

Thus I design a system where the the ARM sends the reference signal and 16 search signal samples plus another 4 search signal sample combination to the FPGA.

Using this, I calculate four multiply and accumulate results and return the sum values of each back to the ARM.

On software I read the sum values of each and compare it to the max value to set the max index.

Thus using this strategy, I reduce the search signal loop from 128 cycles to 128 / 4, that is, 32 clock cycles.

One had to also take care of the wrap around in this strategy to make sure that the search signal wraps around after it reaches 128.

Verilog Design –

I have designed an FSM system on Verilog, where data is being written on the memory from the software and being accessed from the same on the FPGA.

Step 1 –

I send the reference signal first before the starting of the for loops as the reference signal stays constant throughout for that particular function all.

This is shown below –

```
corr_base[0] = *(int *)template;
     corr_base[1] = *(int *)(template + 4);
     corr_base[2] = *(int *)(template + 8);
     corr_base[3] = *(int *)(template + 12);
```

As one can see above, I write four samples (8 bit samples) of waveforms in each corr_base location of 32 bits.

Step 2 –

The 1024 outermost loop is left as it is.

Now inside the loop of 128, I similarly send a total of 16 search signal samples plus another 4 search signal sample combination to the FPGA so that reference signal can slide and calculate the value for 4 multiply and accumulate operations.

And I then call a Sync function in C which notifies the FPGA that the data has been fed.

Step 3 –

Once the FPGA receives the 'sync' signal from the software, it sends an ack signal back to notify it has read the data. Since it gets to know that the input data has been fed and it then goes to the next state and calculates the Multiply and Accumulate (MACs) result for 4 cases.

Step 4 –

The software moves to the next sync function after receiving and waits for a negative ack to get to know that computation has been completed on the FPGA.

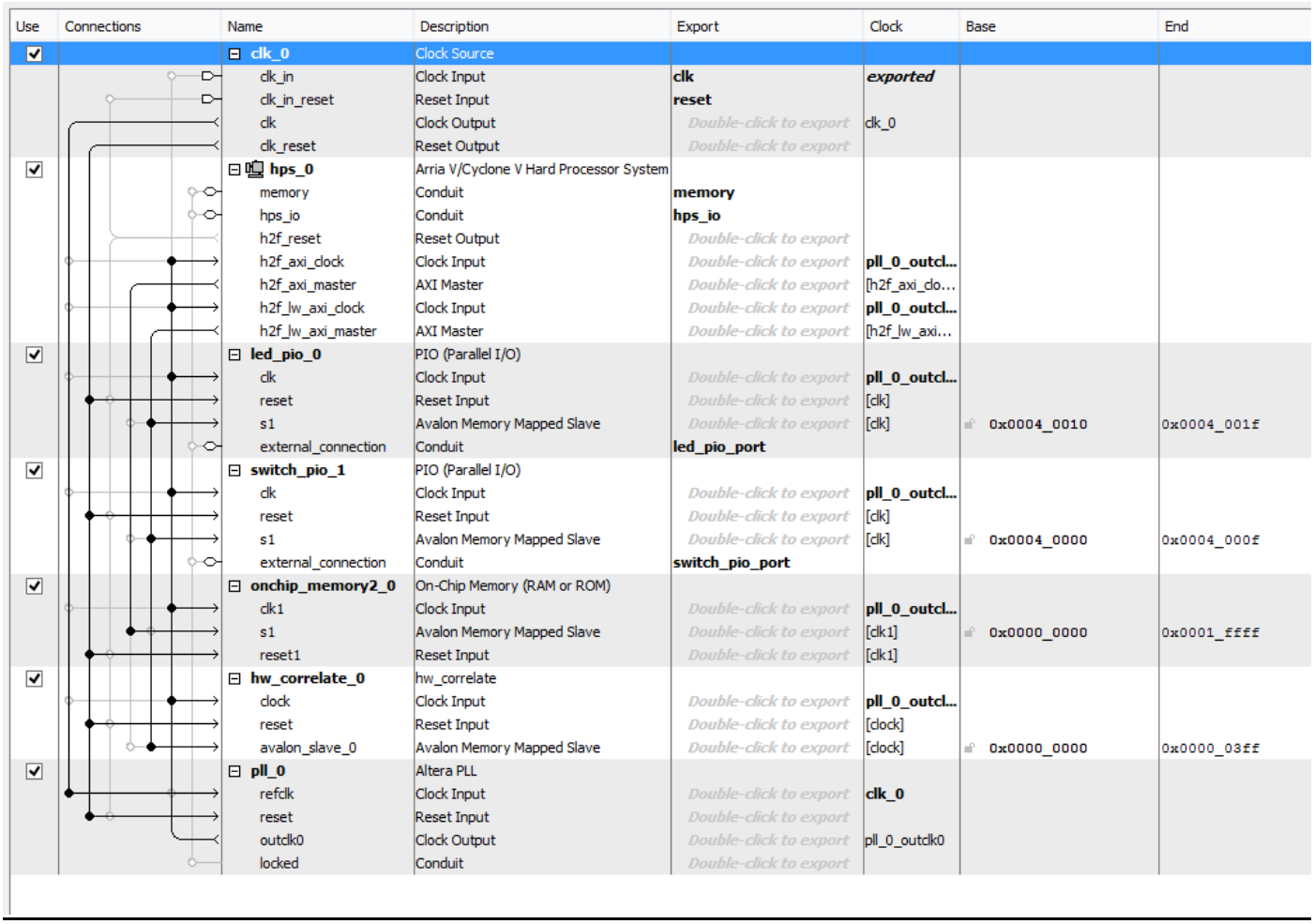
Step 5 –

Once the 4 MACs have been calculated, the FSM moves to the next state and sends a negative ack to the FPGA to notify it that the result has been computed.

Step 6 –

The negative ACK breaks the sync on software and then one reads the 4 MAC results. Each of the results are compared to current max value. If any of the obtained results of MAC are greater than the current max value, the greater MAC value replaces the max value and its index replaces the max index.

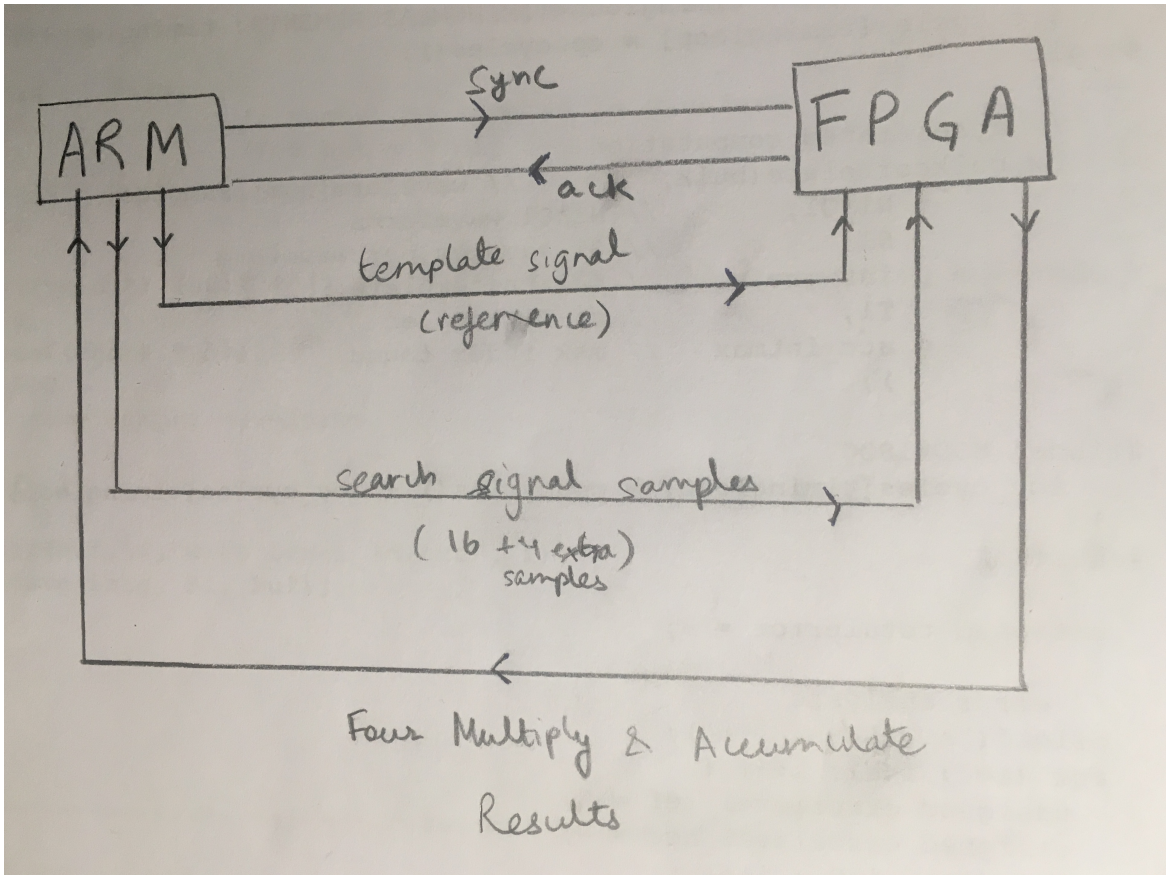
Architecture as seen on QSYS



Use	Connections	Name	Description	Export	Clock	Base	End
☑		**clk_0**	Clock Source				
		clk_in	Clock Input	**clk**	***exported***		
		clk_in_reset	Reset Input	**reset**			
		clk	Clock Output	*Double-click to export*	clk_0		
		clk_reset	Reset Output	*Double-click to export*			
☑		**hps_0**	Arria V/Cyclone V Hard Processor System				
		memory	Conduit	**memory**			
		hps_io	Conduit	**hps_io**			
		h2f_reset	Reset Output	*Double-click to export*			
		h2f_axi_clock	Clock Input	*Double-click to export*	**pll_0_outcl...**		
		h2f_axi_master	AXI Master	*Double-click to export*	[h2f_axi_clo...		
		h2f_lw_axi_clock	Clock Input	*Double-click to export*	**pll_0_outcl...**		
		h2f_lw_axi_master	AXI Master	*Double-click to export*	[h2f_lw_axi...		
☑		**led_pio_0**	PIO (Parallel I/O)				
		clk	Clock Input	*Double-click to export*	**pll_0_outcl...**		
		reset	Reset Input	*Double-click to export*	[clk]		
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0004_0010	0x0004_001f
		external_connection	Conduit	**led_pio_port**			
☑		**switch_pio_1**	PIO (Parallel I/O)				
		clk	Clock Input	*Double-click to export*	**pll_0_outcl...**		
		reset	Reset Input	*Double-click to export*	[clk]		
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0004_0000	0x0004_000f
		external_connection	Conduit	**switch_pio_port**			
☑		**onchip_memory2_0**	On-Chip Memory (RAM or ROM)				
		clk1	Clock Input	*Double-click to export*	**pll_0_outcl...**		
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	0x0000_0000	0x0001_ffff
		reset1	Reset Input	*Double-click to export*	[clk1]		
☑		**hw_correlate_0**	hw_correlate				
		clock	Clock Input	*Double-click to export*	**pll_0_outcl...**		
		reset	Reset Input	*Double-click to export*	[clock]		
		avalon_slave_0	Avalon Memory Mapped Slave	*Double-click to export*	[clock]	0x0000_0000	0x0000_03ff
☑		**pll_0**	Altera PLL				
		refclk	Clock Input	*Double-click to export*	**clk_0**		
		reset	Reset Input	*Double-click to export*			
		outclk0	Clock Output	*Double-click to export*	pll_0_outclk0		
		locked	Conduit	*Double-click to export*			

As seen above, here is my architecture of the design on QSYS. My coprocessor is named “hw_correlate” in the design.

My Design Approach –

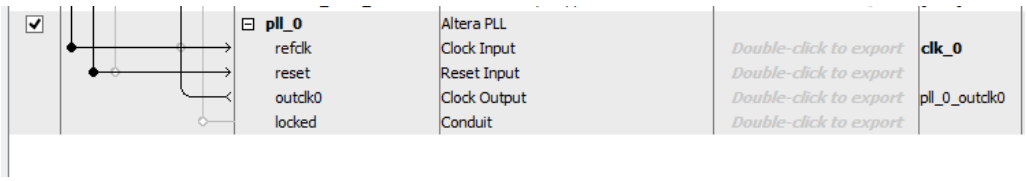


This design helps me achieve a speed up to around 168 million clock cycles.

Further, after this, to increase my speed, I devise to increase the clock speed of the Co-processor to bump up the computation speed.

Thus in my design, I add a PLL (Phase Locked Loop) which gets fed in 50 MHz from the clock source and it increases the clock signal to 150 MHz which gets fed into all the other components in the co-processor.

This is shown below.



The given change in clock speed further increases my computation power and decreases the clock cycles.

Final Clock Cycle Count –

```
root@socfpga:~/challenge# ./correlate
REFERENCE
Noise 0.125 error    0
Noise 0.250 error    0
Noise 0.375 error    3
Noise 0.500 error    5
Noise 0.625 error   19
Noise 0.750 error   18
Noise 0.875 error   34
Noise 1.000 error   77
ACCELERATED
Noise 0.125 error    0 delta    0
Noise 0.250 error    0 delta    0
Noise 0.375 error    0 delta    3
Noise 0.500 error    0 delta    5
Noise 0.625 error    0 delta   19
Noise 0.750 error    0 delta   18
Noise 0.875 error    0 delta   34
Noise 1.000 error    0 delta   77
Total Error: 156
Testbench passes!
Reference Execution time 193461778
Accelerated Execution time 108912012
root@socfpga:~/challenge#
```

As seen in the screenshot above, I achieve a final accelerated cycle count of 108,912,012.

Thus **Speed Up** = 460,000,000 / 108,912,012

= 4.2235

Other Strategies tried but not succeeded –

I tried adding more parallel multiply and accumulate (8 MACs together) to my design but was not successful as more than 5 MACs together do not synthesize on the FPGA board.

I tried another strategy with the following design in mind –

- Use DMA (Direct Memory Access) module to offload all the data at once on the memory slave initially from the ARM.
- Now FPGA will access the search and reference waveforms directly using that memory where the DMA had copied the data.
- The reference signal remains the same, on the other hand the search waveform is stored in the form of a shift register which keeps shifting after calculation of each MAC result to facilitate the wraparound and save computation time.

- This iterates for 128 times to on the FPGA and the max value and index are returned back to the software on ARM.
- The software reads the max value and its index.
- This would have reduced my program to only 1024 iterations to achieve a huge speed up.

Other Strategies One Can Possibly Try –

- Using DMA on existing structure.
- Increasing bus width to more than 32 bit to facilitate more data transfer together.
- Pipelining data on FPGA to reduce computation time.
- Changing bus structure to lightweight AXI bus to complete AVALON master bus to increase transfer bandwidth of data.