COMPSYS 701: Individual Research Project Report

Cecil Symes, csym531

Correlation & Convolution

What is Correlation and Convolution?

Correlation is defined as the relationship between two or more objects when there is a similarity of some sort between them. This holds true for signals as well, and correlation in Digital Signal Processing (DSP) is commonly used to find similarity between two inputs signals [1].

There are two types of correlation: auto-correlation, and cross-correlation. Auto-correlation describes correlating a signal with a time-shifted version of itself. This is typically done to find the time delay between sending and receiving a signal. Cross-correlation is correlation between two different signals and is typically done to try find a shorter known signal within a longer unknown signal [2].

In DSP, signals are often discrete. Discrete Correlation can be described with the following formula:

$$\boxed{R_{xy}[m] = \sum_{n=-\infty}^{\infty} x[n]\, y^{*}[n-m]}$$

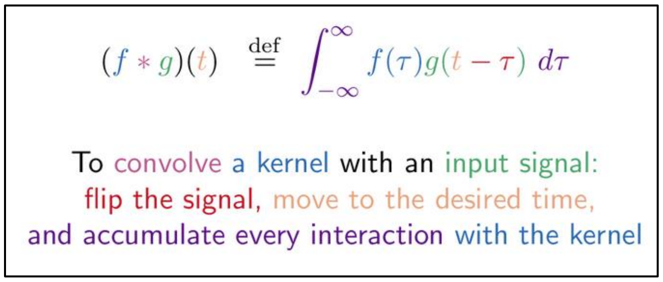
Image obtained from [2]

R[m] represents the output signal, with x[n] and y*[n] representing the two input signals. Note that y*[n] indicates it is simply y[n] but flipped, i.e. the first value is now the last, second value is now second to last, etc.

Correlation is useful in many practical scenarios. A signal can be auto-correlated with itself to help make finding periodicity easier. Cross-correlation is also used in scenarios such as radar tracking to check if a target is present [2].

For the rest of this report, correlation will be used to refer to cross-correlation.

Convolution is mathematically very similar to correlation. In real world terms, convoluting two signals together gives the combined response of both signals in terms of their frequency response.



$$(f * g)(t) \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\, d\tau$$

To convolve a kernel with an input signal:
flip the signal, move to the desired time,
and accumulate every interaction with the kernel

Image obtained from [3]

In the frequency domain, multiplication of two signals gives the time domain equivalent of convolution.

Uses of convolution include in convolutional neural networks, a type of machine learning algorithm, as well as in audio processing, such as reverb, where sounds can be given the illusion of having been recorded in an empty room.

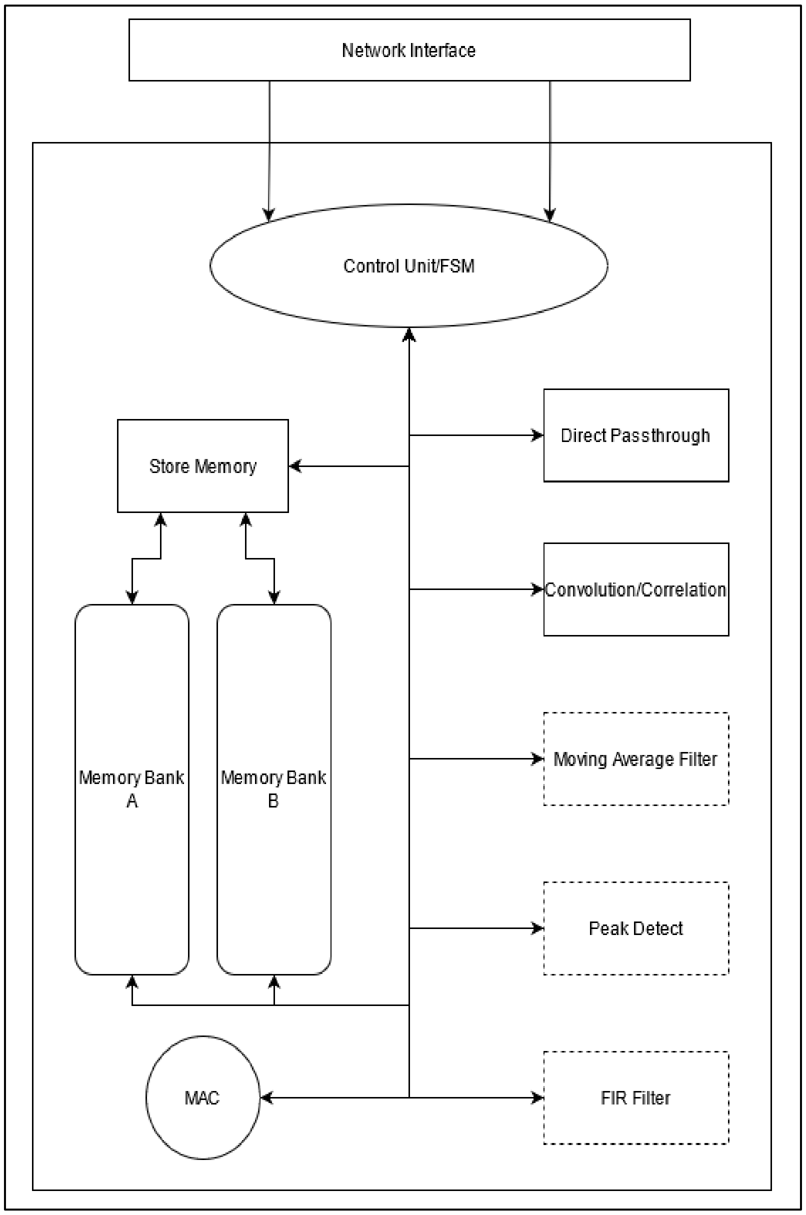
Existing Research

Research papers have investigated various methods of increasing the performance of correlation and convolution operations. [4] looks at optimizing the Multiplication and Accumulation (MAC) unit. [5] discusses a serial approach, a parallel approach, and an approach using Fast Fourier Transforms, or FFT. As previously mentioned, convolution in the time domain is equivalent to multiplication in the frequency domain. To convert a signal from the time domain to the frequency domain, the Fourier transform must be used. The Inverse Fast Fourier Transform, or IFFT, can be used in the reverse direction. Thus, for large amounts of data, it can be beneficial to FFT all data, multiply in the frequency domain, and IFFT back to the time domain to obtain a result [6].

For this report, a serial approach was taken due to its simplicity and ease of implementation. Limitations and Future Work for this serial approach are discussed further on in the report.

Data Processing-Application Specific Processor Design

Current structure



A diagram of the current DP-ASP structure.

Overall Structure

The Network Interface provides connection to the Network-on-a-Chip, or NoC. The Control Unit contains a Finite State Machine, or FSM, and checks each packet to see if it is a configuration packet. If it is, then the state changes on the next clock tick. There are planned to be 6 functionalities once the Data Processing-Application Specific Processor, or DP-ASP, is complete: Direct Passthrough, Storing to Memory, a Moving Average Filter, a Finite Impulse Response (FIR) filter, a Peak Detector, and a Correlation and Convolution mode. Currently, as of this report, the Direct Passthrough and Store to Memory functions work as intended. Convolution and Correlation provides correct results when tested separately, but it is not yet functional within the DP-ASP. The other three functionalities are not yet implemented and are marked with dotted lines to represent their planned future inclusion.

Control Unit & Finite State Machine

The *config_packet_reader* process surveys all incoming packets to check for a DP-ASP configuration packet. Once a DP-ASP configuration packet is received, the process sets the next state to transition to, as well as any accompanying parameters read from the configuration packet.

The *fsm* process handles all state changes, and the accompanying logic that comes within each state. Currently, the *fsm* process does not handle state transitions correctly, and jumps between the previous and intended state before settling. Fixing the state machine is a priority and is briefly discussed further under Future Work.

Memory Blocks and Storing to Memory

The memory can only be stored when in Store Memory mode. The memory banks are 512 words deep, with a width of 16 bits, and there are 2 banks. These are instantiated using a *for...generate* loop, meaning that the number of memory banks can be changed. However, for the scope of the current project, only 2 banks will be used, named memory banks A and B.

The memory cannot be read and output to the NoC directly, meaning results must be output directly from a function block. Writing to memory is therefore only useful for when preparing to use a function block, i.e. loading of values to be correlated, or after a function block is finished and the user plans on chaining different functionalities within a single DP-ASP.

When writing to memory, the user must specify which memory bank to write to, how many samples to store, and the starting address to store at. Currently, if the starting address and number of samples to write exceeds 512 then the data will "wrap around" back to the zeroth address. There is not currently a check in place for this, and the responsibility lies on the user not to "overflow" and store data in an unintended place. A simple solution is briefly discussed in the Future Work section.

Due to the limited size of the DP configuration packet, 9 bits are used to select the number of samples to save. However, as the maximum value representable by 9 bits is 511, the number of samples saved is actually the parameter field + 1. The user must be careful when entering a value. E.g. if the user wishes to store 6 values, then 5, or "0101", must be placed in the DP configuration packet.

When reading from the memory, it takes one cycle for the value to appear at the output. This must be accounted for by the user and is accounted for in other parts of the design, such as the Convolution & Correlation unit, as described further on in the report.

Direct Passthrough

As the name suggests, Direct Passthrough mode does not do any data manipulation and simply directs all data received from the NoC back onto the NoC.

Convolution & Correlation

The Convolution & Correlation mode is the chosen DSP algorithm for the IRP, as previously mentioned. Currently, the Convolution & Correlation logic works fully, giving correct outputs for inputs. This is demonstrated in the provided code and screenshots. The functional Convolution & Correlation logic is currently stored inside "*conv_unit.vhd*" and can be tested with "*conv_unit_testbench.vhd*." The report will focus on the Convolution & Correlation unit, as the Convolution & Correlation logic is not fully functional within the DP-ASP itself as of yet.

The Convolution & Correlation unit uses an FSM to operate, due to the delays present in certain operations. For example, accessing memory takes one clock cycle from choosing an address to actually receiving the data. As a result, the Convolution & Correlation unit has states for fetching, calculating, and storing data separately.

Multiplication & Accumulation Unit

The Multiplication & Accumulation (MAC) unit is a straightforward implementation. Two inputs are taken, multiplied, and added on the rising edge of the input clock if enable is high. The result is produced on the same clock cycle it is enabled, and the result is stored internally until the unit is told to reset.

Design Choices, Trade Offs, & Justifications

Maximum Number of Samples

Within the design, the size of both memory banks was limited to 512 words. For a given discrete Convolution/Correlation operation, the length of the output signal y[n] will be equivalent to the width of both input signals added together, subtract 1. This means that with a maximum memory size of 512 words, the combined length of both input signals could not exceed 513. As a result, input samples are limited up to a maximum of 256, meaning that the total samples of both combined come to 512, meaning there will be no loss of data.

Power of 2 for Number of Samples

Carrying on from the limited number of samples, the choice was made to have the number of samples be a power of 2. This was done due to the limited packet size of the system. Packets are 32 bits in size, and 16 bits are reserved for describing the type of packet, destination of the packet, origin of the packet, and other packet specific properties. For DP-ASP configuration packets, there are 16 unused bits that are free to be used for parameters. If the number of samples were to be represented by an unsigned integer, then 8 bits would be needed to represent all 256 possible values. This would only leave 8 bits for the starting address selection, operation selection, and destination for the data output. As such, powers of 2 were chosen for sample size, meaning that to get up to 256 samples, only 4 bits were needed, as 2^8 = 256, and 8 needs 4 bits to represent.

Number of Samples Same for Both Memory Banks

Due to the limited DP configuration packet size, the design choice was made to have the number of samples selected by the user correspond to both memory arrays. Ideally, if the DP configuration packet were large enough, each memory bank could store a differing number of samples for computation. As the DP configuration packet is limited in length, this is not possible. The user may choose to pad out their data with zeros if one signal is shorter than the other.

Flexible Starting Address Point

As just mentioned, by using powers of 2 for the number of samples to process, the number of bits required decreased from 8 to 4. This allows 9 bits to be used for selecting the starting address when accessing the memory for data to process. The use of 9 bits means memory can be accessed, from 0 to 511 inclusive, and the number of samples to process is flexible from 0-256 in powers of 2.

A key point to note is that it is possible to attempt to index outside the size of the memory, by selecting a starting address close to the end of the memory, and then selecting a number of samples that exceeds 511 when summed with the starting address. A simple failsafe is briefly discussed in the Future Work section, however the bulk of responsibility lies on the user to ensure they are not indexing out of bounds and losing data.

Data Output is either A or NoC

It is planned that when configuring the Convolution & Correlation unit, an output destination has to be specified. This is either memory bank A, or the NoC. This allows the user to either save data within the DP-ASP itself for further computation, or to output data to an external node, such as another DP-ASP or the Nios II.

Memory bank B is not an available destination to write to, and this is for two reasons. First, it is planned that the FIR Filter will use coefficients that are stored in memory bank B. This means that B will have less than 512 spaces available for data storage, and as such it was chosen to simply not have the option to store to B. Excluding memory bank B also allowed the parameter within the DP configuration packet to be kept to a single bit. Including B would mean having to extend the parameter to 2 bits wide, which reduces the already limited 16 available bits even further.

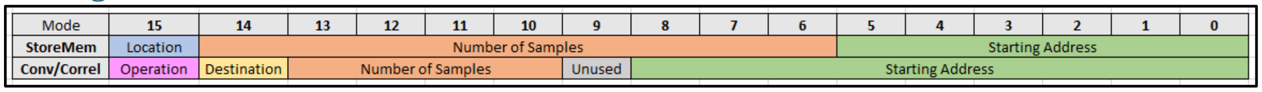
No Autocorrelation

Currently, there is no option for auto-correlation. The user manually has to load the same data into both memory banks, and then choose to do cross-correlation. The reasoning for this is that the memory is only single port, meaning one value can be read at a time. While it would be possible to read the two required values in sequence, the timing would be different to cross-correlation, and the current implementation would not work. A workaround to enable a form of auto-correlation is discussed under Future Work.

Enabling & Disabling the MAC Unit Clock

The current design has a single MAC unit that is active on the rising edge of its clock when the enable signal is high. This is done to ensure that there is not needless clocking of the MAC unit when no operations are required, saving overall power consumption. However, the current implementation is functionally the same as simply having the enable signal be the clock signal. As such, a potential asynchronous design is briefly discussed under Future Work.

Config Packet Breakdown



Mode	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
StoreMem	Location	Number of Samples									Starting Address					
Conv/Correl	Operation	Destination	Number of Samples				Unused	Starting Address								

For StoreMem:

Bit 15 - Location

Storage location for the samples. ‘0’ is memory bank A, ‘1’ is memory bank B.

Bit 14 downto 6 (9 bits) - Number of Samples

Unsigned integer representing the number of samples to store. Add 1 to the unsigned integer to get the value.

Bit 5 downto 0 (6 bits) - Starting Address

Address to start storing values. The bits are shifted left by 3 in order to cover all sections of memory. This essentially means the start address increases in blocks of 8, i.e. “0001” << 3 = “1000”, or 1 becomes 8.

For Convolution & Correlation:

Bit 15 - Operation

Selects the operation. ‘0’ is Convolution, ‘1’ is Correlation.

Bit 14 - Destination

Selects where the output of the operation goes. ‘0’ saves to memory bank A, ‘1’ outputs to NoC.

Bit 13 downto 10 - Number of Samples

Number of samples to compute with. The value is treated as an unsigned integer which is used to raise 2 to the power of. For example:

- If “0000” then 2^0 = 1 sample used
- If “1000” then 2^8 = 256 samples used

8 is the maximum useful value, as 2^8 = 256 and more than 256 samples would lead to data loss with a 512 large memory bank.

Bit 8 downto 0 - Starting Address

The address from which to start taking values. Range is from 0 to 511. There is no check for wraparound, so if the starting address added with the number of samples exceeds 511 then the operation will stop early.

Future Work & Suggestions

Correction of the FSM

Currently, the FSM is unable to exit states properly. When an operation is finished, the FSM struggles to leave the current state and enter the Reset state. This is a priority to fix and must function as intended for the rest of the design to work.

Asynchronous MAC Unit

As previously mentioned, the MAC unit is synchronous and performs computations on the rising edge of an input clock signal. However, the MAC unit also takes in an enable signal and does not compute when the enable is low. In the current implementation, this essentially means that the enable signal acts as a clock, as there are no circumstances under which the MAC unit runs on consecutive clock signals. As a result, an asynchronous MAC unit that simply computes on the rising edge of the enable would reduce the overall complexity of the design by a small amount.

Ensuring StoreMemory Does Not Exceed Bounds

As previously mentioned, StoreMemory does not currently check to see if it will index out of the bounds of the memory banks. If StoreMemory exceeds address 511, it will wrap around and write to the zeroth address. A simple fix can be to check if the current address is 511 and if there are still remaining samples to write. StoreMemory can be terminated early, meaning no data is overwritten at the zeroth address. However, most of the responsbility still lies on the user not to exceed the bounds of the array, losing data in the process.

Swapping between Convolution & Correlation Operations

To swap between Convolution & Correlation, the user will ideally change a bit within the configuration packet to select the desired operation. Currently, *line 125* of "*conv_unit.vhd"* and the following lines must be commented or uncommented to select the desired operation. This is a simple change to be implemented once the design is fully migrated into the larger DP-ASP system, and can be done with a conditional check of the corresponding parameter bit of the DP configuration packet.

Automatic Auto-Correlation

As previously mentioned, the system does not inherently have an auto-correlation mode. The user must manually load identical data into both memory banks first, and then perform cross-correlation. A potential solution to ease this process would be to add a pseudo auto-correlation mode, where once the user chooses auto-correlation, the corresponding data from A is copied over to B, and then the correlation operation is performed. This would preserve the overall timing of the correlation operation. However, a downside is that if there is data in B, it may get overwritten without the user realizing.

More Efficient Indexing

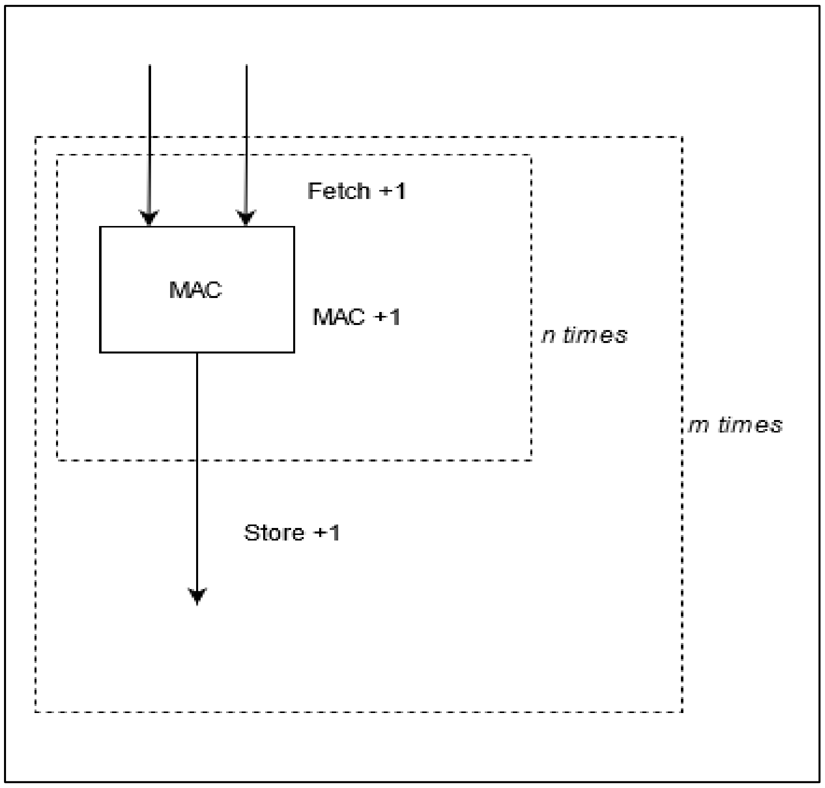
In the current implementation, the indexes for both input signals are calculated first and then checked to see if they fall within valid bounds (not negative, and not more than the number of samples specified). Each computation takes a clock cycle, meaning that even if no multiplication and accumulation is done, there is still a slight delay. A potential optimization would be that for every new output signal value, the first valid indexes as well as the last valid indexes are calculated, and computations are only done within the range of these indexes.

Parallelization

The last major point for future work is parallelization of the implementation. The current design only makes use of a singular MAC unit, meaning each a single output value must wait for the following:

- Fetching data values (1 clock cycle)
- The multiplication & addition (1 clock cycle)

The following diagram indicates the number of times these operations are repeated. The +1 indicates +1 clock cycle, and the dotted outlines and accompanying italic text indicate all actions within the dotted outline are repeating that many times.



A diagram indicating the approximate number of clock cycles for the current serial implementation.

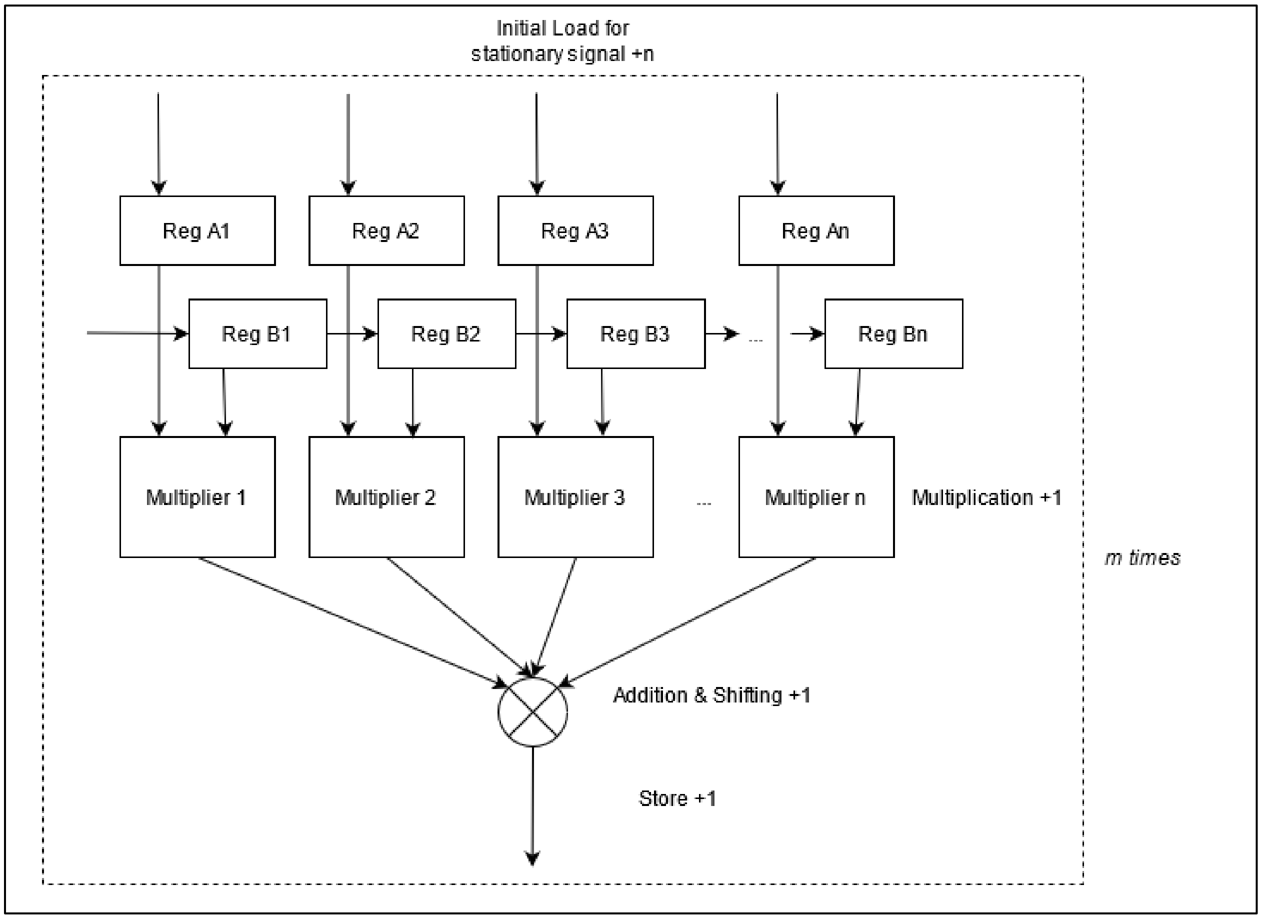
This comes out to *m * (2 * n + 1)* clock cycles.

A potential parallel design would be as follows:

- Instantiate *m* number of multipliers.
- Connect all the multipliers to an accumulator.
- Connect two registers to each multiplier, one for each data input.
- Fill one of the two registers on each multiplier with sample data for the “stationary” input signal. (+*n* clock cycles)
- Place the first sample data point from the “sliding” signal into the leftmost multiplier’s remaining register. (+1 clock cycle)

- Multiply all in parallel. (+1 clock cycle)
- Add all results together and simultaneously shift the values in the second register of each multiplier along. (+1 clock cycle)
- Repeat *m* times.

This ideal solution would require *3 * m + n + 1* clock cycles.



A diagram indicating the approximate number of clock cycles for an ideal parallel implementation.

For an *n* value of 256, *m* would come out to 511. For the serial implementation, this gives approximately 262,000 clock cycles. For the ideal parallel implementation, this gives 1,800 clock cycles. This is roughly a speedup of around 145 times.

All calculations are approximate and are not meant to be exact. The formulae for clock cycles may not be accurate. However, the point of vastly increased speed still stands.

The downsides are that achieving true parallelism comes at a very large hardware resource cost. 2 * *n* registers and multipliers would need to be instantiated, which is much more than the serial implementation. The current MAC unit also is not adequate for the parallel design and would have to be modified to support it.

References

[1] Wikipedia, "Cross-correlation," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Cross-correlation.

[2] S. H.L., "Understanding Correlation," 4 January 2017. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/understanding-correlation/.

[3] "Intuitive Convolution," Better Explained, [Online]. Available: https://betterexplained.com/articles/intuitive-convolution/#Part_5_Applications.

[4] Y. K. D. S. M. V. S. Ahish, "Design of High Performance Multiply-AccumulateComputation Unit," *IEEE International Advance Computing Conference (IACC)*, pp. 915-918, 2015.

[5] D. Gruyter, "A highly scalable FPGA implementation for cross-correlation with up-sampling support," *Impedance Spectroscopy*, pp. 81-91, 2019.

[6] S. W. Smith, "FFT Convolution," in *The Scientist and Engineer's Guide to Digital Signal Processing*, 1999, pp. 311-318.

[7] Wikipedia, "Convolution," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Convolution.