POLITECNICO DI TORINO

Master's degree in Electronic Engineering

Master's Degree Thesis

Hardware Accelerators for Long short-term memory Neural Networks using High Level Synthesis (HLS)

Supervisor
Prof. Luciano Lavagno

Candidate
Muhammad Usman Jamal

Dec 2018

Abstract

Long short-term memory networks, referred as LSTMs, are a notable kind of recurrent neural networks. They allow you to overcome vanishing gradient problem. Some of the different applications of LSTM include speech recognition, handwriting generation and recognition, music generation and composition, etc. FPGA-based hardware accelerators have been used recently due to their good performance in terms of power and flexibility. In this thesis, hardware accelerators have been implemented, synthesized and optimized for LSTM while performing extensive *fixed-point* data type optimization's using the Vivado HLS tool. The data types used are *fixed point - 16*, *float* and *double*. One of the bottlenecks faced during the synthesis is sigmoid activation function which is non-linear. A piecewise linear approximation is used for sigmoid function to overcome this issue. Different optimization's and directives are applied to explore different micro-architectural solutions. Pragmas like loop pipe-lining and unrolling, array partitioning etc. are applied during the synthesis process to find the optimum solution. Co-simulation is performed to check the functionality and validity of generated RTL as the precision and the functionality may change after the synthesis. The synthesized module can be export as an Intellectual Property (IP) and used in other Xilinx tools. As the target is small embedded platform, therefore, the data type *fixed-point 16* are being used which have almost the same precision and accuracy results with respect to data types *float* and *double*.

Acknowledgements

Thanks to all those who shared this ride with me, those who have give me confidence, encouragement and hope in my life.

I would like to be thankful to Prof. Luciano Lavagno who gave me the extraordinary opportunity to work with him. During this time, I find, not only how good of a Prof. he is, but above all, how great of a human he is. I will always appreciate him for this wonderful opportunity.

I would like to thank my friends with whom I have cherished the happiness, joy and sadness which one experience in life.

I would like to pay gratitude and special thanks to my mother and siblings for always staying beside me in every moment of my life.

Last but not the least, My Father. A man with immense character and self-integrity. I will always be grateful to you. The person who planted a seed to have strong roots from which I have grown up as a man.

Contents

List of Figures

List of Tables

List of Acronyms

CPU	Central Processing Unit
FPGA	Field Programmable Gate Array
GPU	Graphical Processing Unit
HDL	Hardware Description Language
HLS	High Level Synthesis
IP	Intellectual Property
LSTM	Long Short Term Memory
PL	Programmable Logic
PLAN	Piece wise Linear Approximation
PNYQ	Python Productivity for ZYNQ
RNN	Recurrent Neural Network
RTL	Register Transfer Level
VHDL	VHSIC Hardware Description Language
Vivado HLS	Vivado High Level Synthesis

Chapter 1

Introduction

Artificial Intelligence will have a greater effect on our daily basis life to the extent that not every human being realize. Machine Learning is one of the branches of this tree, perhaps the most important one. It gives the power to a computer to learn from experience and and improve in the future. Basically, a computer can learn on its own. Neural Networks are kind of networks which are being inspired from the human brain. They are being used for many machine learning algorithms to train them for complex data. Long Short Term Memory, LSTM, networks are one of the type where this approach is used. They fall under the umbrella of Recurrent Neural Networks. They are being used in number of applications in our daily life. To use these in the applications, a powerful computational units are needed. Whether it is during the training or inference, a powerful hardware is needed. The high complexity of the computations limit the software implementation as it will take a long time. To make the computations faster, one can use Graphical Processing Units or multi-core Central Processing Units. But these target platforms are not cheap and consumes a lot of proper. Also, they are not ideal for an embedded platform. An alternative solution is to use a custom hardware accelerator which is specialized for the specific algorithm. This specialized accelerator can be implemented on an embedded system like FPGA's which consumes less power and are cost-efficient.

In this thesis, this work has been depicted by designing a customized hardware accelerator for the LSTM by using a High-Level Synthesis.

1.1 Design Flow

In this thesis, this work has been depicted by designing a customized hardware accelerator for the LSTM by using a High-Level Synthesis. The tool is provided by Xilinx which is called Vivado HLS. This accelerator needs to be implemented on a board. For this, PYNQ-Z2 board has been chosen. It has a dual-core Cortex A9 processor. Chapter 2 will explain the theory of LSTM and how this network works. Chapter 3 will present the

what is High-Level Synthesis and how you can use Vivado HLS tool to design a custom accelerator. Chapter 4 will tell how different hardware accelerator have been designed for different data types and what is the bottleneck problem in designing them for the LSTM. Chapter 5 will show the different solutions for different data types in terms of synthesis and optimizations. Finally, Chapter 6 will present final results and the accelerator chosen for LSTM and how these results are compared with C.P.U. and G.P.U. platforms.

Chapter 2

LSTM

When there is a larger network through time, gradient decays quickly during back propagation. This is called Long-Term dependency problem. Recurrent Neural Networks, commonly called RNNs, are not able to handle long term dependencies [1]. Long Short Term Memory networks are capable of learning long-term dependencies. Long Short Term Memory, referred as LSTM, networks are a kind of Neural Networks first proposed by Sepp Hochreiter and Jurgen Schmidhuber [2]. They allow to alleviate the vanishing gradient problem. A simple LSTM model has one hidden layer while a model, which is needed for applications with a larger target, has multiple hidden layers. A major issue in deep neural networks is that the gradient gets smaller with each layer until it reaches a point where it is too small to affect the deepest layers. LSTM consist of memory elements that allows it to remember values for long and short time. Because of these memory elements in LSTM, there is a continuous gradient flow. This way, vanishing gradient issue is resolved and it enables learning from data which is sequential and hundreds of time steps long [3]. A LSTM unit contains cells and gates through which the information flows. A general architecture of a LSTM unit is shown in Figure 2.1. LSTM are probably the best choice for the applications where the data is sequential. They have been applied extensively to the applications from speech recognition to data analytics.

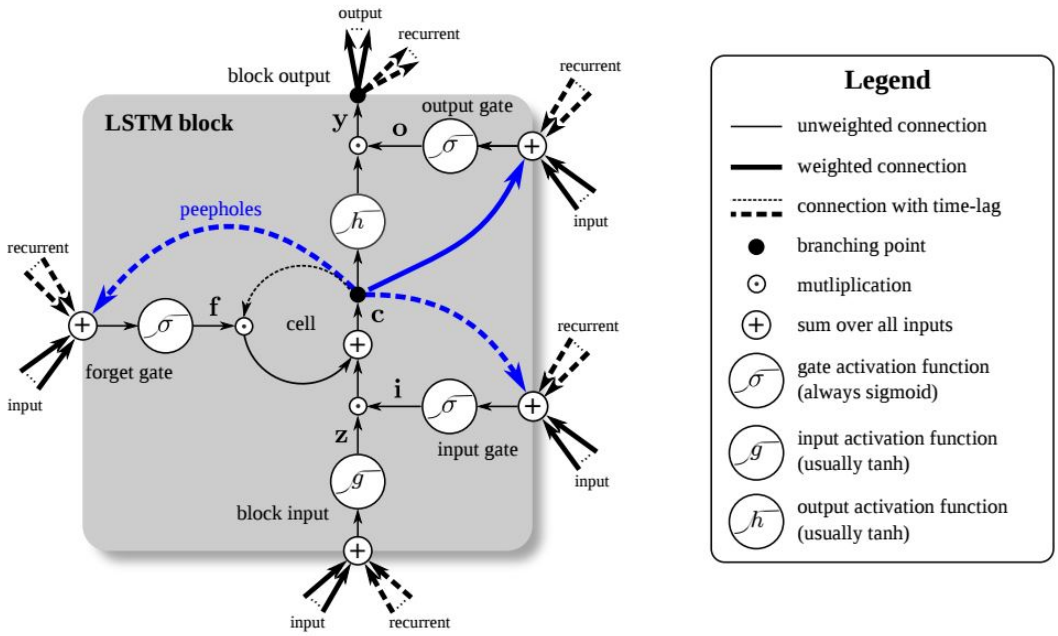


Figure 2.1: LSTM unit
[4]

2.1 General Architecture

A LSTM architecture consists of a *cell*, an *input gate*, an *output gate* and a *forget gate*. The cell acts like a memory where the data is stored over time and gates calculates an activation of a weighted sum. LSTM, like RNNS, do have a chain of modules which are repeated. These repeated modules have a different structure to the RNNs modules. In LSTM, there are four neural network layers connected with each other, as shown in 2.3, in comparison to RNNs where there is only one tanh layer, as shown in 2.4. Every single line represents a vector from one node to the other connected node. Circle represent the point-wise operation, like vector addition. Boxes are basically the trained neural network layers.

Figure 2.2 shows the notations used in LSTM and RNN modules.

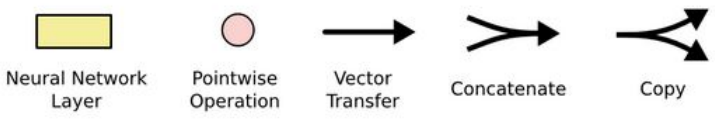


Figure 2.2: Notations for LSTM and RNN modules
[5]

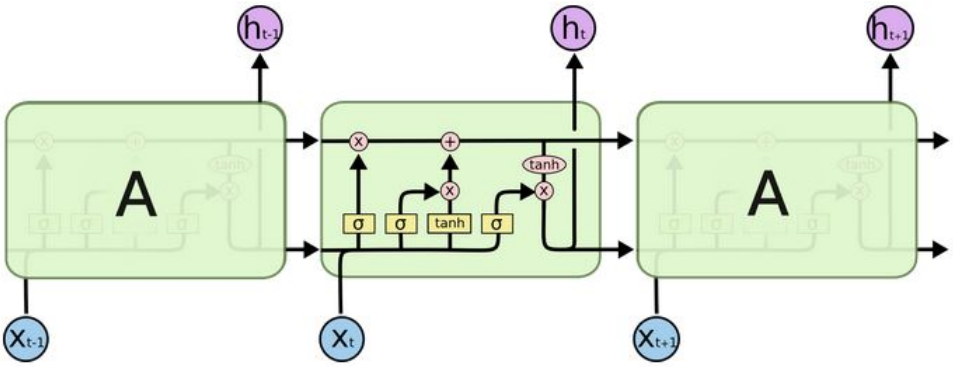


Figure 2.3: LSTM Module
[5]

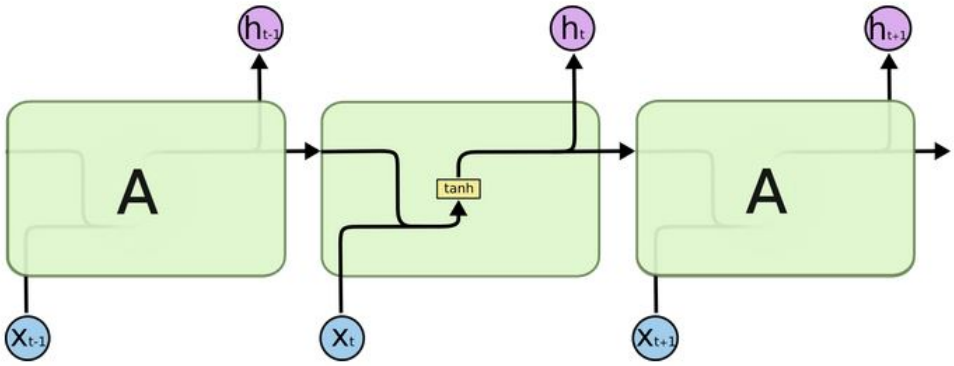


Figure 2.4: RNN Module
[5]

2.2 Fundamentals of LSTM Module

The central part of LSTM network is the memory element which is called *cell state*. The cell state act like a transportation for the flow of information. It take the information from the output of one node to the input of other node. It runs across the entire chain of modules with some linear interactions. Figure 2.5 shows the cell state in LSTM module.

It is possible to add or remove the information to the cell states. This is being done by gates. Gates are a way to let information go through the network. These gates are composed from neural net layer which is a sigmoid activation function and a pointwise multiplication operation. The sigmoid layer gives the outputs from *0* and *1*. This describes how much each element shall be let go through the network. A *Zero* means that no information is passed through and a *One* means that all the information is passed.

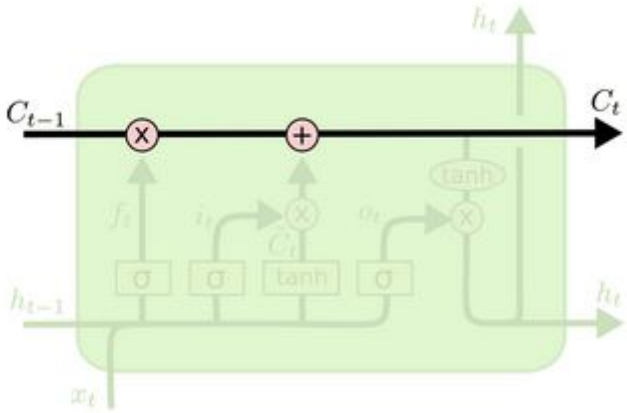


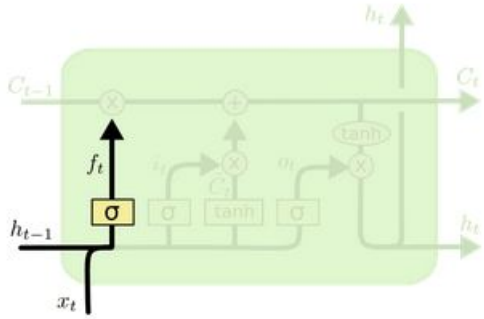
Figure 2.5: Cell State
[5]

LSTM network has three type of gates which are called:

- Forget Gate
- Input Gate
- Output Gate

2.2.1 Forget Gate

In this gate, it is decided what information needs to be thrown away from the cell gate. This is decided by a sigmoid layer which is called *forget gate layer*. It reads the current input x_t which is given to it and output h_{t-1} of the previous LSTM module and gives us a output from 0 to 1 for each element in the cell state C_{t-1}. A value of one means that this information is kept completely while the value of zero represents that the whole information is discarded. This helps in keeping away the unnecessary information which can be released into the network. Figure 2.6 shows how the forget gate is implemented in the LSTM unit.

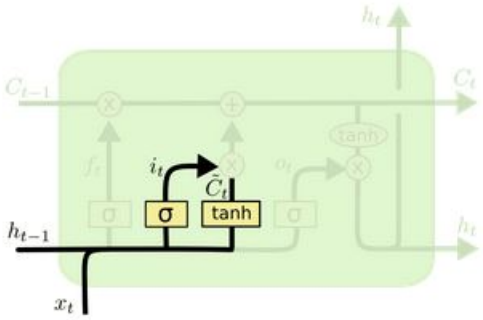


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.6: Forget Gate
[5]

2.2.2 Input Gate

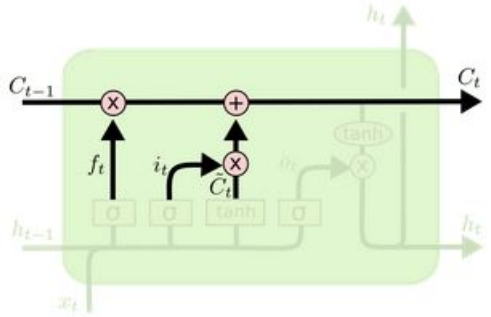
It is decided which new information is needed to be stored in the cell state in this step. This is done in two parts. First of all, a sigmoid layer, which is called the *"input gate layer"*, decides what values will be updated. Secondly, a tanh layer is used to create a new vector for new value $\tilde{C}_t$. This vector is added to the cell state. Figure 2.7 shows how this is done in the LSTM module.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.7: Input Gate (1)
[5]

Once the first part is done, then it's the time to update the old cell state C_{t-1} to a new cell state C_t. The old cell state C_{t-1} is multiplied by forget gate's activation vector f_t and added to i_t*$\tilde{C}_t$. i_t represents input gate's activation vector. This is new the value, which is scaled by how much we decided to update every cell state value. Figure 2.8 represents this operation in LSTM module.

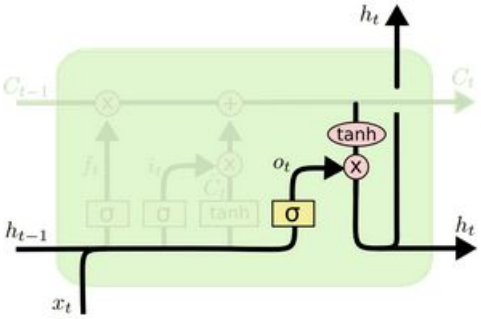


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.8: Input Gate (2)
[5]

2.2.3 Output Gate

In the *output gate*, it is decided which information is going to the output. This output is based on cell state but more of a filtered version. Here, a sigmoid layer decides which elements of the cell state is going to output. Afterwards, this cell state state is passed through tanh layer and multiplied by the output of the sigmoid output gate activation function. By multiplying, it is decided which parts of the information is passed through. Figure 2.9 shows how this is output gate is implemented in the LSTM module.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.9: Output Gate
[5]

2.3 Applications

LSTM's are being in the technology industry for a number of applications. They are being used for speech and voice recognition, handwriting generation and recognition, time series prediction and data analysis. These are just some applications where they are being chosen. *Google* use LSTM's for a numerous applications for their smart phones. Like from speech recognition on the Google phones, for the smart assistant

Allo to the Google translator [6] [7] [8]. *Apple* use it for the *QuickType* keyboard and *Siri* on the *iPhones* [9] [10]. *Amazon* use it for their voice assistant *Alexa* [11].

Chapter 3

High Level Synthesis using Vivado HLS

High Level Synthesis is an automated design process that transforms a high level functional specifications, generally in C/C++ or SystemC, to optimized RTL descriptions for efficient hardware implementation [12]. In this thesis, Vivado HLS tool is being used which is provided by Xilinx ®.

Xilinx High-Level Synthesis tool Vivado HLS transforms a C specifications into a Register Transfer Level (RTL) implementations that synthesizes into a Xilinx Field Programmable Gate Array (FPGA). Users can write C specifications in C, C++, SystemC or an OpenCL API C kernel. This FPGA provides a great parallel architecture with benefits in terms of cost, performance and power consumption over the traditional processors [13].

Basically, High-level synthesis works as a bridge between hardware and software providing the following benefits:

- Improved productivity for hardware designers. It gives flexibility to the hardware designer to work at a higher level of abstraction while creating high-performance hardware.
- It provides possibility to develop different multi-architectural solutions without changing the C specifications. This enables design space exploration and helps in finding the optimal implementation.
- Improved system performance for software designers. They have the possibility to accelerate the intensive parts of their algorithms, which take alot of computation, on a target which is FPGA.

3.1 Design Flow of Vivado HLS

Xilinx Vivado HLS tool synthesizes a C function into an IP block that can be integrated into a hardware system. This IP block can be designed as a hardware accelerator which

has been done in this thesis. It is well integrated with the other Xilinx design tools and provides language support and features for creating the optimal implementations of the C algorithm.

Vivado HLS design flow can be explained as follows:

- Compile, Execute and Debug the C algorithm. In HLS, compiling the C program is C simulation. Executing C algorithm simulates the function to validate whether the algorithm is functionally correct. This C function is the primary input to Vivado HLS.
- Synthesize the C algorithm into an RTL implementation. Optimization directives and constraints can be added to direct the synthesis process to implement a specific optimzation.
- Generate reports about hardware resource utilization to timing and analyze the design in every aspect.
- Verify the RTL implementation using a pushbutton flow. Vivado HLS uses the C test bench to simulate the C function prior to synthesis and to verify the RTL output using C/RTL Co-simulation.
- Package the RTL implementation into a selection of IP packages.

Figure 3.1 shows the high-level synthesis design flow in Vivado HLS.

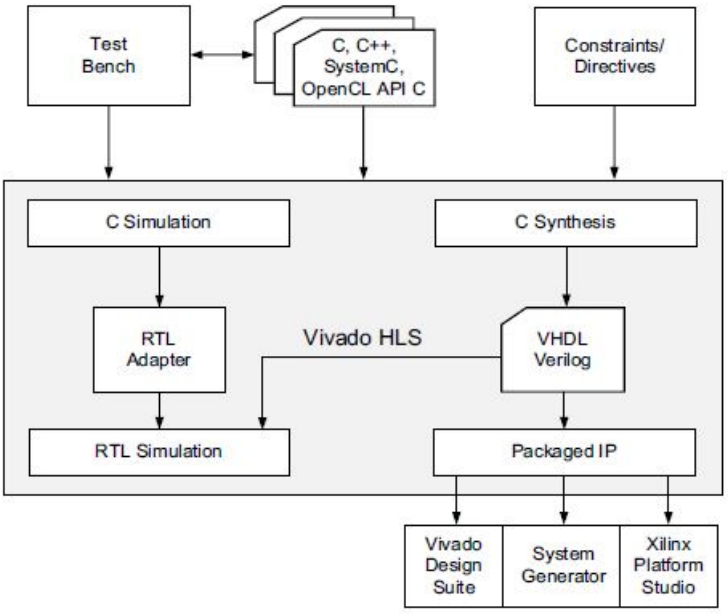


Figure 3.1: Vivado HLS Design Flow
[13]

3.2 Limitations of Vivado HLS

Vivado HLS supports a wide range of the C language but there are still some constructs which are not supported. Therefore, these constructs cannot be synthesized and can result in errors during the design flow. For the design to be synthesized, following changes must be done in the code.

- C function must contain the entire functionality of the design.
- None of the functionality can be performed by system calls to the operating system.
- C constructs must be of a fixed or bounded size.
- Implementations of those constructs must be unambiguous.

Following are the constructs which cannot be synthesized in Vivado HLS.

- **System Calls**: These cannot be synthesized as they are the actions that relate to performing some task upon the operating system in which the C program is running. Vivado HLS ignores system calls that have no impact on the execution of the algorithm. Examples are *getc()*, *printf()*, sleep() and *time()*.
- **Dynamic Memory Usage**: Memory allocation system calls must be removed from the design code before synthesis. Any system calls that manage memory allocation within the system, like, *alloc*, *free()* and *malloc()* are using resources that exist in the memory of the operating system and are created and released during run time.
- **Pointer Limitations**: Vivado HLS does not support general pointer casting, unless it between native C types. Function pointers are also not supported. But pointer arrays are supported for synthesis, provided that every pointer points to scalar, not to the other pointers.
- **Recursive Functions**: These cannot be synthesized. This applies to those functions which form endless recursion. This also applies to tail recursion in which there is a finite number of function calls.
- **Standard Template Libraries**: Many of the C++ standard template libraries use dynamic memory allocation and function recursion. Therefore, these cannot be synthesized as well. It is a good practice to create a local function which exhibits the same identical functionality but don't use the above previously mentioned characteristics.

3.3 Optimization Methodology

Vivado HLS provides a number of optimization's that are applied to C/C++ code through the use of directives and pragmas in the code. There are two flows for optimizing the hardware functions.

- **Top-down flow**
- **Bottom-down flow**

In this thesis, second flow has been used. In this flow, the hardware functions are optimized in isolation from the system using the Vivado HLS compiler provided in the Vivado Design suite. The hardware functions are analyzed, optimization directives can be applied to create an implementation other than the default, and the resulting optimized hardware functions are then incorporated into the SDSoC environment [14].

Figure 3.2 shows the detailed optimization methodology for hardware functions in Vivado HLS.

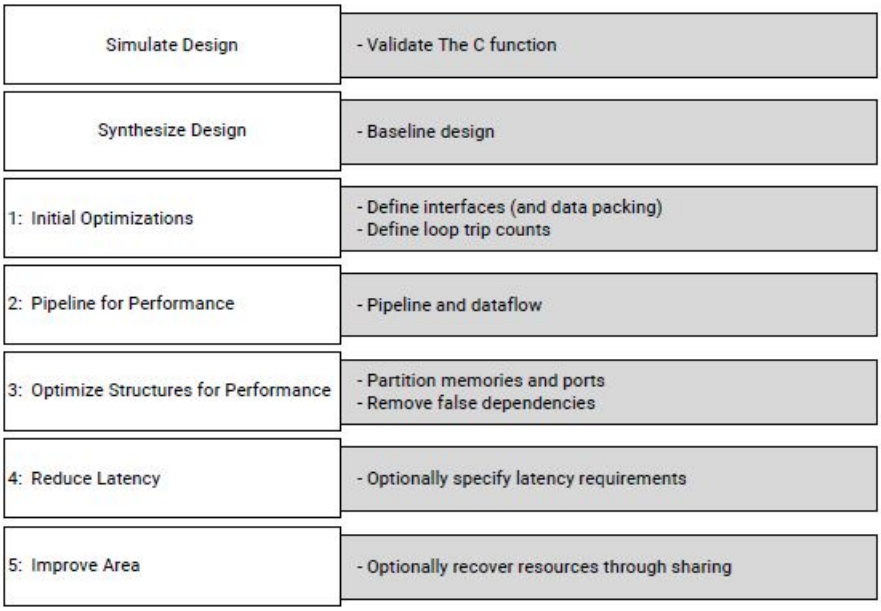


Figure 3.2: Vivado HLS Methodology
[14]

In order to get the optimize design, in terms of reducing the latency, improving the throughput, reducing the device resource utilizations of the resulting RTl code, different pragmas can be applied directly to the source code of the design. Some of the pragmas which have been used in this thesis are:

- **Array Partition**: This pragma partitions an array into smaller arrays or even individual elements. This results in RTL with multiple small memories instead of one large memory. It increase the amount of read and write ports. It potentially improves the throughput of the design. This is also requires more memory elements or registers on the FPGA.

- **Dependence**: This pragma provides additional information that can overcome loop-carry dependence and allow loops to be pipelined. Vivado HLS automatically detects dependencies within loops or between different iterations of a loop.

- **Interface**: It specifies how RTL ports are created from the function definition during interface synthesis. The

- **Pipeline**: It reduces the initiation interval for a function or loop by allowing the concurrent execution of operations. It is also possible to specify the initiation interval through the use of the II option for the pragma.

- **UNROLL**: It transforms loops by creating multiples copies of the loop body in the RTL design, which allows some or all loop iterations to occur in parallel. It also allows the loop to be fully or partially unrolled. Fully unrolling the loop creates a copy of the loop body in the RTL for each loop iteration, so the entire loop can be run concurrently. Partially unrolling a loop lets you specify a factor N, to create N copies of the loop body and reduce the loop iterations accordingly.

3.4 RTL Verification and Export

Vivado HLS uses the C test bench to simulate the functionality of the top-level function during the C simulation step. Afterwards, it reuses the C test bench to automatically verify the RTL output using C/RTL co-simulation. Vivado HLS automatically generate the files required to reuse the C test bench during the co-simulation step. When verification completes, the console displays message *SIM—1000* to confirm the verification was successful. If the C test bench returns a non-zero value, Vivado HLS reports that the simulation failed. Vivado HLS automatically creates the infrastructure to perform the C/RTL co-simulation and executes the simulation using one of the supported RTL simulators, which can be chosen independently, inside the tool.

The final step in the Vivado HLS design flow is to package the RTL output as an IP. It is possible to export the RTL and package the final RTL output files as IP in any of the following Xilinx IP formats:

- Vivado IP Catalog

- System Generator for DSP

- Synthesized Checkpoint

It is possible to execute logic synthesis from within Vivado HLS to evaluate the final results of RTL synthesis and implementation. This confirms the estimates provided by Vivado HLS for hardware utilizations and timing before handing off the IP package.

In this thesis work, C/RTL simulations has been performed during every single multi-architectural solution which confirmed the functionality of the VHDL code of each solution. Beside, the accuracy of the estimated values of hardware utilizations and timing was evaluated before exporting the design as an IP. RTL was exported successfully in each solution.

Chapter 4

Design of Hardware Accelerators using Vivado HLS

The main part of this thesis work is to design a hardware accelerator via C-based FPGA design tool provided by Xilinx which is Vivado HLS. The design flow, methodology and how the tool can be used to design the hardware accelerator has been explained in Chapter 3.

4.1 C Simulation

First of all, the C simulations were ran to validate the C++ specifications with the original data type *double*. Once validated, it was important to convert the data type from *double* to *float* and *fixed-points 16*. The reason behind was that the hardware accelerators, needed to be designed, were for embedded platform. To be specific, these hardware accelerators were being designed for *"signal processing and trajectory reconstruction in an indoor locationing application using capacitive sensors"*. After converting into data types *float* and *fixed-point 16*, C simulations were ran again to validate the C++ specifications.

4.2 Testing of the LSTM algorithm

Once the C simulations were performed and C++ specifications were validated, next step was to create a C-based test bench in order to verify the functionality of the algorithm. If there is no test bench used, the results obtained from C and RTL simulation may be different. One of the great thing about Vivado HLS is that it re-uses the C-based test bench to automatically verify the RTL output. This helps in saving time in terms of writing a test bench for RTL level. The test bench basically compared the results obtained during the C simulations with the *golden output file*. The golden output file

was generated during the training of the algorithm. Algorithm was trained with the data type *double*. Hence, there was no error when the results, achieved during the C simulation when the data type was double, compared with the output file. In case of *float*, % error was 0.0133%. For *fixed-point 16*, % error was 0.0136%. These results were pretty satisfying.

4.3 Pre-Synthesis

This is the most important part of the hardware accelerator design as the C code is being translated to RTL code, which can be in VHDL or Verilog depending upon the user preference. To perform this and implementing the accelerators, PYNQ-Z2 was chosen as target was the embedded platform. PYNQ-Z2 have a dual-core Cortex A9 processor. THe hardware resources present on the board are shown in Table 4.1.

Resources	BRAM_18K	DSP48E	FF	LUT
Quanity	280	220	106400	53200

Table 4.1: Resources on PYNQ-Z2

where *BRAM_18K* stands for Block RAM, *DSP48E* represents the digital signal processing slice, *FF* stands for flip-flops and *LUT* stands for Look up table.

Pre-synthesis was performed, which actually is a synthesis but it gives an estimated values of resource utilization and timing. This showed which components were using the most resources. Table 4.2 show the estimation results of resource usage in the case of data type *double*. The minimum latency was *3028235* clock cycles. The estimated time was *10.89* ns, which was well within the target time, *12.50* ns.

Resources	BRAM_18K	DSP48E	FF	LUT
Total Used	35	143	20222	29839
Available	280	220	106400	53200
Utilization %	12	65	19	56

Table 4.2: Utilization Estimate for *double*

Table 4.3 show the estimation results of resource usage in the case of data type *float*. The minimum latency was *2266635* clock cycles. The estimated time was *11.72* ns, which was well within the target time, *12.50* ns.

Table 4.4 show the estimation results of resource usage in the case of data type *float*. The maximum latency was *2391435* clock cycles. The estimated time was *11.31* ns, which was well within the target time, *12.50* ns.

Resources	BRAM_18K	DSP48E	FF	LUT
Total Used	19	72	8641	14772
Available	280	220	106400	53200
Utilization %	6	32	8	27

Table 4.3: Utilization Estimate for *float*

Resources	BRAM_18K	DSP48E	FF	LUT
Total Used	6	10	2718	3275
Available	280	220	106400	53200
Utilization %	2	4	2	6

Table 4.4: Utilization Estimate for *Fixed-point 16*

4.4 Implementation of PLAN

By looking at the result from the pre-synthesis, it was noted that the sigmoid activation function consumed the most resources. This sigmoid function in a non-linear. All in all, it took 90% of the total resources used. It was important to find out a solution for this. An efficient piece-wise linear approximation, PLAN, was used to replace it [15]. Figure 4.1 shows how it is implemented in the algorithm.

```
template<typename T>
const T sigmoid (const T in) {
    T tmp,tmp1;

    if (in < 0)
        {
            tmp = -in;
        } else
            tmp = in;

    if ( tmp >= (T) 5)
        {
            tmp1 = (T) 1;
        }
        else if ((T) 2.375 <= tmp && tmp < (T) 5)
        {
            tmp1 = (tmp * (T) 0.03125) + (T) 0.84375;
        }
        else if ((T) 1 <= tmp && tmp < (T) 2.375)
        {
            tmp1 = ((T) 0.125 * tmp) + (T) 0.625;
        }
        else if ((T) 0 <= tmp && tmp < (T) 1)
        {
            tmp1 = ((T) 0.25 * tmp) + (T) 0.5;
        }

    if ( in < (T) 0)
        {
            tmp1 = (T) 1 - tmp1;
        }

    return tmp1;
}
```

Figure 4.1: Piece-Wise Linear Approximation (PLAN)
[4]

After implementation of PLAN, C simulations were performed and C++ specifications were validated.

Chapter 5

Synthesis and Optimization

This chapter explains how Vivado HLS allow to explore different solutions during the synthesis phase. During this, by adding different directives or pragmas, it is also perform the optimizations for different solutions. When a new solution is created, it is possible to copy the directives and settings of the previous solutions to the present solution. User can add more directives to the present solution to get the desired result in terms of performance. It is also possible to compare different solutions which help in finding the optimal solution. This optimal solution can be find by comparing the resource usage and the performance achieved.

Following are the different solutions which were explored during the work. It was kept in mind that the PYNQ-Z2 have a limited number of hardware resources.

5.1 Solution 1

This solution is for data type *double*. In this solution, no directives were added in the design. The timing achieved after post synthesis was *11.04* ns and after post-implementation wa *11.65* ns. The minimum number of clock cycles were *2977305* c.c. The execution time was *37.21* ms. Table 5.1 shows the resource usage in this solution.

Resources	Used (VHDL)
SLICE	4561
LUT	13336
FF	11651
DSP	113
BRAM	30

Table 5.1: Summary of Resources Utilized

Figure 5.1 and Figure 5.2 shows the total resources utilization % after implementation and the power consumption.

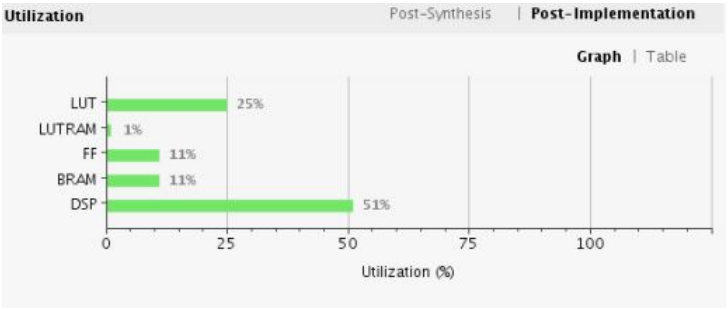


Figure 5.1: Utilization % Post-Implementation

Figure 5.2: Power Consumption

5.2 Solution 2

This solution is for data type *float*. In this solution, no directives were added in the design. The timing achieved after post synthesis was *11.32* ns and after post-implementation wa *11.64* ns. The minimum number of clock cycles were *2250635* c.c. The execution time was *28.13* ms. Table 5.2 shows the resource usage in this solution.

Resources	Used (VHDL)
SLICE	8980
LUT	12224
FF	24084
DSP	84
BRAM	13

Table 5.2: Summary of Resources Utilized

Figure 5.3 and Figure **??** shows the total resources utilization % after implementation and the power consumption.

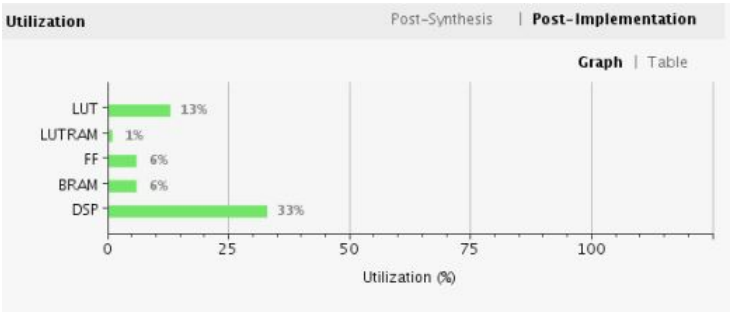


Figure 5.3: Utilization % Post-Implementation

Figure 5.4: Power Consumption

5.3 Solution 3

This solution is for data type *fixed-point 16*. In this solution, no directives were added in the design. The timing achieved after post synthesis was *10.48* ns and after post-implementation wa *12.37* ns. The minimum number of clock cycles were *625635* c.c. The execution time was *7.82* ms. Table 5.3 shows the resource usage in this solution.

Resources	Used (VHDL)
SLICE	3417
LUT	6354
FF	11880
DSP	75
BRAM	0

Table 5.3: Summary of Resources Utilized

Figure 5.5 and Figure 5.6 shows the total resources utilization % after implementation and the power consumption.

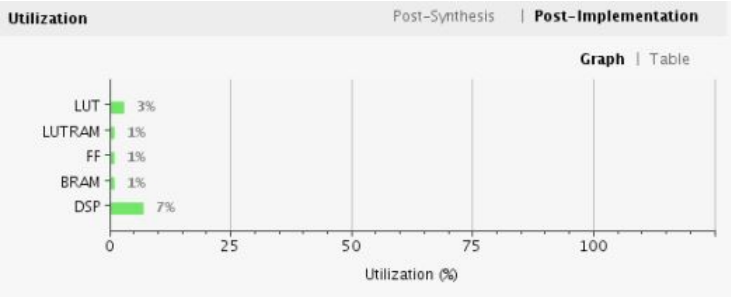


Figure 5.5: Utilization % Post-Implementation

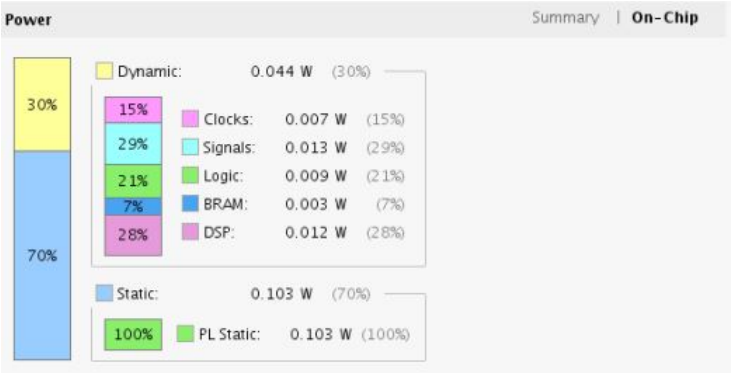


Figure 5.6: Power Consumption

5.4 Solution 4

This solution is for data type *double* where the directives were added to the design to optimize it. As there are memory elements in the LSTM unit, therefore,"ARRAY PARTITION" directive was added. By adding this directive, arrays were divided into small arrays to have multiple read and write operations. This resulted in increase of resource usage and reduction of latency and execution time. The timing achieved after post synthesis was *11.042* ns and after post-implementation was *12.27* ns. The minimum number of clock cycles were *1733767* c.c. The execution time was *21.67* ms. Table 5.4 shows the resource usage in this solution.

Figure 5.7 and Figure 5.8 shows the total resources utilization % after implementation and the power consumption.

Resources	Used (VHDL)
SLICE	13058
LUT	24888
FF	49116
DSP	144
BRAM	18

Table 5.4: Summary of Resources Utilized

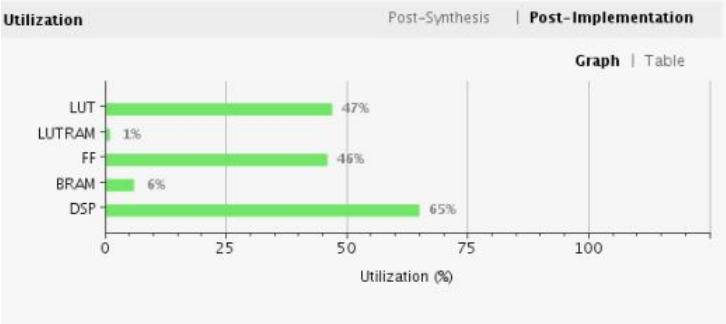


Figure 5.7: Utilization % Post-Implementation

Figure 5.8: Power Consumption

Directives like pipelining and unrolling were added to this design to explore more possibilities but in vain. The results achieved during the pre-synthesis, the estimated results, showed us that it would required more resource usage in comparison to the available ones on the board. The timing constraints were not met as well. As the target was the embedded platform, therefore, it wasn't possible to explore more solutions for data type *double*.

5.5 Solution 5

This solution is for data type *float* where the directives were added to the design to optimize it. "ARRAY PARTITION" directive was copied from the previous solution5.4. "PIPELINE" directive were also applied to smaller loops as well. This would help in reducing the initiation intervals of the loops. This resulted in increase of resource usage and reduction of latency and execution time. The timing achieved after post synthesis was *11.32* ns and after post-implementation was *11.63* ns. The minimum number of clock cycles were *1452137* c.c. The execution time was *18.15* ms. Table 5.5 shows the resource usage in this solution.

Resources	Used (VHDL)
SLICE	8980
LUT	12224
FF	24084
DSP	84
BRAM	13

Table 5.5: Summary of Resources Utilized

Figure 5.9 and Figure 5.10 shows the total resources utilization % after implementation and the power consumption.

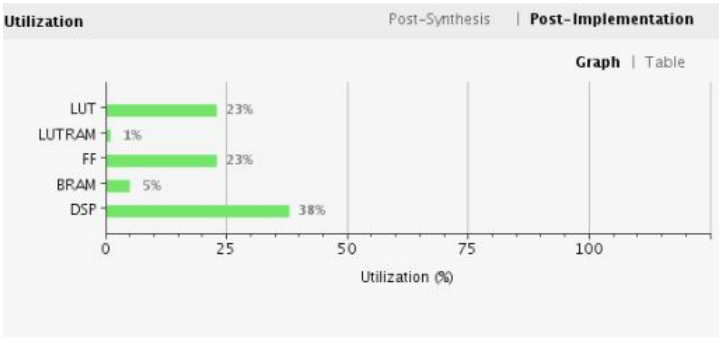


Figure 5.9: Utilization % Post-Implementation

Figure 5.10: Power Consumption

Pipelining was applied to bigger loops and unrolling was also applied to the design to explore more solutions to have better performance. The results achieved during the pre-synthesis, the estimated results, showed us that it would required more resource usage in comparison to the available ones on the board. As the target was the embedded platform, as in the previous solution, it wasn't possible to explore more solutions.

5.6 Solution 6

This solution is for data type *fixed-point 16* where the directives were added to the design to optimize it. The directives were copied from the previous solution 5.5. This resulted in increase of resource usage and reduction of latency and execution time. The timing achieved after post synthesis was *9.02* ns and after post-implementation was *10.14* ns. The minimum number of clock cycles were *542337* c.c. The execution time was *6.77* ms. Table 5.6 shows the resource usage in this solution.

Resources	Used (VHDL)
SLICE	3166
LUT	4998
FF	10170
DSP	16
BRAM	0

Table 5.6: Summary of Resources Utilized

Figure 5.11 and Figure 5.12 shows the total resources utilization % after implementation and the power consumption.

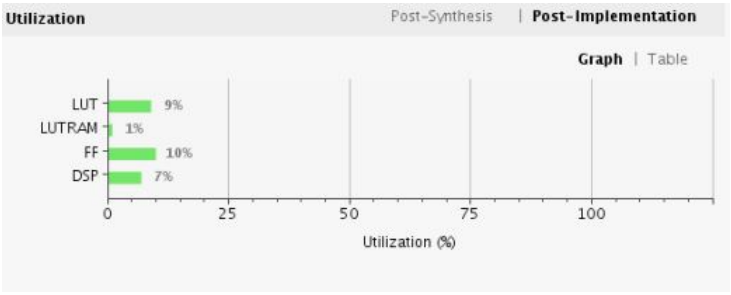


Figure 5.11: Utilization % Post-Implementation

Figure 5.12: Power Consumption

By looking at the results, it was realized that the design for *fixed-point 16* could be more optimized to find optimal solution as there were a lot of hardware resources still available. The next mentioned solutions explored this case.

5.7 Solution 7

This solution is for data type *fixed-point 16* where the directives were added to the design to optimize it. The directives were copied from the previous solution 5.6. "PIPELINE" directive was also applied to all the loops except the outer-most loop. This resulted in increase of resource usage and reduction of latency and execution time. The timing achieved after post synthesis was *10.48* ns and after post-implementation was *12.37* ns. The minimum number of clock cycles were *24737* c.c. The execution time was *0.30* ms. Table 5.7 shows the resource usage in this solution.

Figure 5.13 and Figure 5.14 shows the total resources utilization % after implementation and the power consumption.

Resources	Used (VHDL)
SLICE	3417
LUT	6354
FF	11880
DSP	75
BRAM	0

Table 5.7: Summary of Resources Utilized

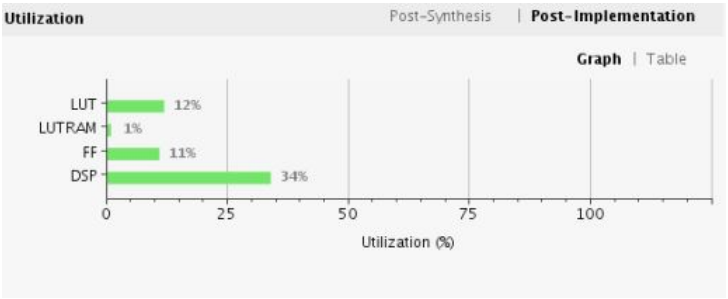


Figure 5.13: Utilization % Post-Implementation

Figure 5.14: Power Consumption

The results obtained were incredible. The execution time and latency were decreased 22x and 23x respectively. The resource utilization were increased though.

5.8 Solution 8

This solution is for data type *fixed-point 16* where the directives were added to the design to optimize it. The directives were copied from the previous solution 5.7. Beside the directives from previous solution, "UNROLL" directive was applied to the outer most loop by the factor of 4. The timing achieved after post synthesis was *10.48* ns and

after post-implementation was *12.28* ns. The minimum number of clock cycles were *23887* c.c. The execution time was *0.29* ms. Table 5.8 shows the resource usage in this solution.

Resources	Used (VHDL)
SLICE	3862
LUT	6886
FF	11990
DSP	75
BRAM	0

Table 5.8: Summary of Resources Utilized

Figure 5.15 and Figure 5.16 shows the total resources utilization % after implementation and the power consumption.

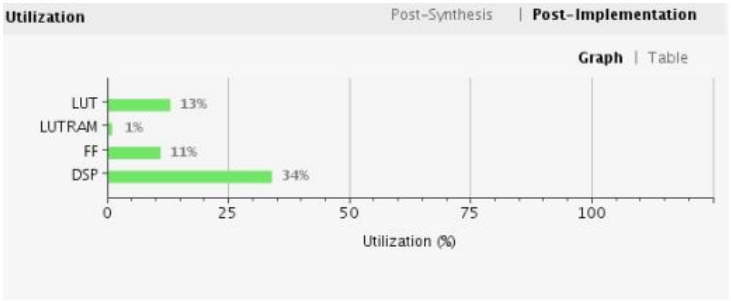


Figure 5.15: Utilization % Post-Implementation

Figure 5.16: Power Consumption

Latency and Execution time were decreased but not so much. Utilization of hardware resources were increased a bit as well.

5.9 Solution 9

This solution is for data type *fixed-point 16* where the directives were added to the design to optimize it. The directives were copied from the previous solution 5.8. Beside the directives from previous solution, "UNROLL" directive was applied to the outer most loop by the factor of 8. The timing achieved after post synthesis was *10.48* ns and after post-implementation was *12.34* ns. The minimum number of clock cycles were *22930* c.c. The execution time was *0.28* ms. Table 5.9 shows the resource usage in this solution.

Resources	Used (VHDL)
SLICE	3797
LUT	7172
FF	12184
DSP	75
BRAM	0

Table 5.9: Summary of Resources Utilized

Figure 5.17 and Figure 5.18 shows the total resources utilization % after implementation and the power consumption.

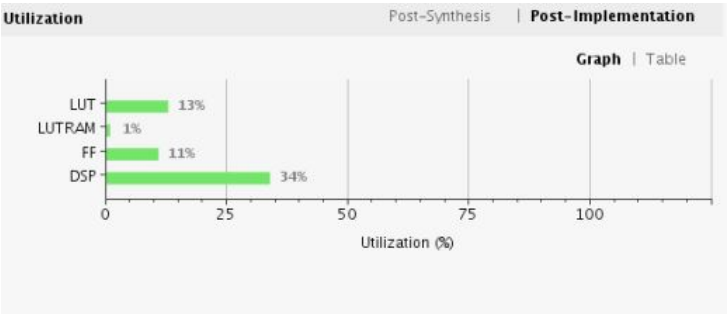


Figure 5.17: Utilization % Post-Implementation

Figure 5.18: Power Consumption

The increase of resource usage was barely minimal. The latency and execution time were decreased by *957* c.c. and *0.01* ms. By applying different other directives, either the resource usage would increase more than the available on the board or there wasn't much significant change in the latency and execution time.

Chapter 6

Conclusion

6.1 Summary

The work of this presented thesis is to design a hardware accelerator for the embedded platform in order to reduce the execution time and increase the throughput of the design. This Hardware accelerator is being designed for signal processing and trajectory reconstruction in an indoor locationing application using capacitive sensors.

6.2 Results

During this work, as explained in Chapter 5, different optimizations were performed for different data types to evaluate the latency and execution time.

For *double* data type, the optimal solution, considering the constraint in terms of hardware resources on the PNYQ-Z2, is Solution 4 in the chapter 5. The latency and execution time were decreased *2x*.

For *float* data type, the optimal solution, considering the constraint in terms of hardware resources on the PNYQ-Z2, is Solution 5 in the chapter 5. The latency and execution time were decreased *4x* and *2x* respectively.

For *fixed-point 16* data type, the optimal solution, considering the constraint in terms of hardware resources on the PNYQ-Z2, is Solution 9 in the chapter 5. The latency and execution time were decreased *27x* and *28x* respectively.

LSTM was also implemented and optimized on C.P.U. as well to calculate the execution time for different data types. The C.P.U., on which this was performed, was *Intel core i7-6900K @ 3.20 GHz*. The results are shown in Table 6.1.

As the target is small embedded platform, therefore, specialized accelerator adapted is of data type *fixed-point 16* which have almost the same precision and accuracy results with respect to data types *float* and *double*. The design space was explored while performing extensive *fixed-point 16* data type optimizations using the Vivado HLS tool.

Data Types	Execution Time (ms)
Double	0.682
Float	0.639
Fixed-Point 16	2.49

Table 6.1: Execution Time on C.P.U.

The optimal solution found is the one explained in Solution 9 of chapter 5. The execution time in comparison to the C.P.U. used is $9x$ lower.

Figure 6.1 shows how the design have been implemented on PYNQ-Z2.

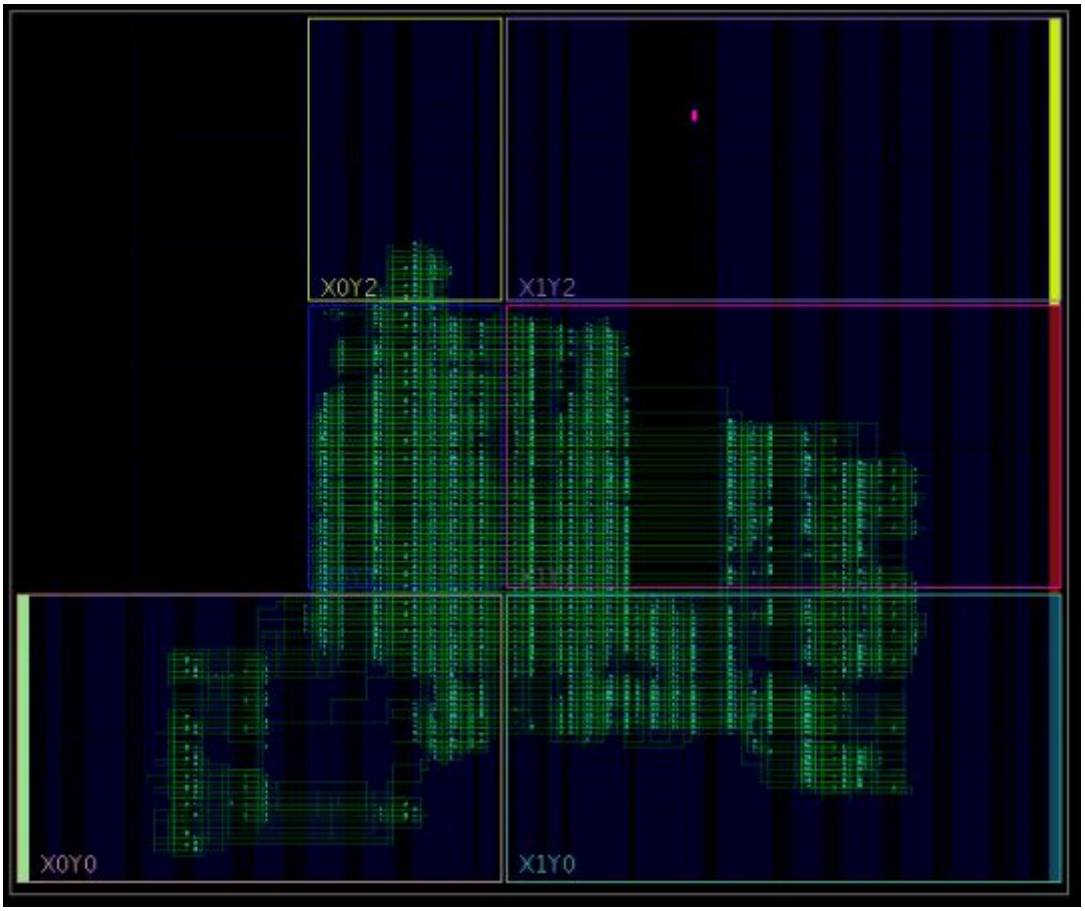


Figure 6.1: Implemented Design on PYNQ-Z2

In comparison to Graphic Processing Unit (GPU), the power consumption is much lower and the results obtained are comparable in terms of efficiency [16].

Appendix A

Reports

A.1 Resources Usage Summary

Listing A.1: Report Utilization Synthesis

```
Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------
| Tool Version : Vivado v.2018.2.1 (lin64) Build 2288692 Thu Jul 26 18:23:50 MDT 2018
| Host         : shaheen.polito.it running 64-bit CentOS release 6.10 (Final)
| Command      : report_utilization -file ./report/position_utilization_synth.rpt
| Design       : position
| Device       : 7z020clg400-1
| Design State : Synthesized
------------------------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+----------------------------+-------+-------+-----------+-------+
|          Site Type         |  Used | Fixed | Available | Util% |
+----------------------------+-------+-------+-----------+-------+
| Slice LUTs*                |  7212 |     0 |     53200 | 13.56 |
|   LUT as Logic             |  7119 |     0 |     53200 | 13.38 |
|   LUT as Memory            |    93 |     0 |     17400 |  0.53 |
|     LUT as Distributed RAM |     0 |     0 |           |       |
|     LUT as Shift Register  |    93 |     0 |           |       |
| Slice Registers            | 12184 |     0 |    106400 | 11.45 |
|   Register as Flip Flop    | 12184 |     0 |    106400 | 11.45 |
```

```
|   Register as Latch        |     0 |     0 |    106400 |  0.00 |
| F7 Muxes                   |   448 |     0 |     26600 |  1.68 |
| F8 Muxes                   |   144 |     0 |     13300 |  1.08 |
+----------------------------+-------+-------+-----------+-------+


1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
| 0     |          Yes |           - |            - |
| 0     |          Yes |           - |          Set |
| 0     |          Yes |           - |        Reset |
| 10    |          Yes |         Set |            - |
| 12174 |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+


2. Memory
---------

+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| Block RAM Tile |    0 |     0 |       140 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |       140 |  0.00 |
|   RAMB18       |    0 |     0 |       280 |  0.00 |
+----------------+------+-------+-----------+-------+


3. DSP
------

+----------------+------+-------+-----------+-------+
|    Site Type   | Used | Fixed | Available | Util% |
+----------------+------+-------+-----------+-------+
| DSPs           |   75 |     0 |       220 | 34.09 |
|   DSP48E1 only |   75 |       |           |       |
+----------------+------+-------+-----------+-------+


4. IO and GT Specific
---------------------

+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Bonded IOB                 |    0 |     0 |       125 |  0.00 |
| Bonded IPADs               |    0 |     0 |         2 |  0.00 |
| Bonded IOPADs              |    0 |     0 |       130 |  0.00 |
| PHY_CONTROL                |    0 |     0 |         4 |  0.00 |
| PHASER_REF                 |    0 |     0 |         4 |  0.00 |
| OUT_FIFO                   |    0 |     0 |        16 |  0.00 |
| IN_FIFO                    |    0 |     0 |        16 |  0.00 |
```

```
| IDELAYCTRL                    |    0 |     0 |         4 |  0.00 |
| IBUFDS                        |    0 |     0 |       121 |  0.00 |
| PHASER_OUT/PHASER_OUT_PHY     |    0 |     0 |        16 |  0.00 |
| PHASER_IN/PHASER_IN_PHY       |    0 |     0 |        16 |  0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY   |    0 |     0 |       200 |  0.00 |
| ILOGIC                        |    0 |     0 |       125 |  0.00 |
| OLOGIC                        |    0 |     0 |       125 |  0.00 |
+-------------------------------+------+-------+-----------+-------+


5. Clocking
-----------

+------------+------+-------+-----------+-------+
|  Site Type | Used | Fixed | Available | Util% |
+------------+------+-------+-----------+-------+
| BUFGCTRL   |    0 |     0 |        32 |  0.00 |
| BUFIO      |    0 |     0 |        16 |  0.00 |
| MMCME2_ADV |    0 |     0 |         4 |  0.00 |
| PLLE2_ADV  |    0 |     0 |         4 |  0.00 |
| BUFMRCE    |    0 |     0 |         8 |  0.00 |
| BUFHCE     |    0 |     0 |        72 |  0.00 |
| BUFR       |    0 |     0 |        16 |  0.00 |
+------------+------+-------+-----------+-------+


6. Specific Feature
-------------------

+-------------+------+-------+-----------+-------+
|  Site Type  | Used | Fixed | Available | Util% |
+-------------+------+-------+-----------+-------+
| BSCANE2     |    0 |     0 |         4 |  0.00 |
| CAPTUREE2   |    0 |     0 |         1 |  0.00 |
| DNA_PORT    |    0 |     0 |         1 |  0.00 |
| EFUSE_USR   |    0 |     0 |         1 |  0.00 |
| FRAME_ECCE2 |    0 |     0 |         1 |  0.00 |
| ICAPE2      |    0 |     0 |         2 |  0.00 |
| STARTUPE2   |    0 |     0 |         1 |  0.00 |
| XADC        |    0 |     0 |         1 |  0.00 |
+-------------+------+-------+-----------+-------+


7. Primitives
-------------

+----------+-------+----------------------+
| Ref Name |  Used | Functional Category |
+----------+-------+----------------------+
| FDRE     | 12174 |        Flop & Latch |
| LUT6     |  2364 |                  LUT |
| LUT3     |  2074 |                  LUT |
| LUT2     |  1482 |                  LUT |
| LUT5     |  1168 |                  LUT |
| LUT1     |   740 |                  LUT |
| CARRY4   |   635 |          CarryLogic |
| MUXF7    |   448 |               MuxFx |
| LUT4     |   357 |                  LUT |
| MUXF8    |   144 |               MuxFx |
| DSP48E1  |    75 |    Block Arithmetic |
| SRL16E   |    72 |  Distributed Memory |
| SRLC32E  |    21 |  Distributed Memory |
| FDSE     |    10 |        Flop & Latch |
```

```
+----------+-------+----------------------+


8. Black Boxes
--------------

+----------+------+
| Ref Name | Used |
+----------+------+


9. Instantiated Netlists
------------------------

+----------+------+
| Ref Name | Used |
+----------+------+
```

Listing A.2: Report Utilization Routed

```
Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2018.2.1 (lin64) Build 2288692 Thu Jul 26 18:23:50 MDT 2018
| Host         : shaheen.polito.it running 64-bit CentOS release 6.10 (Final)
| Command      : report_utilization -file ./report/position_utilization_routed.rpt
| Design       : position
| Device       : 7z020clg400-1
| Design State : Fully Placed
------------------------------------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Slice Logic Distribution
3. Memory
4. DSP
5. IO and GT Specific
6. Clocking
7. Specific Feature
8. Primitives
9. Black Boxes
10. Instantiated Netlists

1. Slice Logic
--------------

+----------------------------+-------+-------+-----------+-------+
|          Site Type         |  Used | Fixed | Available | Util% |
+----------------------------+-------+-------+-----------+-------+
| Slice LUTs                 |  7172 |     0 |     53200 | 13.48 |
|   LUT as Logic             |  7083 |     0 |     53200 | 13.31 |
|   LUT as Memory            |    89 |     0 |     17400 |  0.51 |
|     LUT as Distributed RAM |     0 |     0 |           |       |
|     LUT as Shift Register  |    89 |     0 |           |       |
| Slice Registers            | 12184 |     0 |    106400 | 11.45 |
|   Register as Flip Flop    | 12184 |     0 |    106400 | 11.45 |
|   Register as Latch        |     0 |     0 |    106400 |  0.00 |
| F7 Muxes                   |   448 |     0 |     26600 |  1.68 |
| F8 Muxes                   |   144 |     0 |     13300 |  1.08 |
```

```
+----------------------------+-------+-------+-----------+-------+
```

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
0	Yes	-	Reset
10	Yes	Set	-
12174	Yes	Reset	-

2. Slice Logic Distribution

Site Type	Used	Fixed	Available	Util%
Slice	3797	0	13300	28.55
SLICEL	2627	0		
SLICEM	1170	0		
LUT as Logic	7083	0	53200	13.31
using O5 output only	3			
using O6 output only	6013			
using O5 and O6	1067			
LUT as Memory	89	0	17400	0.51
LUT as Distributed RAM	0	0		
LUT as Shift Register	89	0		
using O5 output only	52			
using O6 output only	33			
using O5 and O6	4			
LUT Flip Flop Pairs	2612	0	53200	4.91
fully used LUT-FF pairs	795			
LUT-FF pairs with one unused LUT output	1792			
LUT-FF pairs with one unused Flip Flop	1733			
Unique Control Sets	284			

* Note: Review the Control Sets Report for more information regarding control sets.

3. Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	0	0	140	0.00
RAMB36/FIFO*	0	0	140	0.00
RAMB18	0	0	280	0.00

4. DSP

Site Type	Used	Fixed	Available	Util%
DSPs	75	0	220	34.09
DSP48E1 only	75			

5. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	0	0	125	0.00
Bonded IPADs	0	0	2	0.00
Bonded IOPADs	0	0	130	0.00
PHY_CONTROL	0	0	4	0.00
PHASER_REF	0	0	4	0.00
OUT_FIFO	0	0	16	0.00
IN_FIFO	0	0	16	0.00
IDELAYCTRL	0	0	4	0.00
IBUFDS	0	0	121	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	16	0.00
PHASER_IN/PHASER_IN_PHY	0	0	16	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	200	0.00
ILOGIC	0	0	125	0.00
OLOGIC	0	0	125	0.00

6. Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	0	0	32	0.00
BUFIO	0	0	16	0.00
MMCME2_ADV	0	0	4	0.00
PLLE2_ADV	0	0	4	0.00
BUFMRCE	0	0	8	0.00
BUFHCE	0	0	72	0.00
BUFR	0	0	16	0.00

7. Specific Feature

Site Type	Used	Fixed	Available	Util%
BSCANE2	0	0	4	0.00
CAPTUREE2	0	0	1	0.00
DNA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE2	0	0	1	0.00
ICAPE2	0	0	2	0.00
STARTUPE2	0	0	1	0.00

```
| XADC        |    0 |     0 |          1 |  0.00 |
+-------------+------+-------+------------+-------+


8. Primitives
-------------

+----------+-------+----------------------+
| Ref Name |  Used | Functional Category |
+----------+-------+----------------------+
| FDRE     | 12174 |        Flop & Latch |
| LUT6     |  2364 |                 LUT |
| LUT3     |  2074 |                 LUT |
| LUT2     |  1482 |                 LUT |
| LUT5     |  1168 |                 LUT |
| LUT1     |   705 |                 LUT |
| CARRY4   |   635 |          CarryLogic |
| MUXF7    |   448 |               MuxFx |
| LUT4     |   357 |                 LUT |
| MUXF8    |   144 |               MuxFx |
| SRL16E   |    78 |  Distributed Memory |
| DSP48E1  |    75 |    Block Arithmetic |
| SRLC32E  |    15 |  Distributed Memory |
| FDSE     |    10 |        Flop & Latch |
+----------+-------+----------------------+


9. Black Boxes
--------------

+----------+------+
| Ref Name | Used |
+----------+------+


10. Instantiated Netlists
-------------------------

+----------+------+
| Ref Name | Used |
+----------+------+
```

A.2 Timing Reports

Listing A.3: Timing Report Synthesis

```
------------------------------------------------------------------------------------
Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
| Tool Version : Vivado v.2018.2.1 (lin64) Build 2288692 Thu Jul 26 18:23:50 MDT 2018
| Host         : shaheen.polito.it running 64-bit CentOS release 6.10 (Final)
| Command      : report_timing -file ./report/position_timing_synth.rpt
| Design       : position
| Device       : 7z020-clg400
| Speed File   : -1  PRODUCTION 1.11 2014-09-11
------------------------------------------------------------------------------------

Timing Report

Slack (MET) :             2.018ns  (required time - arrival time)
```

```
  Source:                 grp_feed_forward_fu_1066/grp_matMul_2_fu_204/tmp_3_fu_467_p2/CLK
(rising edge-triggered cell DSP48E1 clocked by ap_clk  {rise@0.000ns fall@6.250ns period=12.500ns})
  Destination:            grp_feed_forward_fu_1066/grp_matMul_2_fu_204/p_Val2_29_6_fu_626_p2/C[10]
(rising edge-triggered cell DSP48E1 clocked by ap_clk  {rise@0.000ns fall@6.250ns period=12.500ns})
  Path Group:             ap_clk
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            12.500ns  (ap_clk rise@12.500ns - ap_clk rise@0.000ns)
  Data Path Delay:        8.697ns  (logic 5.489ns (63.115%)  route 3.208ns (36.885%))
  Logic Levels:           8  (CARRY4=5 DSP48E1=2 LUT2=1)
  Clock Path Skew:        -0.049ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    0.924ns = ( 13.424 - 12.500 )
    Source Clock Delay      (SCD):    0.973ns
    Clock Pessimism Removal (CPR):    0.000ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
  ----------------------------------------------------------------------

                        (clock ap_clk rise edge)     0.000     0.000
    Location            Delay type                Incr(ns)  Path(ns)
                                                     0.000     0.000
                        net (fo=12351, unset)        0.973     0.973
                        DSP48E1
  ----------------------------------------------------------------------
                        DSP48E1 (Prop_dsp48e1_CLK_P[25])
                                                     0.434     1.407
                        net (fo=1, unplaced)         0.800     2.207
                        DSP48E1 (Prop_dsp48e1_C[25]_P[10])
                                                     1.820     4.027
                        net (fo=2, unplaced)         0.800     4.826
                        LUT2 (Prop_lut2_I0_O)        0.124     4.950
                        net (fo=1, unplaced)         0.000     4.950
                        CARRY4 (Prop_carry4_S[1]_CO[3])
                                                     0.533     5.483
                        net (fo=1, unplaced)         0.009     5.492
                        CARRY4 (Prop_carry4_CI_CO[3])
                                                     0.117     5.609
                        net (fo=1, unplaced)         0.000     5.609
                        CARRY4 (Prop_carry4_CI_CO[3])
                                                     0.117     5.726
                        net (fo=1, unplaced)         0.000     5.726
                        CARRY4 (Prop_carry4_CI_CO[3])
                                                     0.117     5.843
                        net (fo=1, unplaced)         0.000     5.843
                        CARRY4 (Prop_carry4_CI_O[0])
                                                     0.232     6.075
                        net (fo=1, unplaced)         0.800     6.875
                        DSP48E1 (Prop_dsp48e1_C[25]_P[10])
                                                     1.995     8.870
                        net (fo=1, unplaced)         0.800     9.670
                        DSP48E1
  ----------------------------------------------------------------------

                        (clock ap_clk rise edge)    12.500    12.500
                                                     0.000    12.500
                        net (fo=12351, unset)        0.924    13.424
                        DSP48E1
                        clock pessimism              0.000    13.424
                        clock uncertainty           -0.035    13.389
                        DSP48E1 (Setup_dsp48e1_CLK_C[10])
                                                    -1.701    11.688
```

```
  ------------------------------------------------------------------
                        required time                         11.688
                        arrival time                          -9.670
  ------------------------------------------------------------------
                        slack                                  2.018
```

Listing A.4: Timing Report Routed

```
Copyright 1986-2018 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------
| Tool Version : Vivado v.2018.2.1 (lin64) Build 2288692 Thu Jul 26 18:23:50 MDT 2018
| Host         : shaheen.polito.it running 64-bit CentOS release 6.10 (Final)
| Command      : report_timing_summary -file ./report/position_timing_routed.rpt
| Design       : position
| Device       : 7z020-clg400
| Speed File   : -1  PRODUCTION 1.11 2014-09-11
------------------------------------------------------------------------------------

Timing Summary Report

------------------------------------------------------------------------------------------------
| Timer Settings
| --------------
------------------------------------------------------------------------------------------------

  Enable Multi Corner Analysis               :  Yes
  Enable Pessimism Removal                   :  Yes
  Pessimism Removal Resolution               :  Nearest Common Node
  Enable Input Delay Default Clock           :  No
  Enable Preset / Clear Arcs                 :  No
  Disable Flight Delays                      :  No
  Ignore I/O Paths                           :  No
  Timing Early Launch at Borrowing Latches   :  false

  Corner  Analyze    Analyze
  Name    Max Paths  Min Paths
  ------  ---------  ---------
  Slow    Yes        Yes
  Fast    Yes        Yes


All user specified timing constraints are met.

------------------------------------------------------------------------------------------------
| Timing Details
| --------------
------------------------------------------------------------------------------------------------


------------------------------------------------------------------------------------------------
From Clock:  ap_clk
  To Clock:  ap_clk

Setup :            0  Failing Endpoints,  Worst Slack        0.155ns,  Total Violation
0.000ns
Hold  :            0  Failing Endpoints,  Worst Slack        0.058ns,  Total Violation
0.000ns
PW    :            0  Failing Endpoints,  Worst Slack        5.270ns,  Total Violation
0.000ns
------------------------------------------------------------------------------------------------
```

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             0.155ns  (required time - arrival time)
Source:          grp_lstm_fu_844/grp_matMul_1_fu_1459/position_mac_mulag8j_U76/
             position_mac_mulag8j_DSP48_5_U/p/CLK
(rising edge-triggered cell DSP48E1 clocked by ap_clk  {rise@0.000ns fall@6.250ns period=12.500ns})
Destination:       grp_lstm_fu_844/grp_matMul_1_fu_1459/position_mac_mulag8j_U79/
             position_mac_mulag8j_DSP48_5_U/p/C[28]
(rising edge-triggered cell DSP48E1 clocked by ap_clk  {rise@0.000ns fall@6.250ns period=12.500ns})
  Path Group:             ap_clk
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            12.500ns  (ap_clk rise@12.500ns - ap_clk rise@0.000ns)
  Data Path Delay:        12.028ns  (logic 7.649ns (63.592%)  route 4.379ns (36.408%))
  Logic Levels:           2  (DSP48E1=2)
  Clock Path Skew:        -0.049ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    0.924ns = ( 13.424 - 12.500 )
    Source Clock Delay      (SCD):    0.973ns
    Clock Pessimism Removal (CPR):    0.000ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns

    Location             Delay type                Incr(ns)  Path(ns)
  ----------------------------------------------------------------------
                         (clock ap_clk rise edge)     0.000     0.000
                                                      0.000     0.000
                         net (fo=12351, unset)        0.973     0.973
    DSP48_X3Y16          DSP48E1
  ----------------------------------------------------------------------
    DSP48_X3Y16          DSP48E1 (Prop_dsp48e1_CLK_P[25])
                                                      4.009     4.982
                         net (fo=23, routed)          0.974     5.956
    DSP48_X3Y17          DSP48E1 (Prop_dsp48e1_C[45]_P[25])
                                                      1.820     7.776
                         net (fo=23, routed)          2.274    10.050
    DSP48_X2Y17          DSP48E1 (Prop_dsp48e1_C[45]_P[25])
                                                      1.820    11.870
                         net (fo=23, routed)          1.131    13.001
    DSP48_X2Y18          DSP48E1
  ----------------------------------------------------------------------

                         (clock ap_clk rise edge)    12.500    12.500
                                                      0.000    12.500
                         net (fo=12351, unset)        0.924    13.424
    DSP48_X2Y18          DSP48E1
                         clock pessimism              0.000    13.424
                         clock uncertainty           -0.035    13.389
    DSP48_X2Y18          DSP48E1 (Setup_dsp48e1_CLK_C[28])
                                                     -0.233    13.156
  ----------------------------------------------------------------------
                         required time                         13.156
                         arrival time                         -13.001
  ----------------------------------------------------------------------
                         slack                                 0.155



Min Delay Paths
--------------------------------------------------------------------------------------
```

```
Slack (MET) :             0.058ns  (arrival time - required time)
  Source: grp_lstm_fu_844/tmp_s_reg_7576_reg[0]/C
(rising edge-triggered cell FDRE clocked by ap_clk  {rise@0.000ns fall@6.250ns period=12.500ns})
  Destination: grp_lstm_fu_844/tmp_s_reg_7576_pp0_iter33_reg_reg[0]_srl32/D
(rising edge-triggered cell SRLC32E clocked by ap_clk  {rise@0.000ns fall@6.250ns period=12.500ns})
  Path Group:             ap_clk
  Path Type:              Hold (Min at Fast Process Corner)
  Requirement:            0.000ns  (ap_clk rise@0.000ns - ap_clk rise@0.000ns)
  Data Path Delay:        0.197ns  (logic 0.141ns (71.611%)  route 0.056ns (28.389%))
  Logic Levels:           0
  Clock Path Skew:        0.022ns (DCD - SCD - CPR)
    Destination Clock Delay (DCD):    0.432ns
    Source Clock Delay      (SCD):    0.410ns
    Clock Pessimism Removal (CPR):    -0.000ns

    Location             Delay type                Incr(ns)  Path(ns)
  ----------------------------------------------------------------------
                         (clock ap_clk rise edge)     0.000     0.000
                                                      0.000     0.000
                         net (fo=12351, unset)        0.410     0.410
    SLICE_X63Y64         FDRE
  ----------------------------------------------------------------------
    SLICE_X63Y64         FDRE (Prop_fdre_C_Q)         0.141     0.551
                         net (fo=1, routed)           0.056     0.607
    SLICE_X62Y64         SRLC32E
  ----------------------------------------------------------------------

                         (clock ap_clk rise edge)     0.000     0.000
                                                      0.000     0.000
                         net (fo=12351, unset)        0.432     0.432
    SLICE_X62Y64         SRLC32E
                         clock pessimism              0.000     0.432
    SLICE_X62Y64         SRLC32E (Hold_srlc32e_CLK_D)
                                                      0.117     0.549
  ----------------------------------------------------------------------
                         required time                          -0.549
                         arrival time                            0.607
  ----------------------------------------------------------------------
                         slack                                   0.058



Pulse Width Checks
--------------------------------------------------------------------------------------
Clock Name:         ap_clk
Waveform(ns):       { 0.000 6.250 }
Period(ns):         12.500
Sources:            { ap_clk }

Check Type        Corner  Lib Pin      Reference Pin  Required(ns)  Actual(ns)  Slack(ns)
Min Period        n/a     DSP48E1/CLK  n/a            3.884         12.500      8.616
Low Pulse Width   Fast    SRL16E/CLK   n/a            0.980         6.250       5.270
High Pulse Width  Slow    SRL16E/CLK   n/a            0.980         6.250       5.270
```

Appendix B

LSTM C++ Code

B.1 lstm_sensor.cpp

```
#include "lstm_hls.h"
#include "params.h"
#include "ap_fixed.h"
#include "ap_int.h"
#include "hls_math.h"
#include "hls_half.h"
#include <cmath>
#include <cstdlib>

void position(const DataType input[DataSize][Capa_In_Size + Infrared_In_Size],
DataType p[DataSize][Output_Size]){

        static const int InputSize = Capa_In_Size + Infrared_In_Size;
        static const int LSTM_SIZE = 16;

        static const DataType WI[InputSize][LSTM_SIZE * 4] ={
#include "expr_0.txt"
        };
        static const DataType WS[LSTM_SIZE][LSTM_SIZE * 4] = {
#include "expr_1.txt"
        };
        static const DataType BiasS[LSTM_SIZE * 4]={
#include "expr_2.txt"
        };

        static const DataType W1[LSTM_SIZE][Output_Size] ={
```

```
#include "expr_3.txt"
        };
        static const DataType Bias1[Output_Size]={
#include "expr_4.txt"
        };

        DataType state[LSTM_SIZE];

        for(int k=0; k<LSTM_SIZE;k++){

         state[k] = 0;
        }

        DataType output[LSTM_SIZE];

        for(int j=0; j<LSTM_SIZE;j++){

         output[j] = 0;
        }

        for(int i=0; i<DataSize;i++){

        lstm<InputSize, LSTM_SIZE>(input[i], state, output, BiasS, WI, WS);

        feed_forward<LSTM_SIZE, Output_Size>(output, p[i], W1, Bias1);

        }
}
```

B.2 lstm.h

```
#include <iostream>
#include <cstdlib>
using namespace std;
#pragma once
#include <cmath>
#include "ap_fixed.h"
#include "ap_int.h"
#include "hls_half.h"
#include "hls_math.h"
```

```
/*template<typename T>
const T sigmoid(const T value){
        T tmp = T(1)/(T(1) + exp(-value));
        return tmp;
}*/

template<typename T>
const T sigmoid (const T in) {
        T tmp,tmp1;

        if (in < 0)
                {
                        tmp = -in;
                } else
                        tmp = in;

        if ( tmp >= (T) 5)
                {
                        tmp1 = (T) 1;
                }
                else if ((T) 2.375 <= tmp && tmp < (T) 5)
                {
                        tmp1 = (tmp * (T) 0.03125) + (T) 0.84375;
                }
                else if ((T) 1 <= tmp && tmp < (T) 2.375)
                {
                        tmp1 = ((T) 0.125 * tmp) + (T) 0.625;
                }
                else if ((T) 0 <= tmp && tmp < (T) 1)
                {
                        tmp1 = ((T) 0.25 * tmp) + (T) 0.5;
                }

        if ( in < (T) 0)
                {
                        tmp1 = (T) 1 - tmp1;
                }

        return tmp1;
}
```

```
template<typename T>
const T tanh(const T in) {
        T tmp;
        tmp = (hls::exp(in) -  hls::exp(-in)) / (hls::exp(in) + hls::exp(-in));

        return tmp;
}

template<int H, int W, typename T>
void matMul(const T w[W][H], const T z[W], T o[H]){

        T tmp;
        for(int i=0;i<H;i++){

                for(int j=0;j<W;j++){
                        if (j==0){
                        tmp = T(0);
                        }

                tmp += (w[j][i] * z[j]);

                }
                o[i] = tmp;

        }
}

template<int N, typename T>
void add(const T a0[N], const T a1[N], T a2[N]){
        for(int i=0;i<N;i++){

                a2[i] = a0[i] + a1[i] ;

        }
}

template<int N, typename T>
void add(const T a0[N], const T a1[N], const T a2[N], T a3[N]){
        for(int i=0;i<N;i++){

                a3[i] = a0[i] + a1[i] + a2[i];
```

```
        }
}

template<int INPUT_SIZE, int LSTM_SIZE, typename T>
void lstm(const T input[INPUT_SIZE],T state[LSTM_SIZE], T output[LSTM_SIZE],
const T BIAS[LSTM_SIZE * 4], const T WI[INPUT_SIZE][LSTM_SIZE * 4],
const T WS[LSTM_SIZE][LSTM_SIZE * 4]){

        T gates[LSTM_SIZE * 4];

        T out1[LSTM_SIZE * 4];

        T out2[LSTM_SIZE * 4];

        matMul<LSTM_SIZE * 4, INPUT_SIZE>(WI, input, out1);

        matMul<LSTM_SIZE * 4, LSTM_SIZE>(WS, output, out2);

        add<LSTM_SIZE * 4>(out1, out2, BIAS, gates);

        for(int i=0;i<LSTM_SIZE;i++){

        state[i] = sigmoid(gates[LSTM_SIZE + i]) * state[i] +
        sigmoid(gates[i])* tanh(gates[LSTM_SIZE*2 + i]);
        output[i] = sigmoid(gates[LSTM_SIZE*3 + i]) * tanh(state[i]);

        }
}

template<int INPUT_SIZE, int OUTPUT_SIZE, typename T>
void feed_forward(const T input[INPUT_SIZE], T output[OUTPUT_SIZE],
const T W1[INPUT_SIZE][OUTPUT_SIZE],const T BIAS1[OUTPUT_SIZE]){

        T tmp[OUTPUT_SIZE];

        matMul<OUTPUT_SIZE, INPUT_SIZE>(W1, input, tmp);

        add<OUTPUT_SIZE>(tmp, BIAS1, output);
        }
```

Bibliography

[1] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Trans. Neur. Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994. [Online]. Available: http://dx.doi.org/10.1109/72.279181

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[3] Nvidia Accelerated Computing, "Long Short-Term Memory (LSTM)," https://developer.nvidia.com/discover/lstm.

[4] K. Greff, R. Kumar Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, 03 2015.

[5] Christopher, Olah, "Understanding LSTM Networks," https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[6] Françoise, Beaufays, "The neural networks behind Google Voice transcription ," https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html.

[7] Pranav, Khaitan, "Chat Smarter with Allo," https://ai.googleblog.com/2016/05/chat-smarter-with-allo.html.

[8] Cade, Metz, "An Infusion of AI Makes Google Translate More Powerful Than Ever," https://www.wired.com/2016/09/google-claims-ai-breakthrough-machine-translation/.

[9] Amir, Efrati, "AppleâĂŹs Machines Can Learn Too," https://www.theinformation.com/articles/apples-machines-can-learn-too.

[10] Chris, Smith, "iOS 10: Siri now works in third-party apps, comes with extra AI features," https://bgr.com/2016/06/13/ios-10-siri-third-party-apps/.

[11] Werner, Vogels, "Bringing the Magic of Amazon AI and Alexa to Apps on AWS," https://www.allthingsdistributed.com/2016/11/amazon-ai-and-alexa-for-all-aws-apps.html.

[12] P. Coussy and A. Morawiec, *High-Level Synthesis From Algorithm to Digital Circuit*. Springer Netherlands, 2008.

[13] Xilinx®, "Vivado Design Suite User Guide: High-Level Synthesis," 2018.

[14] Xilinx, "Vivado HLS Optimization Methodology Guide," 2018.

[15] H. Amin, K. Curtis, and B. Hayes-Gill, "Piecewise linear approximation applied to

nonlinear function of a neural network," *Circuits, Devices and Systems, IEE Proceedings* -, vol. 144, pp. 313 – 317, 01 1998.

[16] F. Muslim, L. Ma, M. Roozmeh, and L. Lavagno, "Efficient fpga implementation of opencl high-performance computing applications via high-level synthesis," *IEEE Access*, vol. PP, pp. 1–1, 02 2017.