Hardware Acceleration for LSTM Neural Networks

Alveera Gill
1215298240
agill17@asu.edu

Sanchari Datta
1215306118
sdatta24@asu.edu

Oindrila Das
1215037681
odas2@asu.edu

***Abstract*—Machine Learning is a fast progressing domain that requires competent hardware to keep up with it. The aim of this project is to implement hardware acceleration techniques on the circuit of a basic LSTM model. Focusing on various circuit optimization techniques, architectural techniques, etc., this project aims at improving throughput as much as possible while keeping an eye on the power consumption.**

Keywords—LSTM, word length, circuit optimization, parallelism, unfolding, time multiplexing, throughput, power consumption

I. INTRODUCTION

Machine Learning is a fast-evolving concept that constantly requires the machinery and hardware to keep up with it. One such commonly used machine learning algorithm is the Long Short Term Memory Network. The LSTM model is mainly used to predict numbers or strings based on past inputs and past predicted values. The model takes a concatenated input comprising of the output of the last 3 time steps and the current input. The number of previous inputs here is selected as 3 as it provides better accuracy when predicting the output while it also reduces the burden on the hardware compared to if more inputs were to be selected. The output received after a cycle corresponds to a prediction of what the value should be in the next time instant.

Figure 1 shows the internal structure of an LSTM Network. The network works in the following manner:

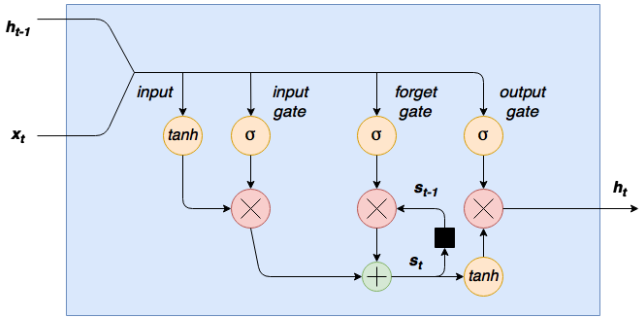

Figure 1: Components of an LSTM Network

Firstly, the input is confined between -1 and 1 using a tanh activating function. Along with the format of the inputs taken in, this can be represented as:

$$g = \tanh(b_g + x_t U_g + h_{t-1} V_g)$$

Where U_g, V_g, W_g and Z_g are the weights of x_t, h_{t-1}, h_{t-2}, h_{t-3} where x_t is the current input, h_{t-1}, h_{t-2} and h_{t-3} are outputs from the respective previous time instants. Another term that is added here is B_g which is the bias term.

Beyond this, there are three layers of sigmoid activation nodes namely the Input gate, the Forget gate and the Output gate. The value obtained in g is then multiplied by the output of the input gate, which can be expressed as:

$$i = \sigma(b_i + x_t U_i + h_{t-1} V_i)$$

Here as well, U_i, V_i, W_i and Z_i are the weights of x_t, h_{t-1}, h_{t-2}, h_{t-3} respectively. Similarly, B_i is the bias term. Thus, we see that the values from the input gate act as weights for the normalized input g.

The Forget Gate helps to learn the relationship between the various inputs separated by time. The Forget Gate generates s_t using another sigmoid activated node such that:

$$f = \sigma(b_f + x_t U_f + h_{t-1} V_f)$$

It is this delay block s_t that makes the LSTM block different from other Neural Networks as it overcomes obstacles that other models face.

The output from the previous state s_{t-1} and the forget gate are multiplied to act as weights to generate a new state which is given as:

$$s_t = (s_{t-1} \circ f) + (g \circ i)$$

Here the value from the previous state is multiplied (dot product) with the output of the forget gate which is then added to the dot product of g and i. The output gate is expressed as (where the weights here correspond to that of the output gate):

$$O = \sigma(b_o + x_t U_o + h_{t-1} V_o)$$

Thus, the final Output from the LSTM cell can be formulated as:

$$h_t = \tanh(s_t) \circ o$$

The value s_t is passed through a tanh activation function to normalize the value. This is then multiplied to the output of the output gate to get the final predicted value.

Thus, this is the design that is implemented in Simulink. Hardware acceleration techniques are put into use here to help optimize this design.

Neural networks usually require large amounts of hardware and computation time to give an output that is even remotely accurate. Our implementation here has numerical inputs. This provides multiple applications that involve predicting numbers, including but not limited to, price of a commodity (effect of inflation on general products, predicting the value of stocks, forex, etc, for any future time instant, rent in a locality), population (of a neighborhood, a bacteria mould, etc.), and many more.

Our implementation here, focuses on the application to predict stock prices and the optimization techniques are implemented accordingly. The focus here is to implement the design on ASIC targets. This proves advantageous because the key difference

between each prediction is the weights that represent the input values. This is a difference in the input data from prediction to prediction and not the hardware design itself. Neural networks require tremendous amounts of computation, both in space and in time, thus, in current scenarios, it only makes sense to use the networks in stationary circuits that have plenty resources available such as data centers. Data centers have huge amounts of hardware that dissipate massive amounts of power where the area of the circuit does not pose any limitation. However, such locations require some form of a cooling facility to ensure the durability and reliability of these circuits. Plus, applications that use time series prediction are numerous. There will always be a data set to run through such hardware, even if the circuit is only limited to the stock market, there is always going to be hundreds of stocks with values at thousands of time instants and the circuit will never run out of use.

Thus, the design target begins to take form. The purpose of this project is to increase throughput as much as possible while maintaining a power constraint. Table 1 shows the area and power numbers produced in our baseline architecture.

	Operating Frequency (MHz)	Area (um)	Dynamic Power (mW)	Leakage Power (mW)	Total Power (mW)	Throughput (Mops/sec)
Baseline arch	632.9113924	22843.8736	3.274	2.7511	6.0251	7.032348805

Table 1: Power consumption and throughput of the Baseline architecture

This implementation is observed to generate 3.274uW of Dynamic power, 2.7511uW of Leakage power which brings a total of 6.0251uW of total power consumption over an area of 22843.8736um square. The design objective here is to apply optimization techniques to maximise throughput as much as possible while making sure the power is always lesser than the 6.0251uW value that is observed in the baseline architecture.

The throughput of this design is calculated using the following formula:

$$Thrpt = \frac{freq * IF * PLL}{LL * UF * No.I}$$

Where thrpt = Throughput
Freq = Frequency at which the circuit operates
IF = Interleaving Factor
Pll = Extent of parallelism
LL = Loop Latency
UF = Unfolding Factor
NoI = Number of Iterations required to get an output

II. BASELINE ARCHITECTURE

Figure 2 shows the implementation of the baseline architecture. The fact that the outputs from the previous 3 time instants can be seen here. The input sequence is fed by the sequence block. The delay blocks at the output save the data from previous time instants to be used in the next iterations. It must be noted here that the delay in these blocks is of 9 cycles, to accommodate for the loop latency created by the entire model so as to maintain functionality of the circuit. Here, we can see that the first block gives the output g corresponding to that tanh function. It should be noted here that due to the availability of blocks, a saturation function has been used instead, which limits the value of the intermediate output between -1 and 1. The second, third and fourth block correspond to the input gate, the forget gate and the output gate respectively.

The input given here corresponds to 9 time instants, hence the output expected should be the predicted value of the 10^{th} time instant. For instance, if the input given here is 100, 200, 300, 400, 500, 600, 700, 800 and 900, but naturally the output expected here is 1000.

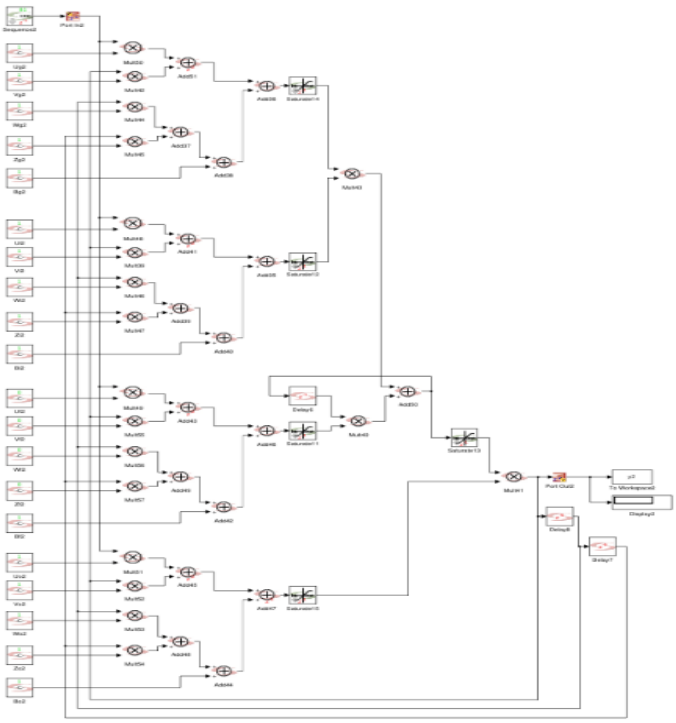

Figure 2: Baseline Architecture Implementation

Thus, the expected accuracy of the circuit is 0.01 (this is our resolution) since this circuit is to handle money. Thus, we calculate the fraction word length in the following manner:

$$W_{fr} = ceil\left(\log_2\left(1/0.001\right)\right) = 6.64 = 7$$

It must be noted here that the first block implemented is a multiplication. So, the largest output that is expected here is 10^6. After this, there are a total of 5 adders that add 1 carry bit to each intermediate output. Thus, the integer word length can be calculated as:

$$W_{int} = ceil\left(\log_2(Range)\right)$$
$$= ceil\left(\log_2(10^6)\right)$$
$$= 19.93$$
$$\approx 20 + 5$$
$$= 25$$

Thus, all the implementations further use the fixed-point notation of (32,7). Fixed point is selected over floating point as it promises higher accuracy.

III. OPTIMISATION TECHNIQUES

The best way to increase throughput is to include interleaving as one of the key optimization techniques. The baseline architecture has a loop latency of 9 cycles. Basically, it requires 9 cycles for an input to reach the output. Thus, the maximum interleaving factor that we can allow is a factor of 9 which we have used from now on.

A. Circuit Optimization

The first optimization technique implemented is circuit optimization. This is done so that reducing the operating voltage, allows room to improve throughput by increasing the power consumed (which is always the case). This includes optimization by reducing the supply voltage, implementing different gates with different threshold values (regular, low and high threshold voltages) and thus difference in sizing as well.

	Clock (ns)	Area (um)	Dynamic Power (mW)	Leakage Power (mW)	Total Power (mW)	Throughput (ops/sec)
1V	1.58	22843.8736	3.274	2.7511	6.0251	63291139.24
0.85V	2.52	23359.7609	1.24	1.3504	2.5904	39682539.68
0.75V	3.05	22693.636	0.9222	0.9328	1.855	32786885.25

Table 2: Power values from Circuit Optimization

Looking at these numbers, it is evident that by reducing voltage, the power consumption reduces considerably from 6.0251mW to 1.855mW at the cost of a reduced frequency. It is seen that the clock period has increased from 1.58ns to 3.05ns. Thus, the throughput has also reduced from 63.29Mops/s to 32.7Mops/s.

Here, an interleaving factor of 9, with 10 iterations, and a loop latency of 9 are used to calculate the throughput. The increase in clock results in the overall decrease in throughput.

It is evident here that the circuit used at 0.75V is the model that is used further for optimization.

B. Parallelism

Looking at the power consumed by the circuit operated at 0.75V (1.855mW), it only makes sense to try parallelism by replicating the circuit 3 times. Thus, Figure 3 shows the block diagram implementing parallelism.

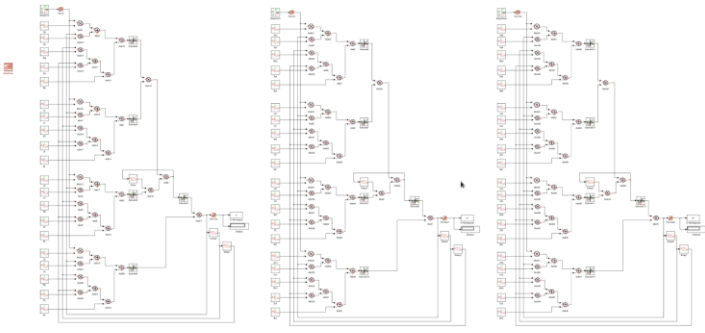

Figure 3: Implementing parallelism by replicating 3 times

Table 3 shows the power and throughput values that are thus obtained.

Here it is seen that all the numbers have tripled. While the clock frequency is still the same, the throughput is increased to 97.4Mops/s with a power consumption of 5.4856mW.

	Clock (ns)	Area (um)	Dynamic Power (mW)	Leakage Power (mW)	Total Power (mW)	Throughput (ops/sec)
Parallelism x3	3.08	53751.7102	2.7362	2.7494	5.4856	97402597.4

Table 3: Output values on implementing parallelism

The throughput is calculated using a loop latency of 9, which is same as the interleaving factor used. The output is observed after 10 cycles. Finally, with the parallelism, this is multiplied by 3.

C. Unfolding

Observing that the power consumption in parallelism is close enough to the 6mW threshold, the first implementation to experiment unfolding is with a factor of 3. Figure 4 shows the unfolding implementation and Table 4 shows the power and throughput values obtained. From this, it is seen that the clock at which the circuit operates has increased by a slight value (from 3.05ns to 3.88ns). This is to accommodate the increased wire overheard that is generated by the wires pulled from one implementation to another.

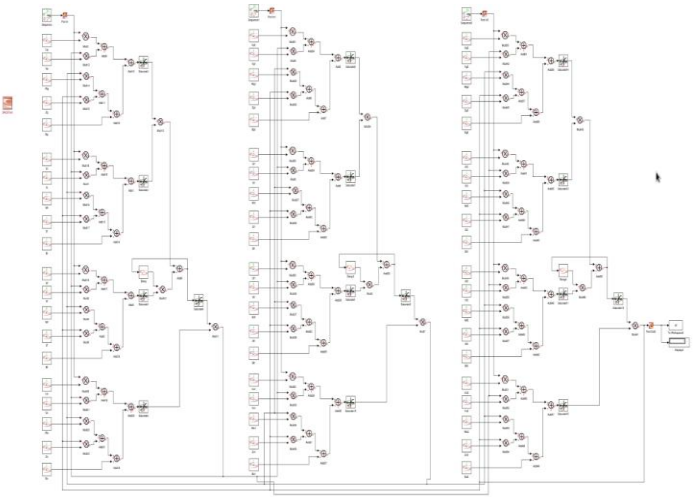

Figure 4: Implementing Unrolling with a factor of 3

	Clock (ns)	Area (um)	Dynamic Power (mW)	Leakage Power (mW)	Total Power (mW)	Throughput (ops/sec)
Unfolding x3	3.88	55068.4466	1.069	2.248	3.317	77320360.83

Table 4: Observations after implementing unrolling

It should be noted here that in the first iteration, the respective blocks do not have any inputs from past time instants that can be used to predict the output for intermediate time instants correctly. However, the LSTM network works in such a way that it can forget these redundant values at a later stage.

Here, an increase in throughput is observed from the 32.7Mops/s to 77.32Mops/s as compared to the baseline architecture operated at 0.75V.

Comparing this to the parallelism implementation, we see that unfolding has a much lesser power consumption. The leakage power is the same for both as the two effectively have 3 blocks that always need to be powered to keep the circuit running. However, in the unfolding architecture, the dynamic power

consumed is much smaller because the computations here are interdependent. Parallelism has 3 independently working blocks that thus have a higher dynamic power consumption.

The throughput here is calculated using an unfolding factor of 3, 9 as the loop latency, 3 iterations required to get the output with an interleaving factor of 27. This is now possible, because there are now 27 cycles in the feed forward path between the first input being fed and the first output being ready.

Since, this generated only 3.317mW of power, there is much more room for improvement, hence, the next implementation is using an unfolding factor of 5. Figure 5 shows the block diagram used to run the implementation and Table 5 shows the numbers thus produced.

The throughput here is calculated using an unfolding factor of 5, 9 as the loop latency, 2 iterations required to get the output with an interleaving factor of 45. This is now possible, because there are now 45 cycles in the feed forward path between the first input being fed and the first output being ready.

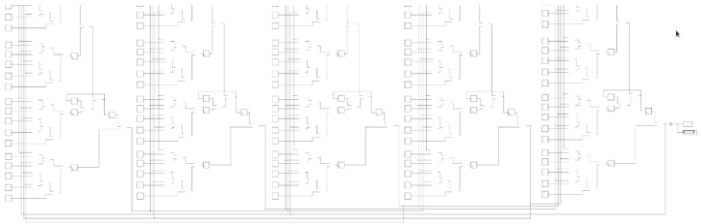

Figure 5: Block diagram implementing unfolding factor of 5

	Operating Frequency (MHz)	Area (um)	Dynamic Power (mW)	Leakage Power (mW)	Total Power (mW)	Throughput (Mops/sec)
Unfolding x5	257.7319588	94582.6921	1.7703	3.7896	5.5599	128.8659794

Table 5: Values observed implementing unfolding factor of 5

Here, it is seen that for the same clock value, the power consumption is at 5.5599mW that generates a throughput of 128.86Mops/s. This keeps the power well within limits, thus any further unfolding is moot.

D. Parallelism and Unfolding

Since the block diagram implementing an unfolding factor of 3 generated such little power, it has much scope to increase throughput.

Thus, another block diagram implemented here is using an unfolding factor of 3 with parallelism. With the power consumed in the unfolding model (3.317mW) in only makes sense to replicate the circuit only once. Figure 6 shows the block diagram implemented and Table 6 shows the values thus observed.

The throughput here is calculated using an unfolding factor of 3, 9 as the loop latency, 3 iterations required to get the output with an interleaving factor of 27. This is possible, because there are now 27 cycles in the feed forward path between the first input being fed and the first output being ready.

In this implementation, we see that at a slightly increased clock (3.94ns) drastically gives us a higher throughput of 152.28Mops/s, however, the power consumption has exceeded the threshold of 6.0251mW (it is 6.436mW now), but, it is still within a 10% range from the threshold (allowing a 10% margin allows to exceed the threshold to 6.628mW), which this design follows perfectly.

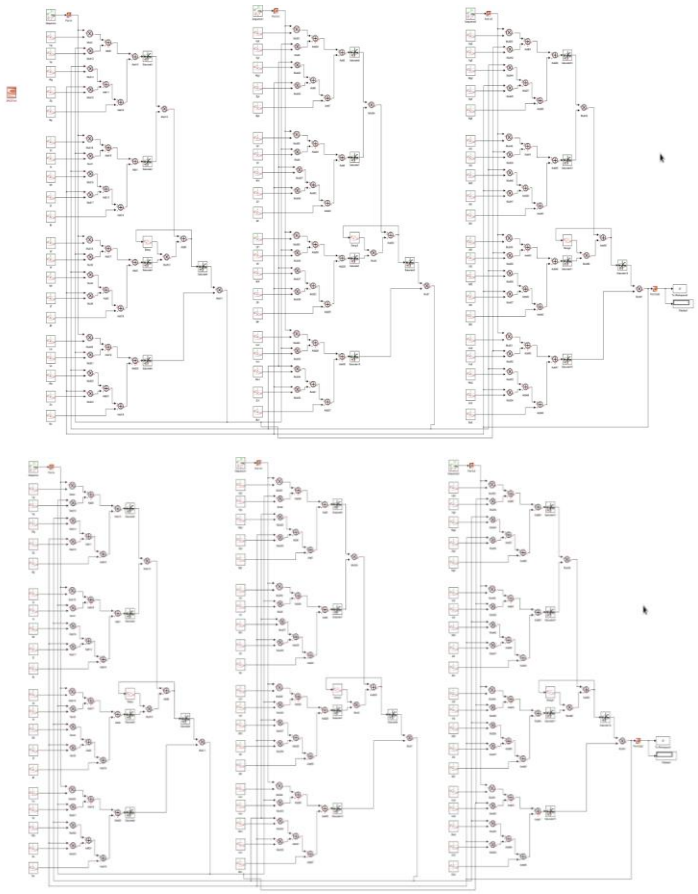

Figure 6: Implementation of Unfolding x3 and Parallelism x2

	Clock (ns)	Area (um)	Dynamic Power (mW)	Leakage Power (mW)	Total Power (mW)	Throughput (ops/sec)
Unfolding x3 and Parallelism x2	3.94	113297.1629	2.1058	4.3305	6.4363	152285786.8

Table 6: Observed values of power consumption

IV. RESULTS

Thus, Table 7 combines all the numbers for comparison.

Design Architecture	Operating Frequency (MHz)	Area (um)	Dynamic Power (mW)	Leakage Power (mW)	Total Power (mW)	Throughput (Mops/sec)	Normalised Throughput	Energy (uJoules/op)
Baseline architecture (1V)	632.9113924	22843.8736	3.274	2.7511	6.0251	7.032348805	1	856.76922
1V	632.9113924	22843.8736	3.274	2.7511	6.0251	63.29113924	9	95.19658
0.85V	396.8253968	23359.7609	1.24	1.3504	2.5904	39.68253968	5.642857143	65.27808
0.75V	327.8688525	22693.636	0.9222	0.9328	1.855	32.78688525	4.662295082	56.5775
Parallelism x3	324.6753247	53751.7102	2.7362	2.7494	5.4856	97.4025974	13.85064935	56.31882667
Unfolding x3	257.7319588	55068.4466	1.069	2.248	3.317	77.32036083	10.99495531	42.89943767
Unfolding x5	257.7319588	94582.6921	1.7703	3.7896	5.5599	128.8659794	18.32474227	43.144824
Unfolding x3 and Parallelism x2	253.8071066	113297.163	2.1058	4.3305	6.4363	152.2857868	21.65503889	42.26461402

Table 7: Final results

We can clearly see how circuit optimisation techniques reduce throughput, and power, thus creating room to make more improvements. Also, including interleaving, the throughput numbers increase drastically. With this, it is seen that the energy consumption (joules per output) also decreases gradually, thus increasing the efficiency of the optimisation techniques.

Figure 7 graphically represents the throughput values for each implementation. It must be noticed here that the only difference between the baseline architecture and the implementation represented by 1V is the Interleaving Factor. The interleaving factor improves throughput numbers drastically.

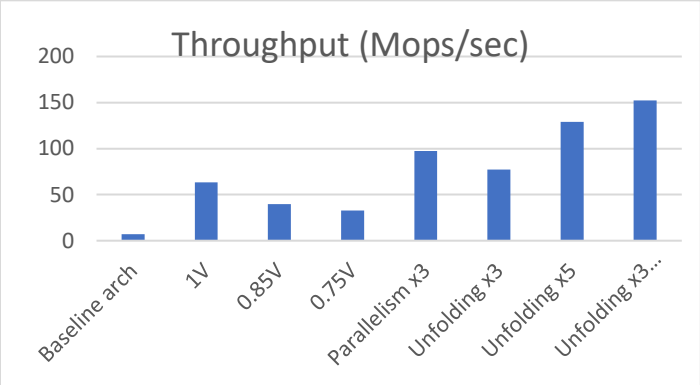

Figure 7: Comparing Throughputs

Here, it is clearly seen that the baseline architecture has a minimal throughput of 7.03Mops/s, which immediately increases 9 fold by implementing interleaving. From here, reducing the voltage values through curcuit optimisation, reduces the throughput as well. Once architectural techniques are implemented, the throughput increases many fold proportionately.

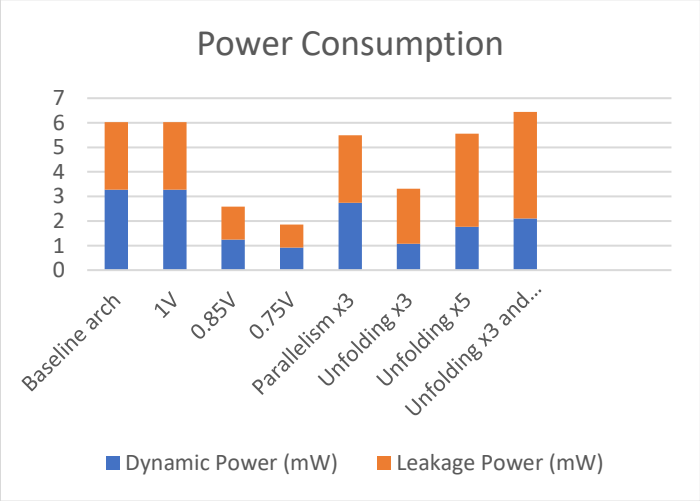

Figure 8: Power Consumption

It is clearly noted in Figure 8 that circuit optimisation reduces the power consumption drastically, which further allows scope to implement architectural techniques. It is observed here that even though the two implementations that observe parallelism and unfolding with a factor of 3 have different power consumtion values. It is seen that both these implementations have the same leakage power (because they are basically 3 blocks that require static power) but unfolding has a much lesser dynamic power consumption compared to parallelism. This is because parallelism and unfolding use the same amount of area (which decides the leakage power) while dynamic power also depends on frequency of operation which is lower for unfolding.

Figure 9 shows the operating frequency of all the designs implemented. It is seen that with each optimisation technique implemented, the frequency decreases. This is due to the reduced operating voltage and the constant increase in complexity of the circuit.

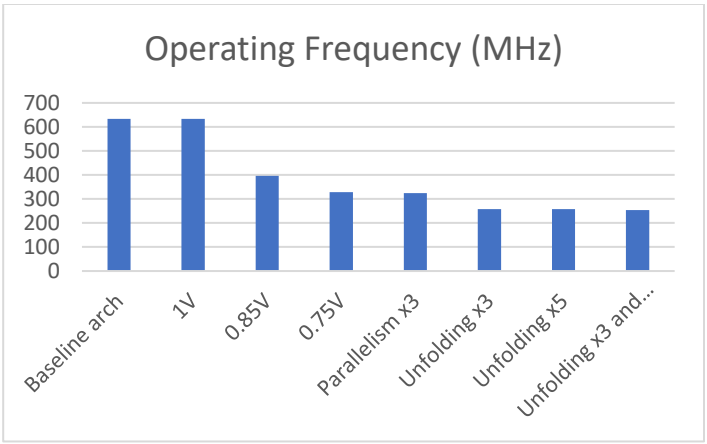

Figure 9: Comparing operating Frequency

Thus, from Figure 10 we can see that in spite of the constant decrease in frequency and attemtping to stay within 6.0251mW of power, the energy consumed per output is still constantly decreasing with each optimisation implemented.

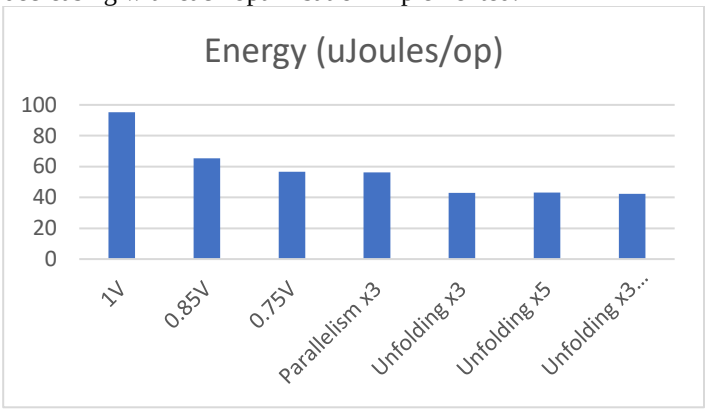

Figure 10: Energy consumption

V. CONCLUSION

Thus, we were succesfully able to implement an LSTM network in Simulink. Hardware acceleration techniques were successfully implemented to increase the throughput 21.6 times more than the baseline architecture while exceeding the power consumption only by 10% in spite of a constant decrease in frequency with each optimisation implementation . By doing so, the area increases by manifold, however, the assumption here is that since the target environment is a stationary climate controlled data center, this is allowed.

REFERENCES

[1] Martin Sundermeyer, Ralph Schluter, Hermann Ney, LSTM Neural Networks for language modelling, RWTH Aachen University http://www.isca-speech.org/archive

[2] Jason Brownlee, How to develop LSTM model for Time Series Forecasting, Deep Learning for Time Series, November 14, 2018, (https://machinelearningmastery.com)

[3] Seril Yesif, Taemin Kim, Andrey Ayupov, Hardware Accelerator design for Data Centers, IEEE, Novenber 6, 2015

[4] Professor Fengbo Ren, CIDSE, Arizona State University