Long Short-term Memory for Radio Frequency Spectral Prediction and its Real-time FPGA Implementation

Siddhartha[1], Yee Hui Lee[1], Duncan J.M. Moss[1],
Julian Faraone[1], Perry Blackmore[2], Daniel Salmond[3]
David Boland[1], Philip H.W. Leong[1]
Email: {siddhartha.siddhartha,yeehui.lee,duncan.moss,
julian.faraone, david.boland,philip.leong}@sydney.edu.au
{daniel.salmond,perry.blackmore}@dst.defence.gov.au

***Abstract*—Reactive communication waveforms hosted in current generation tactical radios often fail to achieve good performance and resilience in highly dynamic and complex environments. Arguably, novel waveforms that can proactively adapt to anticipated channel conditions may better meet the challenges of the tactical environment. This motivates the ability to accurately predict spectral behaviour in real-time. A Long Short-Term Memory (LSTM) network is a type of recurrent neural network which has been extremely successful in dealing with time-dependent signal processing problems such as speech recognition and machine translation. In this paper, we apply it to the task of spectral prediction and present a module generator for a latency-optimised Field-Programmable Gate Array (FPGA) implementation. We show that our implementation obtains superior results to other time series prediction techniques including a naïve predictor, moving average and ARIMA for the problem of radio frequency spectral prediction. For a single LSTM layer plus a fully-connected output layer with 32 inputs and 32 outputs, we demonstrate that a prediction latency of $4.3\ \mu s$ on a Xilinx XC7K410T Kintex-7 FPGA is achievable.**

I. Introduction

Over recent years, neural networks (NNs) have achieved results surpassing all other approaches on difficult pattern recognition problems such as image analysis, speech recognition and machine translation. In particular, the Long Short-Term Memory (LSTM) [13], [9] has achieved the best results of any technique for time-series applications such as handwriting recognition, speech recognition and machine translation.

Communication channels can be characterised as a sequence of observations – e.g. digital samples, bursts or short-time Fourier transform vectors – and hence, they are amenable to processing using LSTM architectures, whether to characterise, generate or predict channel behaviour. Prediction of spectral behaviour supports *proactive* adaptation in response to anticipated channel behaviour as opposed to the reactive implementations that are widely available at present. This could take the form of novel waveforms that dynamically adapt their parameters such as modulation scheme or forward error correction. We believe low-latency spectral prediction will support a family of waveforms that adapt on a burst-by-burst basis, enabling more efficient channel utilisation and more resilient communication networks.

Military communication networks operate in unanticipated and dynamic environments, and stand to benefit from proactive waveforms that adapt to the ambient and anticipated conditions. Designing waveforms to work in these environments is challenging because the variety of operational contexts is too large to be exhaustive. Applying deep learning for spectral prediction is attractive because, if trained properly, deep learning models generalize well to unseen sequence contexts.

Despite its dominance in other domains, little work in applying LSTMs to real-time communications systems has been undertaken due to the challenge of real-time processing of radio frequency signals. In this paper we describe a Field-Programmable Gate Array (FPGA) implementation of an LSTM prediction system which is energy efficient, parallel, integrated on the same chip as the software defined radio hardware, and customised to achieve high performance. Specifically the contributions are as follows:

- A flexible LSTM module generator which can produce optimised, self-verifying LSTM inference cores of arbitrary size and arbitrary fixed-point precision.
- A detailed comparison of LSTM spectral prediction accuracy compared with standard techniques, showing that LSTM provides a significant improvement in accuracy, which is crucial for resilient military communication applications.
- An empirical study of tradeoffs between fixed-point precision and accuracy.

To the best of our knowledge, this is the first reported system-level, real-time implementation of LSTMs for spectral prediction in radio frequency applications.

[1]The University of Sydney, Australia, [2]Defence Science and Technology Group, Canberra, Australia, [3]Defence Science and Technology Group, Edinburgh, Australia.

II. Background

A. Long short-term memory (LSTM)

The LSTM was introduced in 1997 by Hochreiter and Schmidhuber [13]. Compared to earlier recurrent neural networks for time-series prediction, mechanisms were introduced to protect state stored in memory cells from being corrupted by weight updates from irrelevant inputs and irrelevant memory contents. Many variants of the original LSTM have been proposed. The version used in this work is the BasicLSTMCell [8] class that can be found in Google's Tensorflow neural network package. This, in turn, is an implementation of the network published in references [9] and [19].

Using the same notation as [19], let $h_t^l \in \mathbb{R}^{n_l}$ be the output vector of layer l at timestep t, and $c_t^l \in \mathbb{R}^{n_l}$ a memory cell. As illustrated in Figure 1, an LSTM layer computes a new output and memory cell value using the state transition: $h_t^{l-1}, h_{t-1}^l, c_{t-1}^l \rightarrow h_t^l, c_t^l$ with the formulae

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T^l_{(n_{l-1}+n_l),(4n_l)} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \tag{1}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g \tag{2}$$

$$h_l^t = o \odot \tanh(c_t^l) \tag{3}$$

where $T^l_{m,n} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is an affine transformation ($Wx + b$ for some W and b) for layer l, $\odot$ is elementwise multiplication, $sigm$ is the elementwise sigmoid activation function, $\tanh$ the elementwise hyperbolic tangent activation function, x_t is the input vector, and $h_t^0 = x_t$. The LSTM output, h_t^L, where L is the number of LSTM layers is then used to predict the final output f_t via a fully connected (FC) layer

$$f_t = T^{L+1}_{n_L,n_L} h_t^L. \tag{4}$$

In order to make predictions from a fixed-length sequence, we apply a periodic context reset in which the memory cell and previous output values for all layers are cleared. This is described in detail in Sec. IV-C

B. Literature Review

LSTMs have been successfully employed in sequence prediction problems such as RF anomaly detection [15] and solar radio spectrum classification [18] but processing in real-time was not addressed.

Only a small number of FPGA implementations of LSTM networks [13] have been reported. Chang et. al. [4] in 2015 implemented

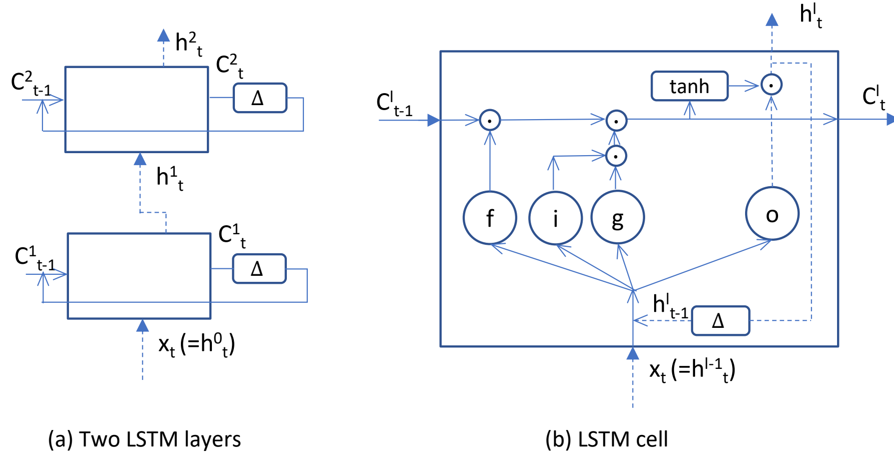

Figure 1. Illustration of the LSTM dataflow, with output given by Equ. 3. Δ represents a unit delay.

an LSTM network on a Zynq 7020 FPGA using a matrix-vector multiplier architecture. Their 2 layer, 128 hidden unit design operated at 142 MHz and was 21× faster than the ARM Cortex A9 processor on the same FPGA. In 2017 Guan et. al. [10] describe an optimised 32-bit floating point LSTM implementation which achieved 7.26 GFLOPS on a Xilinx Virtex7-485t FPGA at 150MHz. Finally, Han et. al. [11] describe a load-balance-aware pruning method to introduce sparsity and quantisation in an LSTM implementation. On a Xilinx XCKU060 FPGA running at 200MHz with 12-bit precision, their design achieved 282 GOP/s, corresponding to 2.52 TOPS on an equivalent dense network. In contrast to previous implementations, our work focuses on a comparison with conventional RF spectral prediction techniques, achieving high accuracy, and optimising latency at the system level through integration. This paper is conceptually similar to our previous work on real-time anomaly detection of RF signals [14] which focused on anomaly detection rather than spectral prediction, and used an autoencoder instead of an LSTM.

III. LSTM-based Spectral Prediction

A. RF Spectral Prediction

Spectral prediction, also known as spectral inference, aims to infer the state of unknown radio spectrum based on previous/known spectrum statistics. In recent years, spectral prediction has gained popularity in wide range of cognitive radio networks applications such as dynamic spectrum access, smart topology control, adaptive spectrum sensing, and predictive spectrum mobility. In practical cognitive radio implementations, prediction accuracy, low latency, and low energy consumption have been the major technical concerns [7]. As the core of many cognitive radio networks applications, the need for high speed spectral prediction with low power consumption is imminent.

B. 802.11p Dataset

The data set used in this study was generated via "over-the-air" captures of IEEE 802.11p frames.[1] An 802.11p framework described in [1] was used and deployed on the GNU Radio software toolkit interfaced to an Ettus Research™ Universal Software Radio Peripheral (USRP™) X310 with SBX RF daughterboard.

Frames encapsulating 500 bytes were generated, comprising of a mix of QAM-64 and BPSK modulation types. Three captures were recorded: a 112 μs QAM-64 frame, a 498 μs BPSK frame, and a 10 μs noise floor sample. The digital storage oscilloscope that was tapped into the network autosampled at 20 GSa/s, 10 GSa/s and 5 GSa/s respectively.

Post-processing was performed, first creating frequency-shifted (5MHz) variants of the frame captures. All sequences were then downsampled before concatenation to create a 10 MHz dual band signal. The first band contained six bursts, while the second band comprised of three bursts of varying duration. The data was stored as a series of 8-byte complex IEEE-754 floating point entries baseband I/Q samples. The signal could then be continuously replayed with a period of 1857 μs. We denote the data set as $D = \{x_i\}, i \in [0, T)$, where T is the size of the data set.

[1]The 802.11p standard was chosen in preference to a/b/g/n standards due to its lower spectral bandwidth (5 MHz) Transmissions were constrained to a co-axial medium to avoid interference with other networks.

C. Number Representation

In order to minimise latency, the lowest precision should be used. Unfortunately, low precisions adversely affect prediction accuracy. To best navigate this tradeoff, processing can be performed in a variable precision fixed-point format that is determined at compile time.

A n-bit fixed-point representation of a number $x = x_{INT} + x_{FR}$ consists of integer and fraction components using I and F bits respectively, where $n = I + F$. I determines the range of numbers that can be represented and F controls the precision, Equ. 5 shows the notation in a binary form. We represent the integer and fractional sizes using the notation $QI.F$.

$$X = (X_{(I-1)} \dots X_1 X_0 . X_{-1} \dots X_{-F}) \tag{5}$$

where $X_i \in \{0, 1\}$ represents the i'th bit, $i \in [-F, I-1]$. In the two's complement system used in our work, $x_{INT} = -X_{I-1}2^{I-1} + \sum_0^{I-2} X_i 2^i$ and $x_{FR} = \sum_{-F}^{-1} X_i 2^i$. It follows that these numbers represent:

$$x = x_{INT} + x_{FR} = -X_{I-1}2^{I-1} + \sum_{-F}^{I-2} X_i 2^i \tag{6}$$

In our implementation, the `ap_fixed` type available in Xilinx Vivado HLS [17] was used with convergent rounding (`AP_TRN`) and saturating arithmetic (`AP_SAT`).

D. Training

The spectral prediction problem is to make a prediction of the future spectrum vector from previous values. This is achieved by

using a sliding-window of the previous W samples, to make a prediction N timesteps into the future. A training set of exemplars, E, is formed from a subset of D, $\{\{x_j, x_{j+W}\}, \{x_{j+N}, x_{j+W+N}\}\}$, where the first vector is the input and the second the target.

Tensorflow was used to train the LSTM and the fully-connected output layer, and the mean-squared error (MSE) is used as the cost function. A batch size of 1, no dropout and randomized selection of batches from the data set were used.

Since the FPGA implementation uses fixed-point numbers, quantization must be modeled during training. We generalised the scheme developed by Courbariaux et. al. [6] for binarized weights to low-precision fixed point numbers. Training proceeds using the standard double-precision floating-point Adam optimiser, a stochastic gradient-descent based technique. A training iteration for a neural network normally consists of three stages: forward propagation (or inference), backward propagation and parameter update. Weights and biases (hereafter referred to as weights) are first clipped to the $[-1, 1]$ range and then quantized by a function that rounds to the nearest representable fixed-point value, as shown in the equation below

$$r(x) = \lfloor x * 2^F + 0.5 \rfloor)/2^F \tag{7}$$

This is done only during the forward and backward propagation steps so that small weight changes can be accumulated and applied during parameter update.

The output of the training process are the weights for the T transformation of Equ. 1, and the weights of the fully-connected layer.

E. Prediction Accuracy with Fixed-point Weight Values

A comparison of prediction performance of the LSTM model (trained with full precision weights) against classical autoregressive and averaging time series prediction models was undertaken. The techniques used for comparison were: (1) Naïve, where the predicted vector is the RF spectrum observed in the last timestep, (2) Moving average, where the predicted vector is the average of RF spectrum (respective FFT bins) within the window length, and (3) Autoregressive integrated moving average: an ARIMA(5, 1, 1) model, where the parameters were selected based on trials using different configurations.

We note that the above-mentioned classical techniques can be implemented with a single cycle of latency, unlike the LSTM, which requires L cycles. Thus for an LSTM prediction to be produced at time t, only data up to x_{t-L} can be used. We assume for the other techniques, data up to x_{t-1} are available. Our study reflects this important detail, ensuring a fair comparison.

Fig. 2 compares a floating-point implementation of the LSTM (single and double precision results are similar) to the other techniques. It shows that LSTM outperforms the baseline models in most cases, especially for predictions further into the future i.e. large N. It is important to note that the key feature of a robust RF spectral predictor is to achieve high accuracy prediction far into the future as it provides adequate response time for required actions. Hence, the proposed LSTM approach for RF spectral prediction gives an important improvement in accuracy, this being especially important for resilient communication applications.

To facilitate efficient FPGA implementation, the minimal precision weight and bias values are desirable. Figure 3 compares MSE of our implementation at different fixed-point weight and bias wordlengths ($Q1.F$) with range [-1,1). It can be seen that wordlengths of 8-bits and above show similar accuracy to floating-point.

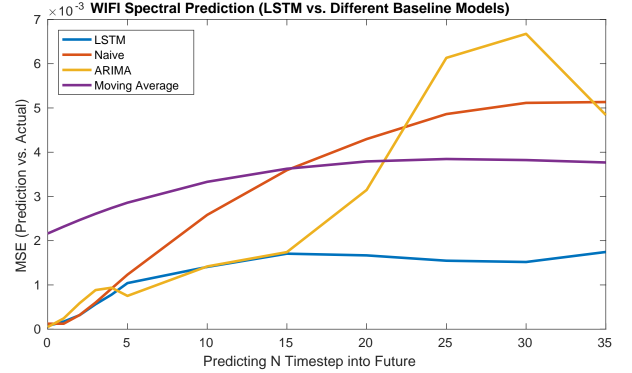

Figure 2. Comparison of mean-squared error (MSE) between floating-point LSTM and different baseline models for WIFI spectral prediction.

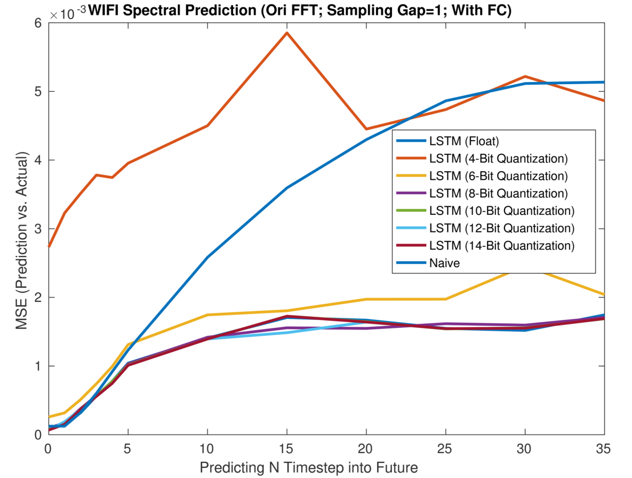

Figure 3. Comparison of LSTM mean-squared error (MSE) with different fixed-point bias and weight wordlengths.

IV. Implementation

Our system implementation is illustrated in Fig. 4 and consists of custom software and hardware blocks designed to deliver an end-to-end software defined radio solution for spectral prediction. Design details are abstracted away from the user through a Python-based software framework. This allows for rapid prototyping and easy modification of the underlying hardware core and resource utilisation exploration.

The system implementation is built on top of the GNU Radio [2] and RFNoC [3] environments, allowing arbitrary components to be incorporated and easy reuse of our design. Additionally, the software framework provides realtime displays and monitoring features, such as prediction error histograms and spectral output.

Although our software is capable of generating designs of arbitrary size and precision, we focus on an implementation with one LSTM and one Fully-Connected (FC) layer, which was found to produce good results. The LSTM has 32 inputs, 32 memory cells and 32-outputs. The LSTM context length, discussed in Sec. IV-C is also 32. The FC layer has 32 inputs and 32 outputs. This choice was made due to hardware resource constraints as a significantly larger implementation was not possible using the chosen hardware platform.

A. GNU Radio

The host is responsible for programming the Ettus board and orchestrating all data movements to and from the platform at runtime. This is achieved through a high-level interface in GNU Radio. We design custom GNU Radio blocks that communicate with the prediction core on the FPGA, while computing and displaying live prediction errors at runtime. The GNU Radio flowgraph is fully

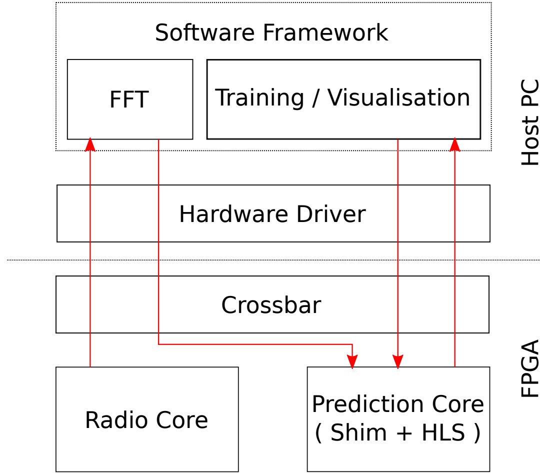

Figure 4. System overview, the shim forms the interface between the HLS code and the AXI bus.

parametric, and we can change parameters such as sampling rate, input/output vector length, prediction offset, etc at runtime.

A separate GNU Radio flowgraph is also designed to load new trained weights at runtime. Currently, these are streamed to the FPGA at low frequency to avoid data corruption. The time to reprogram the FPGA with new weights is in the order of a few minutes, which is still significantly faster than a fresh RTL-to-bitstream synthesis spin which, depending on precision, can take about 5 hours. Nevertheless, future iterations of the design would improve upon this limitation to deliver fast, real-time reprogrammability of the LSTM cell in the order of milliseconds.

B. Module Generator

A Python class, `lstmgen`, is used to generate the Prediction Core (see Fig. 5). Object creation requires the desired LSTM and FC parameters, precision, quantization function, a tensor of trained weights from Tensorflow and a testing set. A method is then called to produce a standalone C program, within which the weights, testing and expected output data are embedded. The C program can be compiled and executed directly under the Linux operating system, and/or synthesised to an FPGA implementation. The C program is self-verifying so that programming errors can be quickly identified. This is possible because the test set inputs and expected outputs are compiled with the program.

In the case of the FPGA implementation, the same C program is synthesised to an FPGA design via a high-level synthesis (HLS) tool. The resulting LSTM Cell and FC layer are then integrated with first-in-first-out (FIFO) buffers and wrapped within an AXI streaming interface as illustrated in Fig. 5. The host PC can execute the verification C code to check the correctness of the FPGA implementation through this interface. As depicted in Fig. 5, our implementation includes an application programmer interface (API) for real-time updating of the LSTM and FC weights without recompilation.

C. Prediction Core

The prediction core and the crossbar are connected by an AXI-Stream Interface, as shown in Fig. 5. The data is sent sequentially to the prediction core.

Before the data from the host is transferred to the FPGA prediction core, the $QI.F$ values are padded in the least-significant bits and represented as a 16-bit short. This abstracts away the $QI.F$ representation from the Crossbar-Prediction Core interface allowing for a single implementation to handle all configurations of $QI.F$ for $I + F \leq 16$. Conversion between $QI.F$ and the 16-bit short format is handled by the prediction core shim and software framework.

At the heart of the LSTM and FC is a matrix-vector multiply and a vector-vector addition, which are required to implement the $T_{m,n}$ transformation in Equ. 1. This is a parallel implementation using independent multipliers and $2F$ precision accumulators to avoid loss of accuracy. The $sigm$ and $tanh$ activation functions are implemented using a complete lookup table of precomputed values. The size of these lookup tables grow exponentially with the number of bits used for precision.

An input vector of length 32 is consumed by the LSTM Cell in 32 cycles after which the local context vector is updated, and a resulting 32-element vector is produced at the output of the LSTM cell. The context length is used so the LSTM can consider a time-series of input vectors and is set to 32 i.e. prediction based on past 32 input vectors. Only the 32nd output result vector is buffered and sent to the FC layer. A context counter module orchestrates this data movement and feeds the context back into the LSTM Cell. At timestep 1024 (= input length $\times$ context length) the output of the LSTM is sent to the FC layer where the final prediction is computed. After this prediction has been produced, the context vector for the LSTM is reset to zero and the process is repeated.

Our implementation allows the LSTM and FC weights to be updated during operation through a memory mapped register interface. This means the FPGA can continuously perform full-speed inference, with simultaneous training occurring on the host processor at a lower speed. Periodic updates of the weights are streamed to the FPGA.

D. FFT

The FFT block takes the raw input from the radio core and pre-processes the data in preparation for the LSTM prediction core. The FFT uses the standard Cooley-Tukey [5] decimation-in-(time/frequency) FFT algorithm with a window size of 16, producing 16 complex values. These complex values are split into their respective real and imaginary components in floating-point precision and subsequently transformed into their $QI.F$ fixed-point representation. This process creates two 16-element fixed-point vectors that contain the real and imaginary parts of the FFT output window, which are concatenated to form the 32-element fixed-point input vector.

V. Results

We target the Ettus USRP X310 software defined radio (SDR) platform with an RF front end that supports DC to 6 GHz bandwidth. The radio has bandwidth of 160 MSa/s per channel and uses a Xilinx XC7K410T Kintex-7 FPGA (specifically the xc7k410tffg900-2 device). The spectral prediction system is synthesised to Verilog register transfer language (RTL) using the Xilinx Vivado HLS tool (v2015.4) and then passed through Xilinx Vivado Design Suite (v2015.4) to produce a configuration bitstream for the FPGA. A Dell Latitude Core i5 laptop, connected to the Ettus board via 100BASE-TX fast Ethernet, is used as the host PC. In a practical low-latency systems implementation, the Prediction Core in Fig. 4 would be directly connected to the Radio Core to minimise latency.

A. Prediction Core Performance

We implement the matrix-vector multiply in the LSTM cell using the matrix-matrix multiplication IP core provided in the Vivado HLS linear algebra library [17]. This core offers different modes to invoke different tradeoffs on the latency and resource utilization design space. Since our goal is a low-latency prediction core, we use the matrix multiplication IP core in $\texttt{ARCH} = 4$ mode.

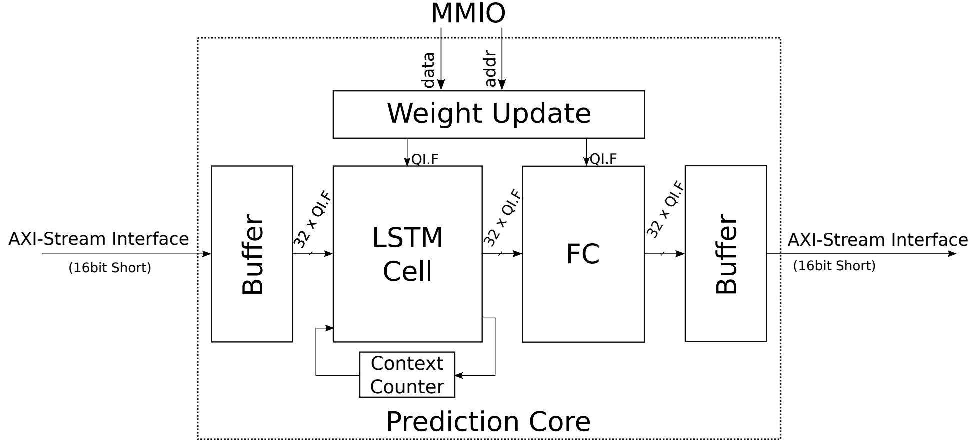

Figure 5. Block diagram of the FPGA Prediction Core

n_l	Clk ns	Latency cycles	BRAM (%)	DSP (%)	LUTs (%)
8	7.67	195	27 (1)	83 (5)	6698 (2)
16	6.43	353	27 (1)	291 (18)	20035 (7)
32	6.43	664	91 (5)	1091 (70)	68924 (27)
64	6.43	1528	154 (9)	195 (12)	18266 (7)

Table I
SYNTHESIS RESULTS FOR THE PREDICTION CORE WHICH INCLUDES A SINGLE LSTM LAYER PLUS FC LAYER. THE LSTM/LC INPUT AND OUTPUT VECTORS ARE OF LENGTH n_1. FPGA UTILIZATION IS GIVEN IN PARENTHESES.

Number System	Clk ns	Latency cycles	BRAM (%)	DSP (%)	LUTs (%)
Q2.8	8.19	636	68 (4)	1091 (70)	67285 (26)
Q2.10	6.43	663	72 (4)	1091 (70)	60710 (23)
Q2.12	6.43	664	91 (5)	1091 (70)	68924 (27)
Q2.14	6.43	664	182 (11)	1091 (70)	77143 (30)
float	8.18	1635	132 (8)	413 (26)	82568 (32)
double	7.77	1921	264 (16)	1070 (69)	156534 (61)

Table II
SYNTHESIS RESULTS FOR USING Q2.F PRECISION WHERE F IS VARIED. FPGA UTILIZATION IS GIVEN IN PARENTHESES.

Tab. I summarizes the Vivado HLS synthesis results for the Prediction Core, obtained by varying the LSTM and FC layer input/output size n_1. The target clock period was 5 ns, and a Q2.12 representation used. Since matrix multiplication scales as $O(n_1^3)$, we would expect our design to be linear in latency and quadratic in DSPs. For 32 inputs, almost all DSPs are used, and LUTs are used for multipliers. For $n_1 = 64$ and above, latency is further sacrificed to control resource utilization.

Tab. II shows resource utilization of the Prediction Core with varying fixed-point fraction length F for fixed-point types, and also includes floating-point types. In this table, the Prediction Core input/output vector lengths are fixed to 32. For fixed-point numbers, the integer part of the QI.F representation is always 2 as we found experimentally that this was sufficient to give good results. In our design, the BRAM and lookup table (LUT) requirements scale exponentially with F. This is due to the brute-force lookup table used to implement the $tanh$ and $sigm$ activation functions and could be improved using more scalable techniques, e.g. the NnCore function generator [12]. The floating point implementations have substantially increased latency and resource utilization compared with the fixed-point implementations, but achieve better accuracy.

Fig. 6 shows the MSE using the 802.11p data using different

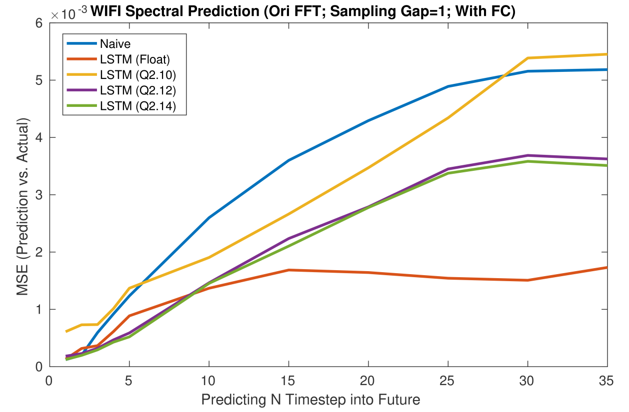

Figure 6. Comparison of LSTM accuracy when all weights and inference calculations are quantized to different precision

number systems. It shows that even at low precision, the LSTM can deliver more accurate predictions than the naïve predictor, and that beyond Q2.12, no further improvement occurs. We believe the reason the fixed-point results do not achieve the same accuracy as floating-point is due to a mismatch between the floating-point Tensorflow training forward pass and the fixed-point FPGA inference, which can be resolved by training with a fixed-point forward pass to allow weight values to compensate for this difference. We are in the process of investigating this issue further.

B. System Performance

A design integrating the Prediction Core with the Ettus RFNoC interface was made, with target clock rate of 166 MHz. Fig. 7 and Tab. III show the resource breakdown of the resulting physical implementation, in which we maximize the use of on-chip DSP blocks to deliver a fast, low-latency design that is suitable for proactive military systems.

We also implement the naïve predictor in our GNU Radio flowgraph as a baseline to compare the performance of our prediction core. Figure Fig. 8 is a density plot of errors of the naïve vs. the LSTM approach. In this experiment, the prediction offset is set to four, i.e. the prediction goal is to predict four timesteps into the future. It can be seen that the quality of LSTM predictions in terms of MSE is superior to the naïve predictor.

Trong et. al [16] measured latency values on an older Ettus USRP-2 platform, reporting times well in excess of 1 ms. Our system

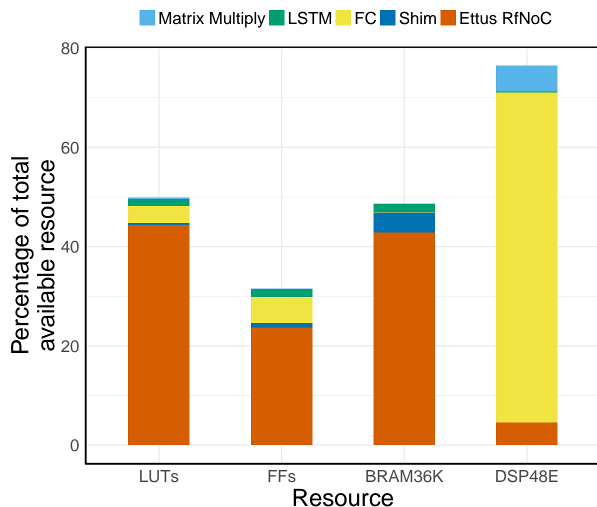

Figure 7. Post place-and-route resource utilization breakdown in percentage of total available resources using Q2.12 number representation

	BRAMs (%)	DSPs (%)	LUTs (%)
Prediction Core	46 (6)	1107 (72)	13889 (5)
Ettus RFNoC	341 (43)	71 (5)	112855 (44)
System Implementation	387 (49)	1178 (77)	126744 (50)
XC7K410T FPGA	795 (100)	1540 (100)	254200 (100)

Table III
RESOURCE UTILIZATION BREAKDOWN OF THE PREDICTION CORE, THE ETTUS RFNOC, AND TOTAL FPGA AVAILABLE RESOURCES

architecture allows predictions to occur with several microseconds of latency, this being at least 2 orders of magnitude lower.

VI. CONCLUSION

We have demonstrated, using captured IEEE 802.11p data, that low-precision LSTM-based machine learning can achieve superior spectral prediction accuracy over classical time series prediction techniques such as ARIMA. This is the case even considering that a real-time implementation of the former necessitates making predictions from less recent windows of the time series, due to computational latency. Next, we described an LSTM module generator which can generate self-validating C programs of arbitrary size, topology and precision, which can be executed on conventional processors and synthesised to FPGA designs. Finally, we described a system level implementation of a spectral prediction engine using the LSTM modules.

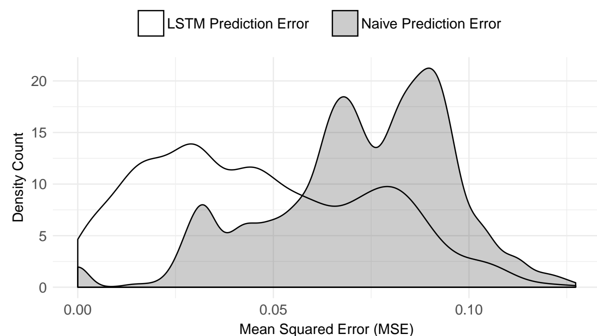

Figure 8. Density plot showing differences in prediction error between naïve predictor and the LSTM Cell using Q2.12 number representation

This work has demonstrated the feasibility of real-time prediction of spectral behaviour, which is enabling technology for proactive systems that are more resilient to interference or jamming. Our ML operations can be executed with a latency of 4.3 μs, offering the potential for actions to be taken with sub-10 μs response times. Future research will focus on further reducing latency, improving scalability, and incorporating on-chip, high-speed parameter updates to allow fast model adaption to changing conditions.

ACKNOWLEDGMENT

The authors gratefully acknowledge support from the Defence Science and Technology (DST) Group's Next Generation Technology Fund under the High Speed Machine Learning using FPGAs project.

REFERENCES

[1] Bastian Bloessl, Michele Segata, Christoph Sommer, and Falko Dressler. Towards an Open Source IEEE 802.11p Stack: A Full SDR-based Transceiver in GNURadio. In *5th IEEE Vehicular Networking Conference (VNC 2013)*, pages 143–149, Boston, MA, December 2013. IEEE.

[2] Eric Blossom. GNU Radio: tools for exploring the radio frequency spectrum. *Linux journal*, 2004(122):4, 2004.

[3] Braun, Martin and Pendlum, Jonathan and Ettus, Matt. RFNoC: RF network-on-chip. In *Proceedings of the GNU Radio Conference*, volume 1, 2016.

[4] Andre Xian Ming Chang, Berin Martini, and Eugenio Culurciello. Recurrent neural networks hardware implementation on FPGA. *CoRR*, abs/1511.05552, 2015.

[5] J.W. Cooley and J.W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. 19:297–301, 01 1965.

[6] Matthieu Courbariaux and Yoshua Bengio. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[7] Guoru Ding, Yutao Jiao, Jinlong Wang, Yulong Zou, Qihui Wu, Yu-Dong Yao, and Lajos Hanzo. Spectrum Inference in Cognitive Radio Networks: Algorithms and applications. *IEEE Communications Surveys & Tutorials*, 2017.

[8] Google. Tensorflow source code. https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/ops/rnn.py.

[9] A. Graves, A. r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

[10] Yijin Guan, Zhihang Yuan, Guangyu Sun, and Jason Cong. FPGA-based accelerator for long short-term memory recurrent neural networks. In *22nd Asia and South Pacific Design Automation Conference, ASP-DAC 2017, Chiba, Japan, January 16-19, 2017*, pages 629–634. IEEE, 2017.

[11] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, Huazhong Yang, and William (Bill) J. Dally. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pages 75–84, New York, NY, USA, 2017. ACM.

[12] S. M. H. Ho and H. K. H. So. NnCore: A parameterized non-linear function generator for machine learning applications in FPGAs. In *2017 International Conference on Field Programmable Technology (ICFPT)*, pages 160–167, Dec 2017.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[14] Duncan Moss, David Boland, Peyam Pourbeik, and Philip H.W. Leong. Real-time FPGA-based anomaly detection for radio frequency signals. In *IEEE Int. Symp. on Circuit and Systems (ISCAS)*, pages 1–5, May 2018.

[15] Timothy J. O'Shea, T. Charles Clancy, and Robert W. McGwier. Recurrent neural radio anomaly detection. *CoRR*, abs/1611.00301, 2016.

[16] N. B. Truong, Y. J. Suh, and C. Yu. Latency Analysis in GNU Radio/USRP-Based Software Radio Platforms. In *IEEE Military Communications Conference*, pages 305–310, 2013.

[17] Xilinx. UG902 Vivado Design Suite Tutorial. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug902-vivado-high-level-synthesis.pdf.

[18] Xuexin Yu, L. Xu, L. Ma, Z. Chen, and Y. Yan. Solar radio spectrum classification with LSTM. In *2017 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 519–524, July 2017.

[19] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *CoRR*, abs/1409.2329, 2014.