Study of Data Numerics and Compression for Efficient NLP Hardware Acceleration with an Emphasis on Posit

Eric Mibuari, Daniel Yang, Thierry Tambe, Zishen Wan

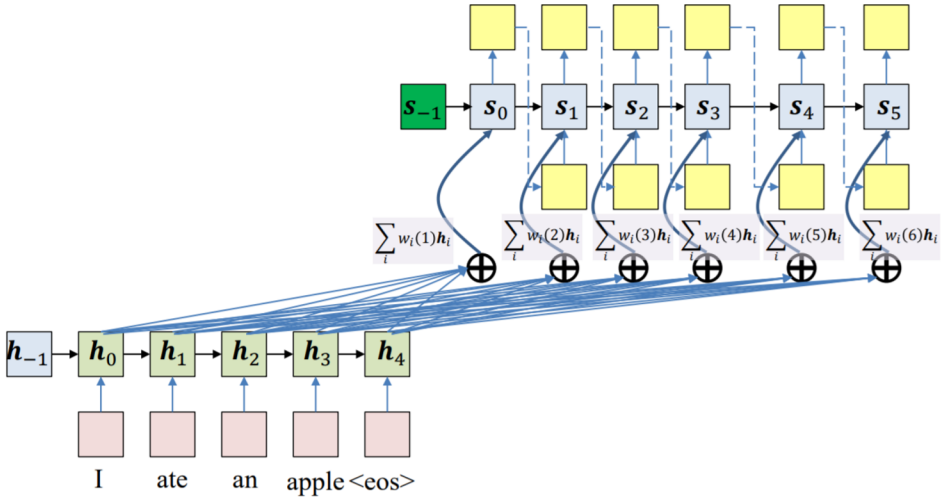
Fall 2018 CS247r Project

HARVARD
John A. Paulson
School of Engineering
and Applied Sciences

Outline

- Description of sequence-to-sequence architecture for NLP
- Overview of various bit numerics including Posit
- Result of various bit numerics on performance of speech-to-text deep learning inference
- Impact of sparsity on performance of speech-to-text inference
- Prototype of NLP accelerator engine
- Area and Power cost estimation of Posit-based Hardware

Attention-based Seq2Seq NLP model



[1]

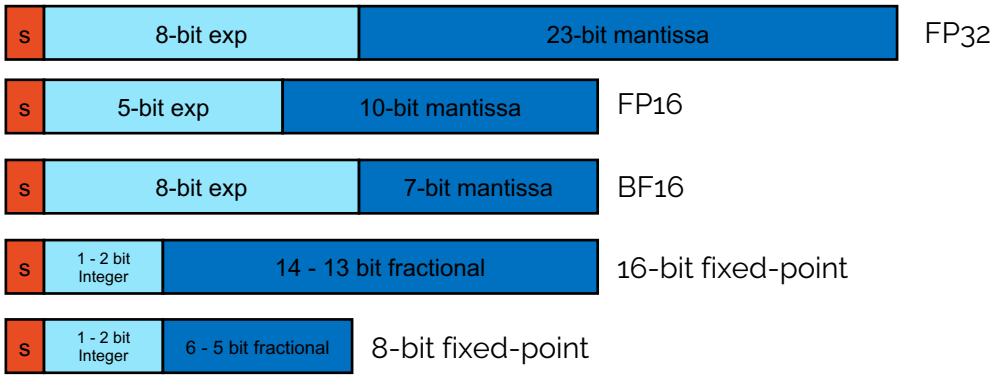
- Attention mechanism computes a weighted combination of all the encoder outputs into a context vector which is consumed by the decoder
- Model can be used for speech recognition and machine translation
- We trained on the Librispeech dataset:
  - A bidirectional GRU seq2seq model with general[2] attention to a WER of 18.8
  - A unidirectional LSTM seq2seq model with MLP[2] attention to a WER of 26.28
  - Both models will be used to evaluate quantization performance of various bit numerics

[1] Chan, et al., Listen, Attend and Spell
[2] Luong, et al., Effective Approaches to Attention-based Neural Machine Translation

Numerical Data Types

The bit numerics we considered are:

- IEEE754 FP32
- IEEE754 FP16
- BF16
- 16-bit fixed-point
- 16-bit posit
- 8-bit fixed-point
- 8-bit posit

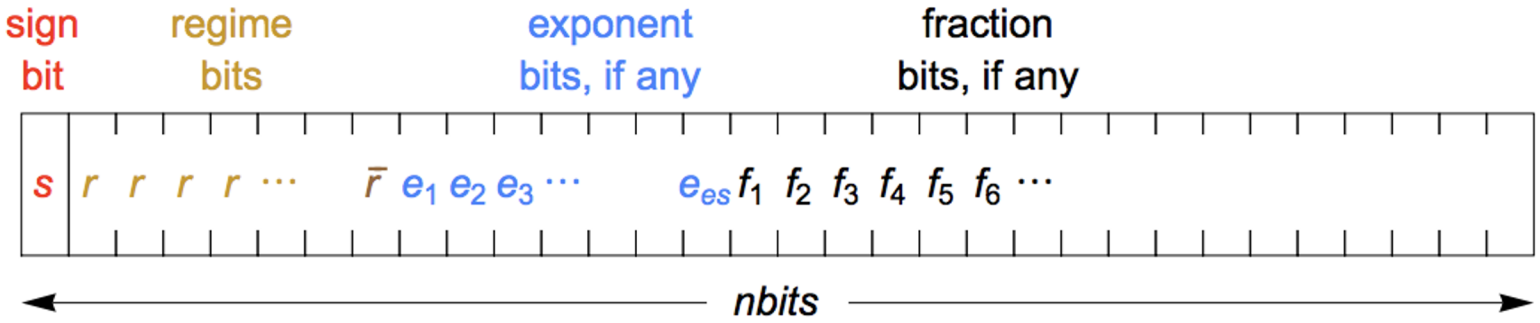


Posit Numeric: Why Posit?

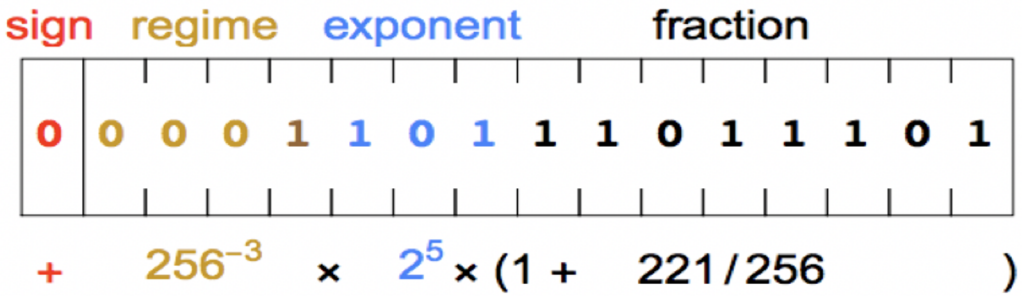
Problems with Floats:

1. IEEE 754 not a standard, but a set of recommendations
1. Overflows and underflows (to zero and infinity)
1. Exponents are too large – not adjustable
1. Accuracy is flat across a vast range, then falls off
1. Wasted bit patterns e.g., “negative zero”, too many NAN values
1. Denormalized/Subnormal numbers are headache
1. Decimal float are expensive; no 32-bit version (just 64 & 128)

Posit Format



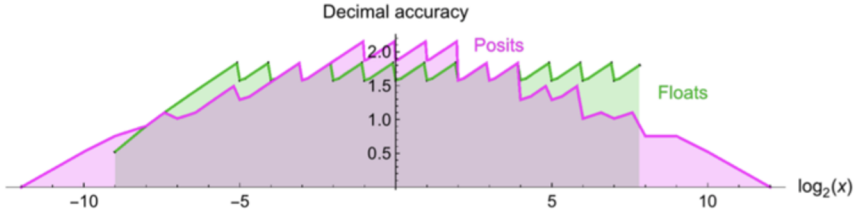
16-bit Posit Example



- Sign bit: 0 positive, 1 negative
- Regime bits: Three 0's terminated by opposite 1. -> **Useed** power = -3
- Exponent bits: 101 = 5. Unsigned binary integer
- Fraction bits: 11011101 = 221. Unsigned binary integer. Converted to fraction using Useed. -> 1 + 221/256

Evaluation: + 256^(-3) * 2^5 * (1 + 221/256) = **3.55393 * 10^(-6)**

Posit Quantization



Advantages

- Use of the same type of low-level circuit constructs that IEEE 754 floats use
- Regime bits automatically and economically create tapered accuracy, where values with small exponents have more accuracy and very large or very small numbers have less accuracy
- Less chip area than floats
- Reduced latency compared to floats of the same precision

Impact of various numerics on speech-to-text inference

Bi-directional GRU, 4-layers encoder, 1-layer decoder, General attention, 20M parameters, weights between -2.5 and 2.5	WER
Native (IEEE754 FP32)	18.80
8-bit fixed-point<3,5>	37.05
8-bit fixed-point<2,6>	22.33
8-bit posit<8,0>	**19.10**
IEEE754 FP16	18.80
Bfloat16	18.97
16-bit fixed-point<3,13>	18.78
16-bit fixed-point<2,14>	18.80
16-bit posit<16,1>	**18.80**

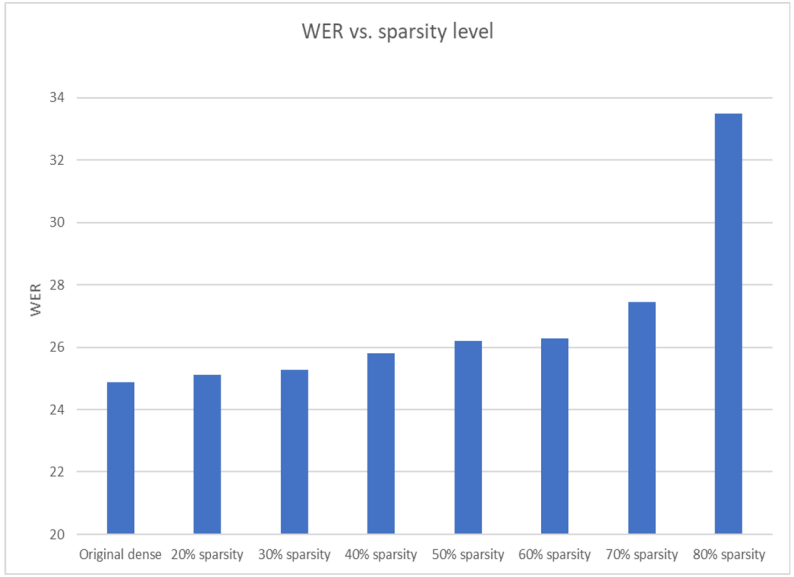
Uni-directional LSTM, 5-layers encoder, 2-layers decoder, MLP attention, 30M parameters, weights between -2 and 2	WER
Native (IEEE754 FP32)	26.28
8-bit fixed-point<2,6>	27.25
8-bit posit<8,0>	**26.28**
IEEE754 FP16	26.29
Bfloat16	26.36
16-bit fixed-point <2,14>	26.27
16-bit posit<16,1>	**26.28**

Observations

- For the most aggressive quantization, 8-bit Posit clearly outperforms 8-bit fixed-point and is comparable in performance to FP32
  - 8-bit posit for NLP deep learning seems promising!
- All 16-bits numerics (FP16, BF16, 16-bit posit, 16-bit fixed-point) have very similar performance

Impact of Sparsity

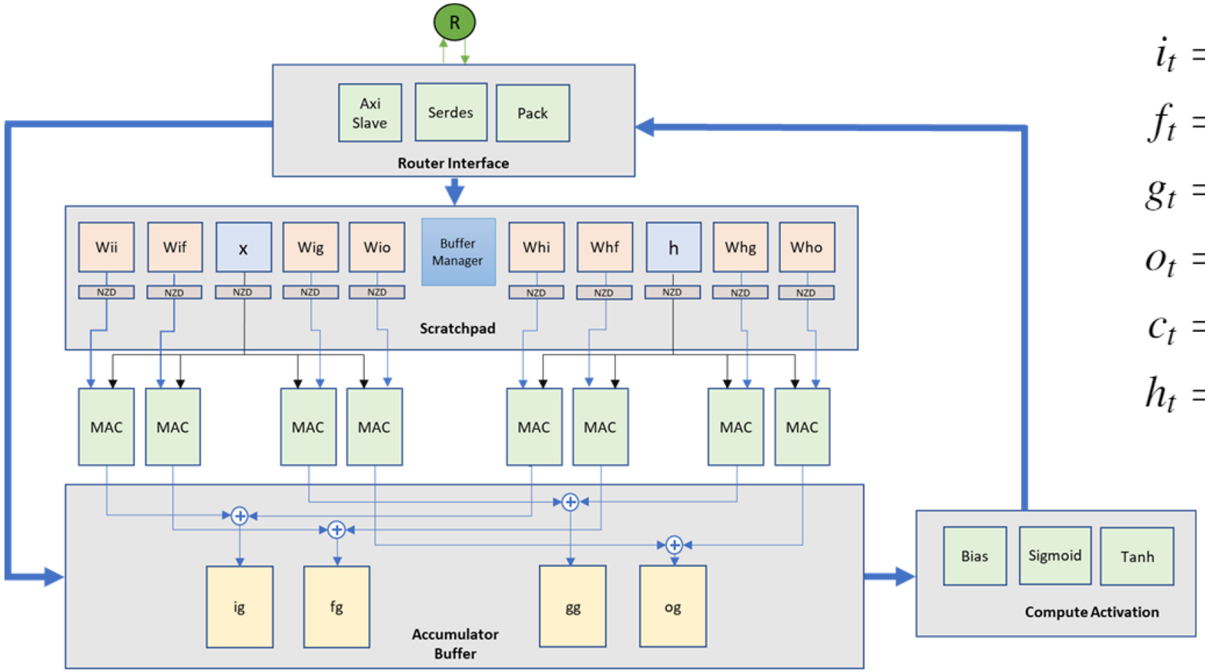
Evaluated on uni-directional LSTM seq2seq model with MLP attention using threshold pruning method



Observations

- Performance starts to conspicuously degrades at 70% sparsity
- It seems the seq2seq model may not be as robust as other FC, CNN models which can be pruned up to 90+% sparsity with negligible loss in accuracy

Prototype NLP Accelerator Engine



$$
\begin{aligned}
i_t &= \sigma\left(W_{ii}x_t + W_{hi}h_{t-1}\right) \\
f_t &= \sigma\left(W_{if}x_t + W_{hf}h_{t-1}\right) \\
g_t &= \tanh\left(W_{ig}x_t + W_{hg}h_{t-1}\right) \\
o_t &= \sigma\left(W_{io}x_t + W_{ho}h_{t-1}\right) \\
c_t &= \sigma\left(f_t * c_{t-1} + i_t * g_t\right) \\
h_t &= \tanh(c_t) * o_t
\end{aligned}
$$

Challenges

- Large I/O vector size, e.g. 800
- Store 8 weight matrices, e.g. 800*800*8*1 Byte ≅ 5MB
- Sigmoid and tanh hardware

Sigmoid and Hyperbolic Tangent

- We use piecewise linear approximations in SystemC library to implement Sigmoid and Tanh activation functions

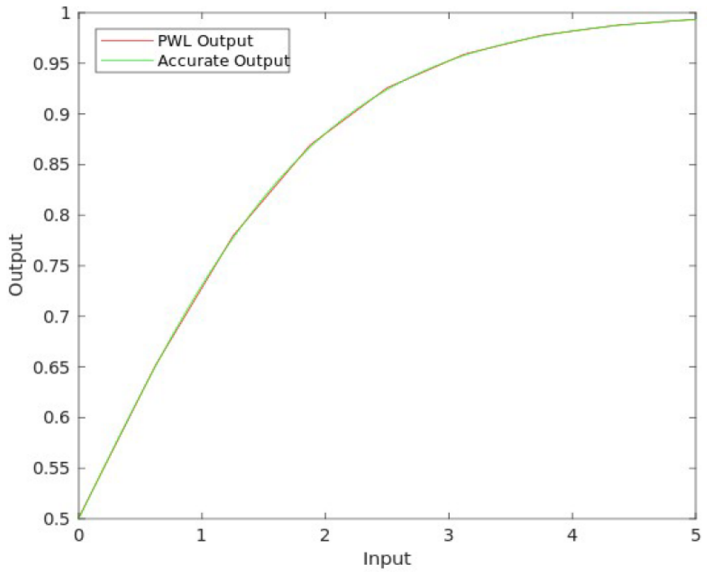


Illustration of Sigmoid PWL Output vs. Accurate Sigmoid Output

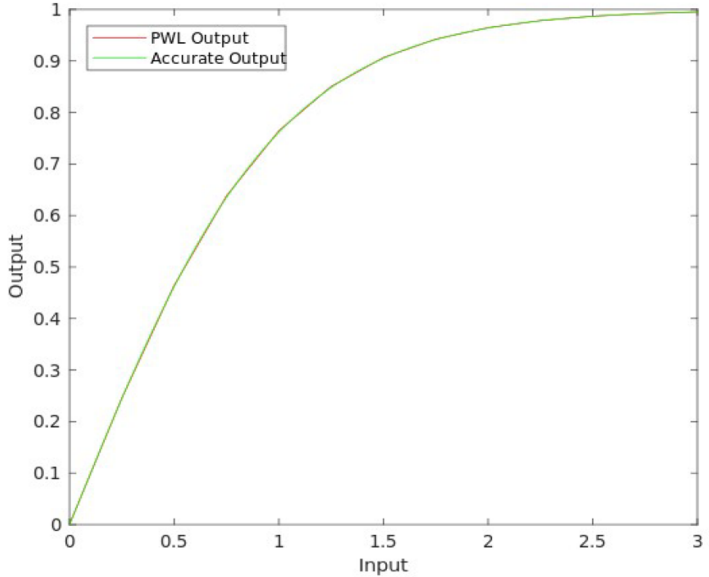


Illustration of Tanh PWL Output vs. Accurate Tanh Output

Reference: Algorithmic C (AC) Math Library Reference Manual, October 2018

SystemC Simulation Result

- Simulated by random values
  8-bit fixed-point data type
  - 6-bit fractional part
  - Precision ≅ 0.016

  The error between PE output and reference is within precision.

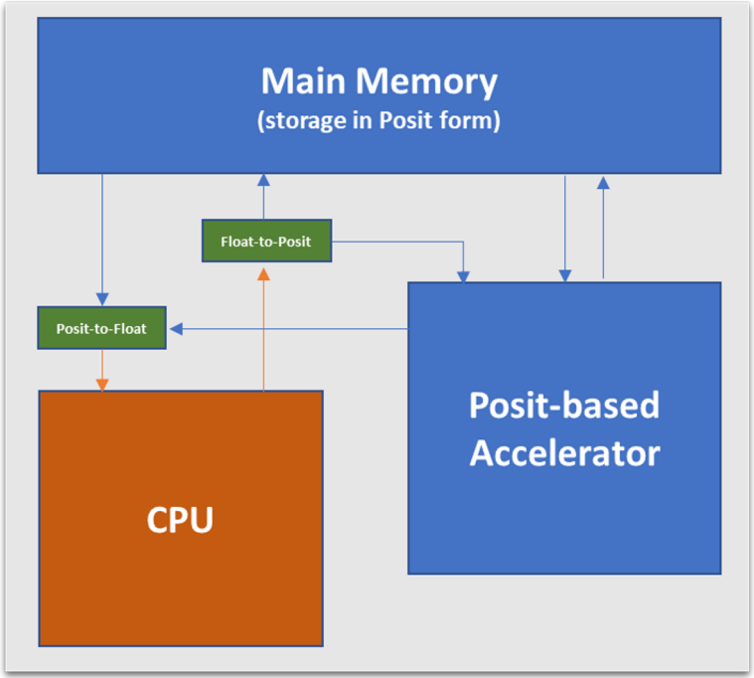
  SystemC code can be synthesized to RTL (Verilog)

```
            SystemC 2.3.0-ASI --- Oct 25 2016 14:51:58
        Copyright (c) 1996-2012 by all Contributors,
        ALL RIGHTS RESERVED

================================
SETTING RANDOM SEED = 1543548244
================================
@4 ns Asserting reset
@8 ns De-Asserting reset
@21 ns  tb.pemodule_inst MAC done
@23 ns Row = 0 output c = 0.21875 | reference = 0.234487
@24 ns Row = 0 output h = 0.1875 | reference = 0.193353
@33 ns  tb.pemodule_inst MAC done
@35 ns Row = 1 output c = 0.078125 | reference = 0.0791525
@36 ns Row = 1 output h = 0.078125 | reference = 0.0789873
@45 ns  tb.pemodule_inst MAC done
@47 ns Row = 2 output c = 1.53125 | reference = 1.53307
@48 ns Row = 2 output h = 0 | reference = 0.000154574
@57 ns  tb.pemodule_inst MAC done
@59 ns Row = 3 output c = -1.03125 | reference = -1.0241
@60 ns Row = 3 output h = -0.78125 | reference = -0.771531
@69 ns  tb.pemodule_inst MAC done
@71 ns Row = 4 output c = 0 | reference = -6.40975e-06
@72 ns Row = 4 output h = 0 | reference = -7.52374e-09
@81 ns  tb.pemodule_inst MAC done
@83 ns Row = 5 output c = -0.03125 | reference = -0.0222118
@84 ns Row = 5 output h = -0.015625 | reference = -0.013651
@93 ns  tb.pemodule_inst MAC done
@95 ns Row = 6 output c = 1.17188 | reference = 1.18571
@96 ns Row = 6 output h = 0.203125 | reference = 0.208639
@105 ns tb.pemodule_inst MAC done
@107 ns Row = 7 output c = -0.40625 | reference = -0.399427
@108 ns Row = 7 output h = -0.375 | reference = -0.378441
```

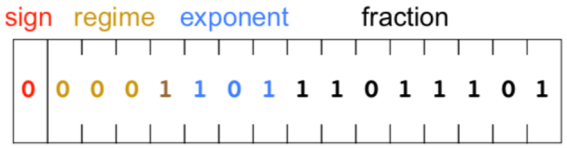
Exploration of a Posit-based SoC

- Float-to-Posit and Posit-to-Float converters for translating CPU-Accelerator and Memory-CPU communications
- Posit-based adder, multiplier and accumulator for use in the Posit-based accelerator
- Main on-chip memory store weights and activations in posit form



Concept of Posit-based Soc

Posit C++ Implementation



- Achieve:
  - Transform between posit number ⟷ float number ⟷ fixed-point number
  - Make posit support add and multiple operations

Algorithm Key Point: Determine and assign different section bits

Accuracy Measurement

  - Relative error
  - Decimal error
  - Decimal accuracy

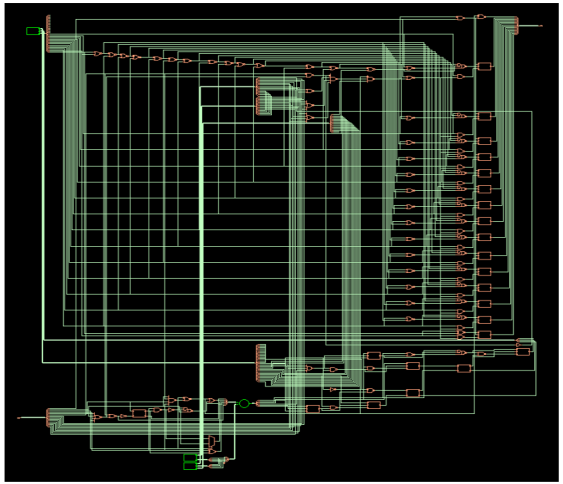
$$\varepsilon_{relative} = \left| \frac{x_{computed} - x_{exact}}{x_{exact}} \right| \longrightarrow \varepsilon_{decimal} = \left| \log_{10}\left(\frac{x_{computed}}{x_{exact}}\right) \right| \longrightarrow Q_{decimal} = -\log_{10} \left| \log_{10}\left(\frac{x_{computed}}{x_{exact}}\right) \right|$$

Compare Posit & Float

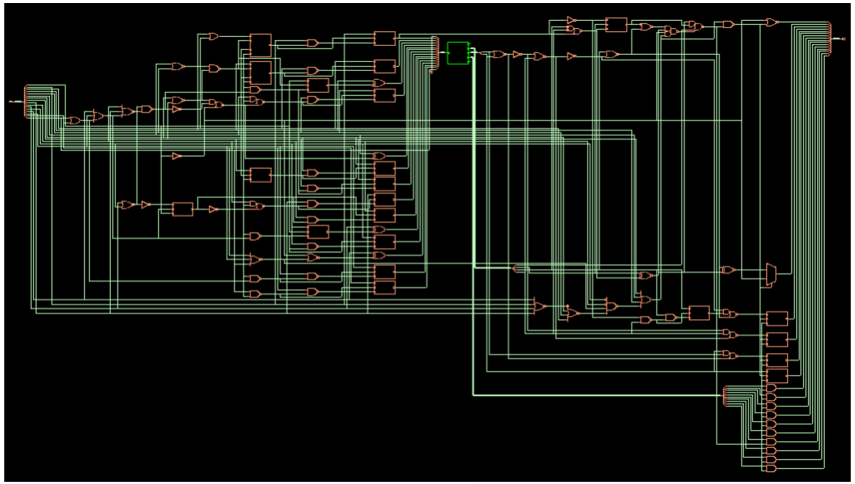
Bit Size	Float exp Size	Float Dynamic Range	Posit es Value	Posit Dynamic Range	Range (Posit Accuracy > Float Accuracy)
8	3	0.008 ~ 2e1	0	0.008 ~ 1e2	¼ ~ 4
16	5	3e-8 ~ 7e-4	1	9e-9 ~ 1e9	1/64 ~ 64
32	8	7e-46 ~ 3e38	2	5e-38 ~ 2e37	1e-6 ~ 1e6

Cost of Posit Hardware

- Float to Posit Hardware (Tech: 90nm)

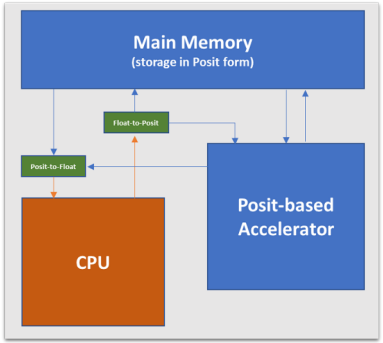


- Posit to Float Hardware (Tech: 90nm)



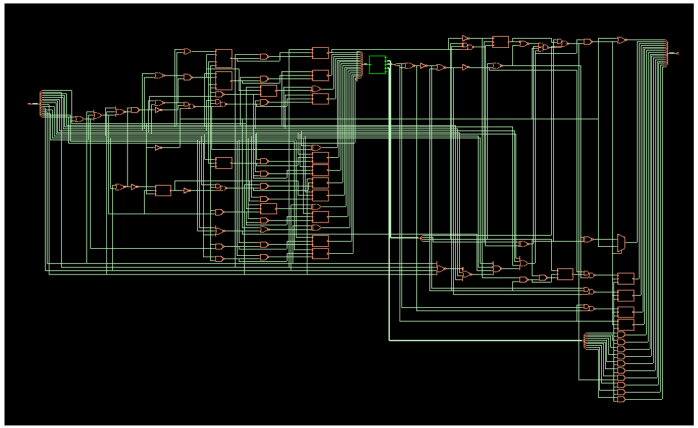
- Cost (Power & Area) Compare

Type	Power(nW)	Area(um^2)
32-bit Float to 8-bit Posit	19841.83	774.49
8-bit Posit to 32-bit Float	19211.70	695.01

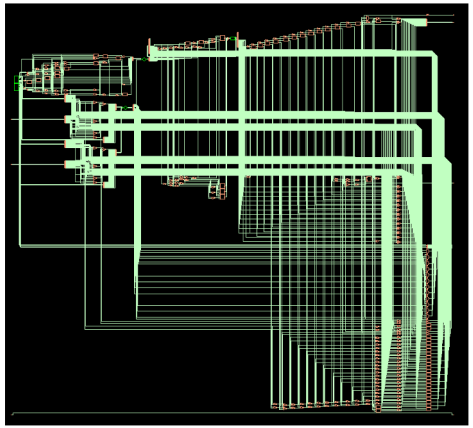


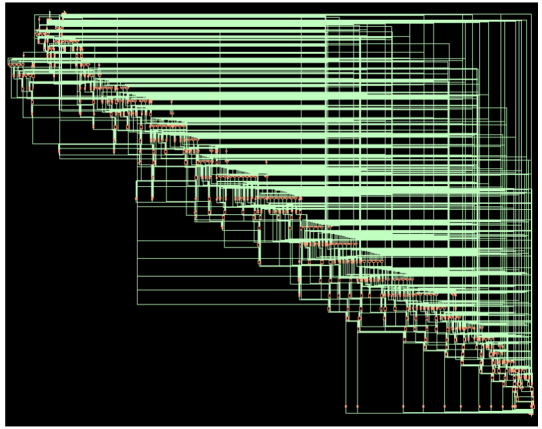
Cost of Posit Hardware

- Posit Adder (Tech: 90nm)

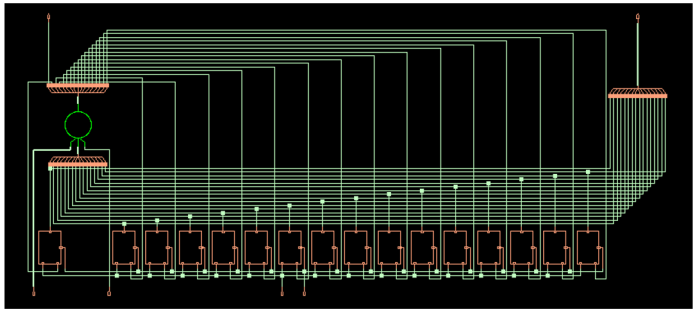


- Posit Multiplier (Tech: 90nm)





- Float Adder (Tech: 90nm)



- Cost (Power & Area) Compare

Type	Power(nW)	Area(um^2)
8-bit Posit Adder	7847.93	274.32
8-bit Posit Multiplier	298139.76	5090.91
8-bit Fixed-point Adder	8465.93	336.58
32-bit Float-point Adder	27753.62	1153.33

Conclusion

- We have **demonstrated**:
  - the impact of various bit quantization on speech-to-text inference. 8-bit Posit is a promising candidate for NLP inference
  - the impact of sparsity on WER performance
  - that Posit-based hardware is less expensive in terms of power and area compared to float
- We have **designed** an LSTM-based processing element and **showed** that its results are pretty close to the reference results
- **Ongoing work** includes:
  - High-level synthesis of the NLP processing element
  - Variable bit length Python implementation for Posit