Design and Implementation of LSTM Digital pre-distortion model inference on RISC V

Shaswat Satapathy (309591029)
Shivani Singh (309591030)

Abstract

- With the ever growing number of devices, the signals are getting more complex with dynamic behaviour, which causes non-linearity in the Power Amplifier (PA) output.
- Digital Pre-distortion (DPD) is considered an effective technique to reduce these effects and to make this system more real-time, scholars are now diving into deep learning methodologies like CNN and LSTM.
- To run these algorithms in devices with less capability of resource storage and computation, is a major challenge since these models are computational and memory escalated.
- We need a scalable and flexible implementation to meet requirements from IoT to high-end applications, supporting both inference and on-device learning for edge devices.
- In this work, we propose a step-by-step guide to build LSTM deep learning based DPD model inference accelerators using RISC V ISA.

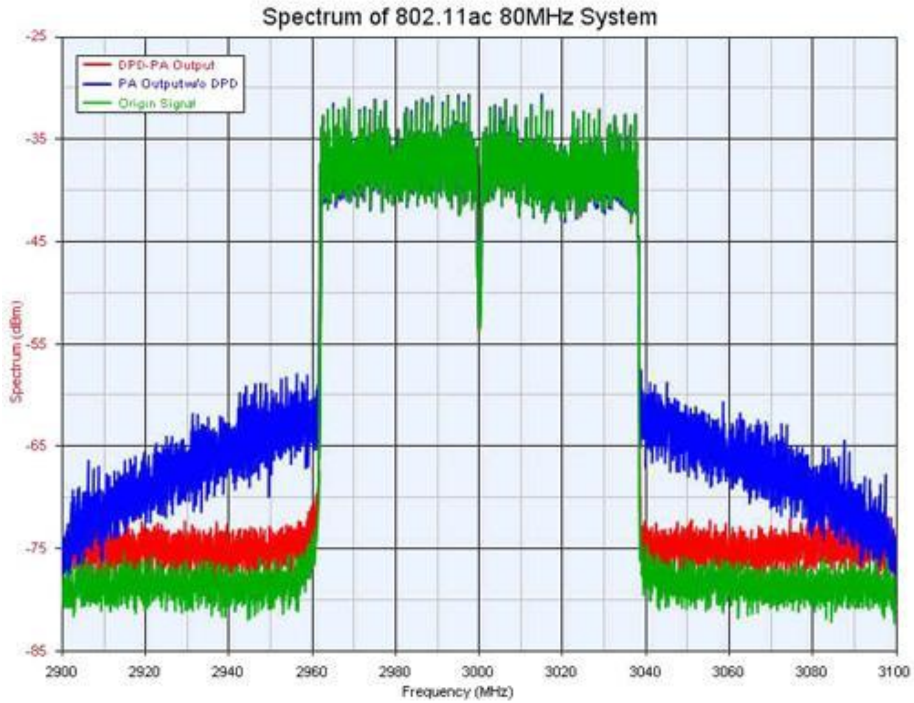
Contents

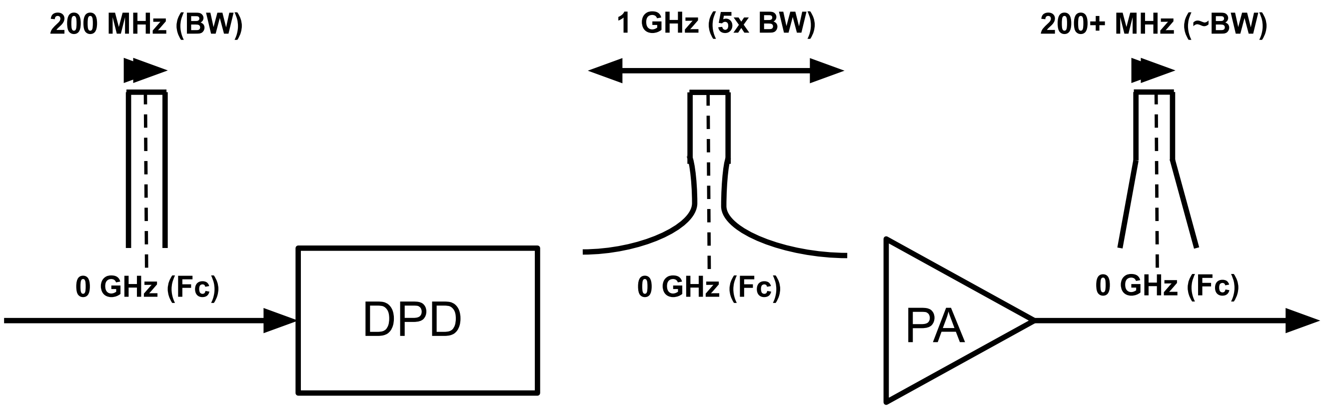
- Introduction to DPD
- Introduction to LSTM
- Software Implementation
- Stratus HLS Implementation
- RISC V Implementation
  - single core
  - Multi - core

  Comparative study
- Result
- Future work

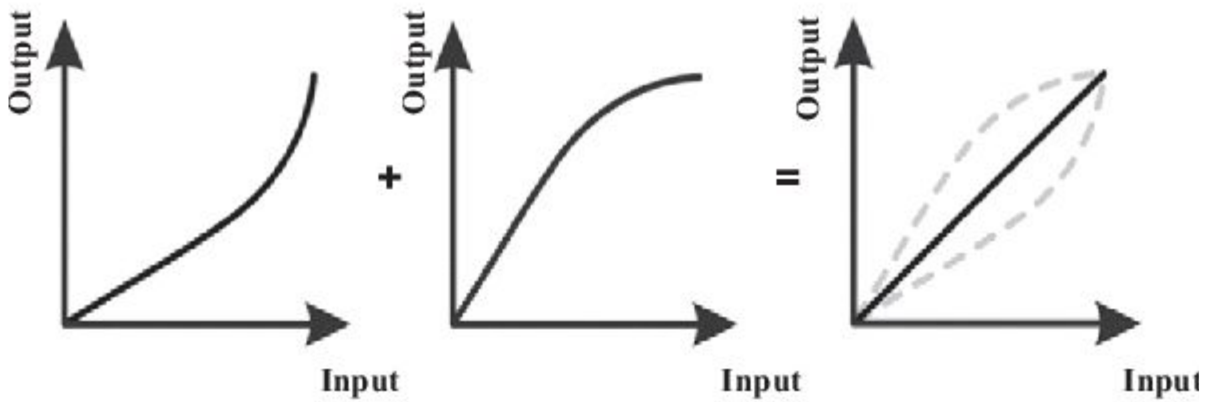
What is Digital Perdistortion (DPD)

- A technique for improving the linearity of power amplifiers
- Ideally the output signal of a PA is the input scaled up perfectly
- Instead the semiconductor physics causes distortions
  - Amplitude, frequency and phase errors
- If we can predict the errors, we can try to reverse them



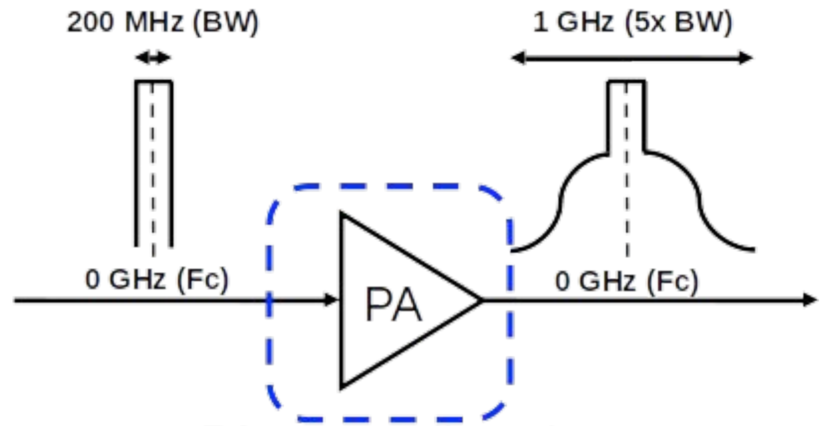


Mitigate PA output distortion **through Pre-distortion**



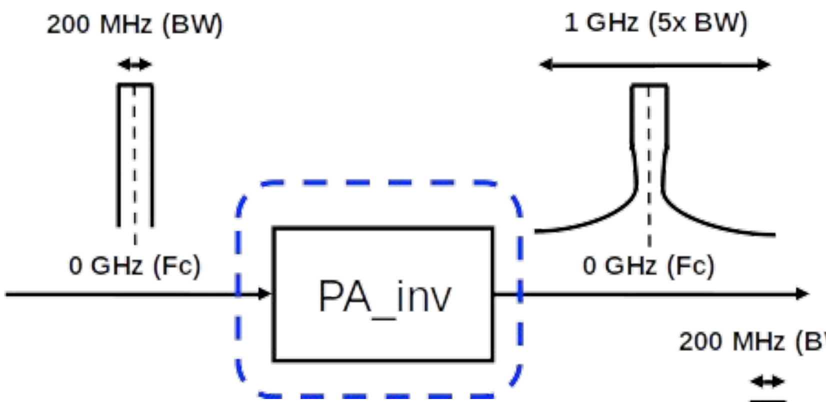
Implementation steps

- Step - 1:

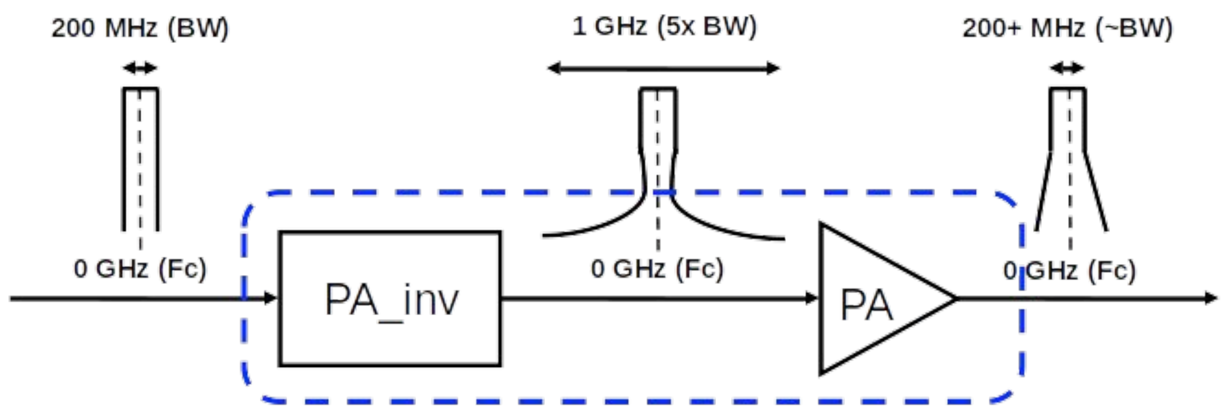


PA output distortion

- Step - 2



- Step - 3

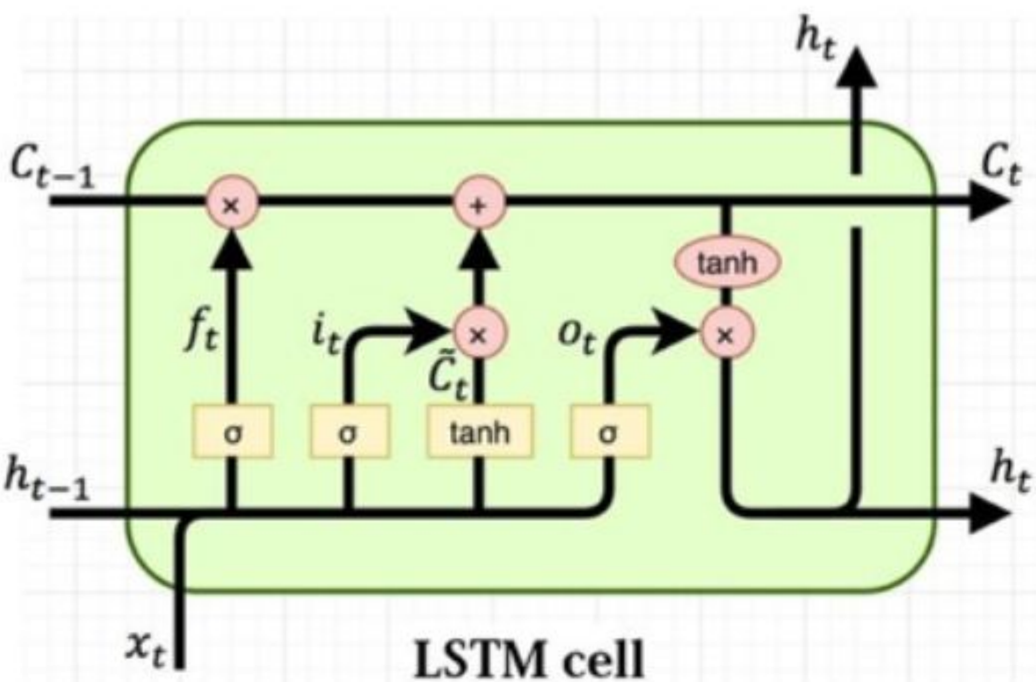


Mitigate PA output distortion **through Predistortion**

Introduction to LSTM

- Long Short Term Memory networks - usually just called "LSTMs" - are a special kind of Recurrent Neural Network (RNN), capable of learning long-term dependencies.

- Unlike regular RNN networks, LSTMs also have this chain like structure but the repeating module has a different structure.



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$

$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$

$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$

$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$

$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$

$$h_t = \tanh(C_t) * o_t$$

LSTM Gates

- Input gate i:
  - Takes previous output h_{t-1} and current input x_t.
  - $i_t \in (0, 1)$
  - $i_t = \sigma(\theta_{xi} x_t + \theta_{ht} h_{t-1} + b_i)$
- Forget gate f:
  - Takes previous output h_{t-1} and current input x_t.
  - $f_t \in (0, 1)$
  - $f_t = \sigma(\theta_{xf} x_t + \theta_{hf} h_{t-1} + b_f)$
  - If $f_t = 0$: **Forget** previous state, otherwise pass through prev. state.
- Read gate g:
  - Takes previous output h_{t-1} and current input x_t.
  - $g_t \in (0, 1)$

LSTM Gates

- Cell gate c:
  - New value depends on f_t, its previous state c_{t-1}, and the read gate g_t.
  - Element-wise multiplication: $c_t = f_t \odot c_{t-1} + i_t \odot g_t$.
  - We can learn whether to **store** or **erase** the old cell value.
- Output gate o:
  - $o_t = \sigma(\theta_{xo} x_t + \theta_{ho} h_{t-1} + b_o)$
  - $o_t \in (0, 1)$
- New output gate h:
  - $h_t = o_t \odot \tanh(c_t)$
  - Will be fed as input into next block.

Software implementation

```
# In[] Load Data


data = sio.loadmat('../Data/outdatapa_1G.mat')
data = sio.loadmat('../Data/indatapa_1G.mat')

# In[] Plot Data

plt.plot(np.reshape(indata_1G,(-1,1)),color='r')
plt.plot(np.reshape(outdata_1G,(-1,1)),color='b')
plt.show()

#spectrum_ = lambda x: np.fft.fftshift(np.fft.fft(x, fftLen)) / Fs * (len(u))

spectrum = lambda x: 5*np.log10(np.fft.fftshift(np.fft.fft(x, fftLen)) / Fs * (len(indata_1G)))
```

```python
# In[] Prepare Training Model

out_real,  out_img = np.reshape(outdata_1G.real,(-1,1)),np.reshape(outdata_1G.imag,(-1,1))
outp = [0]*4000
j = 0
for i in range(4000):
        if (i%2==0):
            outp[i]=out_real[j]

        else:
            outp[i]=out_img[j]
            j=j+1

oo = np.asarray(outp)
np.save("outp.npy",oo)
```

```
# In[4]  Network and Parameter

i_r = Input(batch_shape=(batch_size, timesteps, input_dim), name='main_input')
i_i = Input(batch_shape=(batch_size, timesteps, input_dim), name='aux_input')

x = concatenate([i_r, i_i])
o = LSTM(400, return_sequences=True, stateful=True)(i_r)

# In[4]   Training

history_r = m_r.fit({'main_input': train_x_r, 'aux_input': train_x_i},{'main_output': train_y_r, 'aux_ou

# In[4]   Load Model

m_r.save_weights('./weight/LST.h5')
m_r.load_weights('./weight/LST.h5')


m_r.save_weights('./weight/LSTM.h5py')
m_r.load_weights('./weight/LSTM.h5py')
```

```
# In[4]: Prediction

predict_r = m_r.predict({'main_input': train_y_r})
predict_r_ = np.reshape(predict_r,(-1,1))


trainPredict_r = np.reshape(scaler_ur.inverse_transform(predict_r_),(-1,))
```

Results

- The time taken when performing LSTM in software is **113s**.

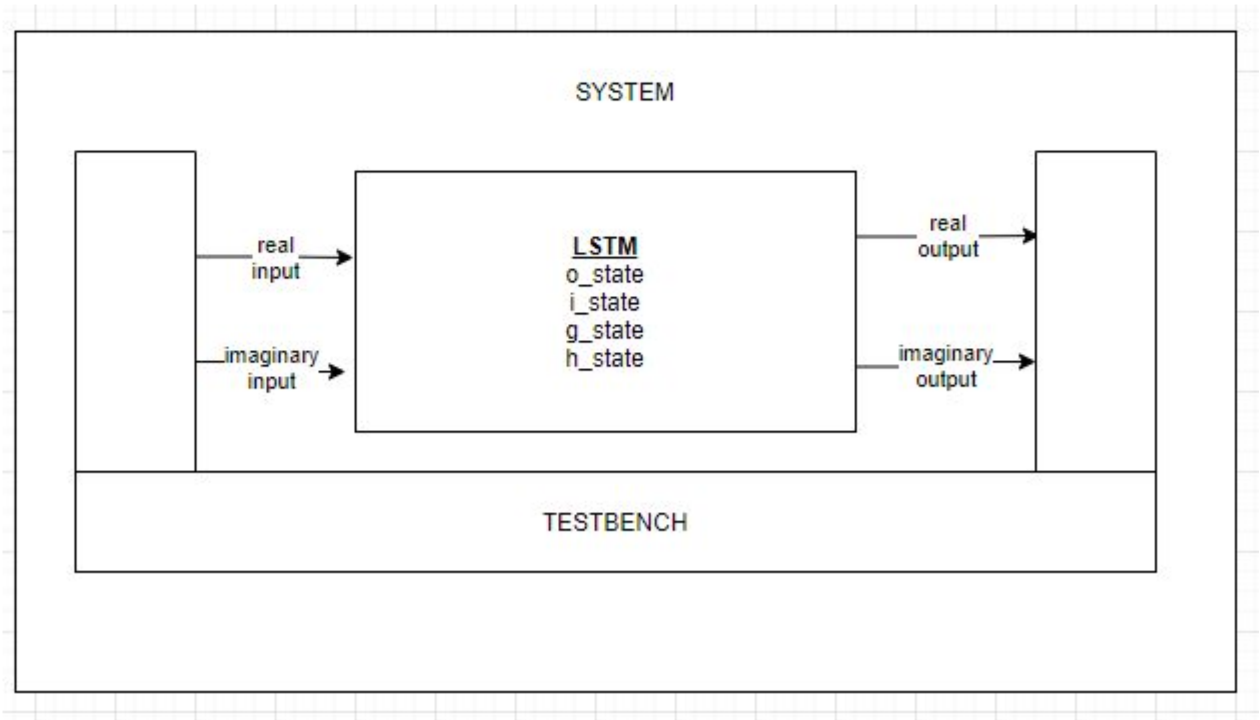
```
sw_ocr = lstm.PynqFrakturOCR(lstm.RUNTIME_SW)

sw_result = sw_ocr.inference(im)
sw_mops_per_s, sw_ms_inference_time, sw_recognized_text = sw_result

print("SW OCRed text: {}".format(sw_recognized_text))
print("SW MOps/s: {}".format(sw_mops_per_s))
print("SW inference time [ms]: {}".format(sw_ms_inference_time))
```

```
SW OCRed text: spruches nichts, daß eine leise Bitterkeit oder ein Wort der Resig-
SW MOps/s: 1.2346284082729002
SW inference time [ms]: 113142.3828125
```

Stratus HLS implementation



Results

```
i_state-0.122049
o_state0.0728104
g_state0.0726688
h_state0.126844
i_state-0.0239955
o_state0.05687
g_state0.0543943
h_state-0.132827
i_state-0.011761
o_state-0.0470634
g_state-0.00885692
h_state-0.15961
i_state-0.0511192
o_state-0.0818065
g_state-0.00753786
h_state-0.0989751
i_state-0.114914
o_state-0.028768
g_state0.0738344
h_state0.120309
i_state-0.0676995
o_state-0.0603062
g_state0.00581274
h_state-0.139757
i_state0.108128
o_state0.0125849
g_state0.0277029
h_state-0.118148
LSTM REAL AND IMAGINARY VALUES
Info: /OSCI/SystemC: Simulation stopped by user.
Total run time = 14417910 ns
Simulated time == 14417970 ns
```

Simulated time:

0.0144 sec

RISC V LSTM Implementation single core

LSTM in VP:

```
#main.cpp

addr_t lstm_start_addr = 0x77000000;
addr_t lstm_size = 0x01000000;
addr_t lstm_end_addr = lstm_start_addr + lstm_size - 1;
bool use_E_base_isa = false;

bus.ports[15] = new PortMapping(opt.lstm_start_addr, opt.lstm_end_addr);

bus.isocks[15].bind(lstm.tsock);

#lstm.h

struct lstm : public sc_module {
//lstm input_1 and input2
 sc_fifo<int> i_1;//since the total input is flatten so at every even position real
 sc_fifo<int> i_2;//after every real is its corresponding imaginary(odd pos)
//output_1 and output2
 sc_fifo<int> o_result1;
 sc_fifo<int> o_result2;
```

```
 i_state = bias_lstm[v] + ker[v]*input1 + ker[N1+v]*input2;
     for(j =0;j<N1;j++){
        recur_ker_l[j] = recur_ker[first + j];
        }
     i_state = i_state + hidden_multiply(h_state,recur_ker_l);
     i_state = sigmoid(i_state);
f_state = bias_lstm[N1+v] + ker[2*N1+v]*input1 + ker[3*N1+v]*input2;
o_state = bias_lstm[2*N1+v] + ker[4*N1+v]*input1 + ker[5*N1+v]*input2;
g_state = bias_lstm[3*N1+v] + ker[6*N1+v]*input1 + ker[7*N1+v]*input2;

 int result1 = (int)((output_main + main_bias)*precision);
 int result2 = (int)((output_aux+ aux_bias)*precision);

 o_result1.write(result1);
 o_result2.write(result2);
```

LSTM in SW

```
#main.cpp

void write_data_to_ACC(char* ADDR, int buffer, int len){
    unsigned char buff[4];

        data.uint = buffer;
        buff[0] = data.uc[0];
        buff[1] = data.uc[1];
        buff[2] = data.uc[2];
        buff[3] = data.uc[3];

     if(_is_using_dma){
        // Using DMA
        *DMA_SRC_ADDR = (uint32_t)(buff);
        *DMA_DST_ADDR = (uint32_t)(ADDR);
        *DMA_LEN_ADDR = len;
        *DMA_OP_ADDR  = DMA_OP_MEMCPY;
      }else{
        // Directly Send
        memcpy(ADDR, buff, sizeof(unsigned char)*len);
      }
    }

#include "input_real.h"
#include "input_imag.h"

 write_data_to_ACC(lstm_START_ADDR, buffer_r, 4);
 write_data_to_ACC(lstm_START1_ADDR, buffer_i, 4);


 read_data_from_ACC(lstm_READ_ADDR, buffer_r, 4);
 read_data_from_ACC(lstm_READ1_ADDR, buffer_i, 4);
```

```
Info: /OSCI/SystemC: Simulation stopped by user.
=[ core : 0 ]============================
simulation time: 229953920 ns
zero (x0) =        0
ra   (x1) =    104a2
sp   (x2) =  1ffffec
gp   (x3) =    24cb0
tp   (x4) =        0
t0   (x5) =        0
t1   (x6) =     36c0
t2   (x7) =        1
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =        0
a2  (x12) =     10f9
a3  (x13) =        0
a4  (x14) =        0
a5  (x15) =        0
a6  (x16) =        1
a7  (x17) =       5d
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        0
t4  (x29) =        2
t5  (x30) =     8800
t6  (x31) =        5
pc = 1b422
num-instr = 7244726
```

RISC V LSTM Implementation multi - core

```
#main.cpp

//address
static char* const lstm_START_ADDR = reinterpret_cast<char* const>(0x45000000);
static char* const lstm_START1_ADDR = reinterpret_cast<char* const>(0x45000036);

// Gaussian Filter ACC 1
static char* const lstm1_START_ADDR = reinterpret_cast<char* const>(0x45000000);;
static char* const lstm1_START1_ADDR = reinterpret_cast<char* const>(0x45000036);

//dma read
void read_data_from_ACC(char* ADDR, int buffer, int len){
    unsigned char buff[4];
     if(_is_using_dma){
        // Using DMA
        *DMA_SRC_ADDR = (uint32_t)(ADDR);
        *DMA_DST_ADDR = (uint32_t)(buff);
        *DMA_LEN_ADDR = len;
        *DMA_OP_ADDR  = DMA_OP_MEMCPY;
      }else{
        // Directly Send
        memcpy(buff,ADDR, sizeof(unsigned char)*len);
      }
      data1.uc[0] = buff[0];
      data1.uc[1] = buff[1];
      data1.uc[2] = buff[2];
      data1.uc[3] = buff[3];
     buffer = data1.uint;
    }
```

```c
//mutex lock
sem_wait(&lock);
          if (hart_id == 0) {
          write_data_to_ACC(lstm_START_ADDR, buffer_r, 4);
          write_data_to_ACC(lstm_START1_ADDR, buffer_i, 4);
          }
          else {
          write_data_to_ACC(lstm1_START_ADDR, buffer_m, 4);
          write_data_to_ACC(lstm1_START1_ADDR, buffer_n, 4);
          }
          sem_post(&lock);
```

```
        SystemC 2.3.3-Accellera --- Jun  8 2021 14:00:49
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
hart_id = 1
======================================
          Reading from array
======================================
 input_rgb_raw_data_offset      = 8
 width                          = 2000
 length                         = 0
 bytes_per_pixel                = 32
======================================
hart_id = 0
======================================
          Reading from array
======================================
 input_rgb_raw_data_offset      = 8
 width                          = 2000
 length                         = 0
 bytes_per_pixel                = 32
======================================
real :[1000 1000] -4816 -4816 -4816 -4816
imag :[1000 1000] -25385 -25385 -25385 -25385
real :[1001 1001] 18537 18537 18537 18537
imag :[1001 1001] -28641 -28641 -28641 -28641
real :[1002 1002] 33889 33889 33889 33889
imag :[1002 1002] -28295 -28295 -28295 -28295
real :[1003 1003] 36360 36360 36360 36360
imag :[1003 1003] -22631 -22631 -22631 -22631
real :[1004 1004] 25142 25142 25142 25142
imag :[1004 1004] -11178 -11178 -11178 -11178
real :[1005 1005] 3577 3577 3577 3577
imag :[1005 1005] 4330 4330 4330 4330
real :[1006 1006] -22176 -22176 -22176 -22176
imag :[1006 1006] 20058 20058 20058 20058
real :[1007 1007] -45059 -45059 -45059 -45059
imag :[1007 1007] 31151 31151 31151 31151
real :[1008 1008] -59125 -59125 -59125 -59125
imag :[1008 1008] 33700 33700 33700 33700
real :[1009 1009] -61016 -61016 -61016 -61016
imag :[1009 1009] 26624 26624 26624 26624
real : [0 0] -28790 -28790 -28790 -28790
imag : [0 0] 13390 13390 13390 13390
real : [1 1] -24153 -24153 -24153 -24153
```

Multi core LSTM (2 Processors)

```
Info: /OSCI/SystemC: Simulation stopped by user.
=[ core : 0 ]============================
simulation time: 27873540 ns
zero (x0) =        0
ra   (x1) =    10c38
sp   (x2) =    18d00
gp   (x3) =    36020
tp   (x4) =        0
t0   (x5) =  2010000
t1   (x6) =        1
t2   (x7) =        1
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =    37b48
a2  (x12) =        1
a3  (x13) =       25
a4  (x14) =        1
a5  (x15) =        0
a6  (x16) =        0
a7  (x17) =       5d
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        3
t4  (x29) =        2
t5  (x30) =        0
t6  (x31) =        0
pc = 10c64
num-instr = 922647
=[ core : 1 ]============================
```

```
=[ core : 1 ]============================
simulation time: 27873540 ns
zero (x0) =        0
ra   (x1) =    10c38
sp   (x2) =    20d00
gp   (x3) =    36020
tp   (x4) =        0
t0   (x5) =    20d00
t1   (x6) =        1
t2   (x7) =        1
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =    37b48
a2  (x12) =        1
a3  (x13) =       2a
a4  (x14) =        2
a5  (x15) =        0
a6  (x16) = fefefeff
a7  (x17) =       40
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        3
t4  (x29) =        2
t5  (x30) =     8800
t6  (x31) =        5
pc = 10c4c
num-instr = 952521
user@ubuntu:~/ee6470/riscv-vp/sw/lstm-multicore$
```

Comparative study

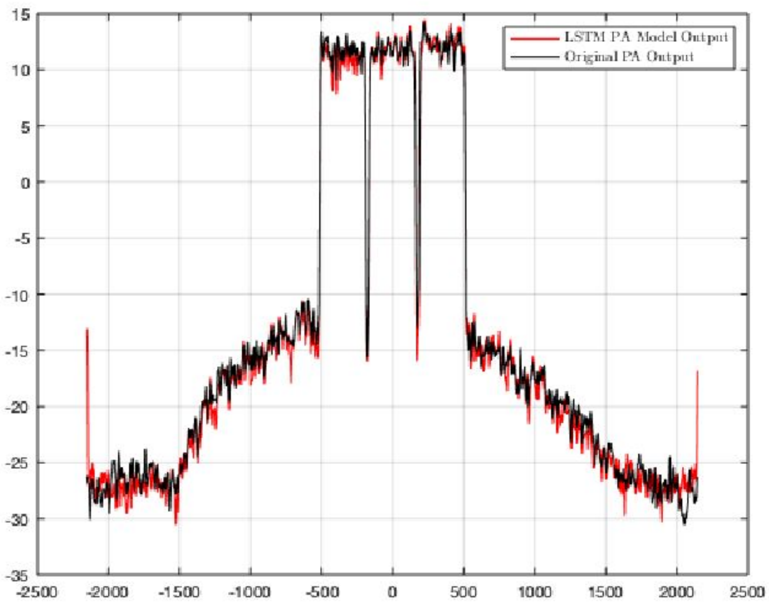
	Simulated Time
Single - Core	229953920 ns
Multi - Core (2 cores)	27873540 ns
DIFFERENCE	~8x reduced

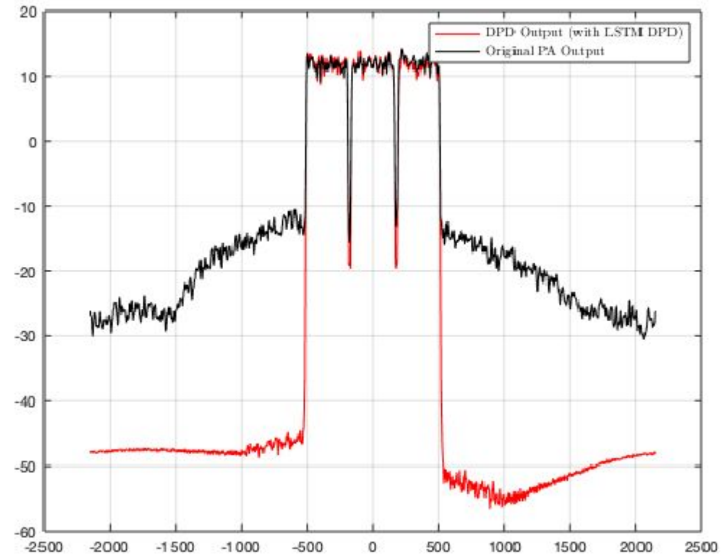
	Simulated time
Software implementation	1131425656 ns
RISC V implementation	27873540 ns
DIFFERENCE	~40x reduced

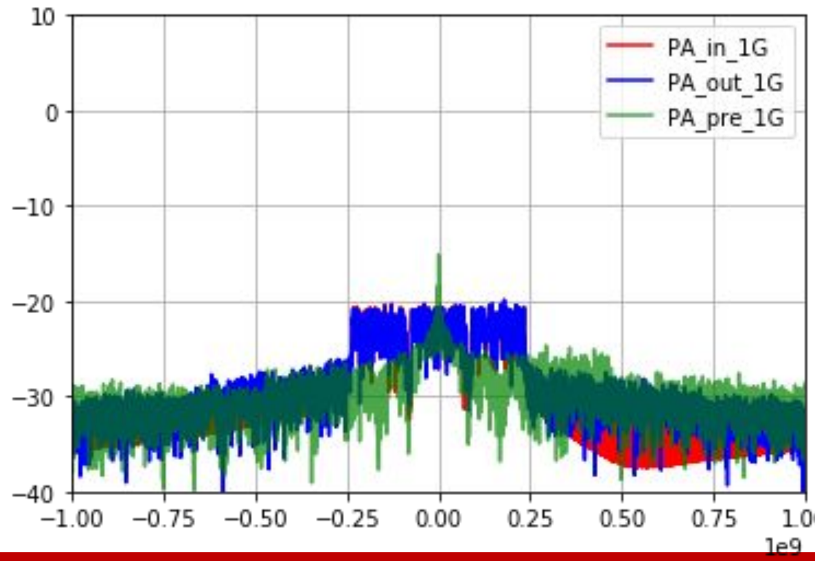
Results

LSTM DPD Result

LSTM PA Model Performance

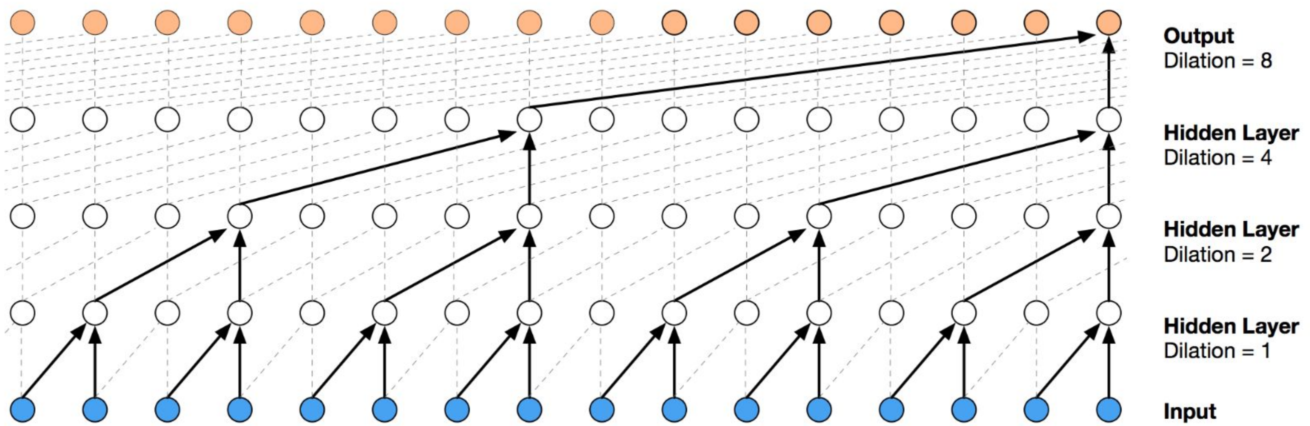






Future work

- Since the data from a power amplifier is 1-Dimensional, we will explore a different type of convolution called **Temporal convolution network** with dilation. Introducing dilation will help us take even smaller details in the put.



- As you can see it has a directional structure, which captures dependencies between the input (in our case words) and aggregates into a number of units. Similar to what LSTM and GRU does, however with less loops.

Thank You!

Questions??