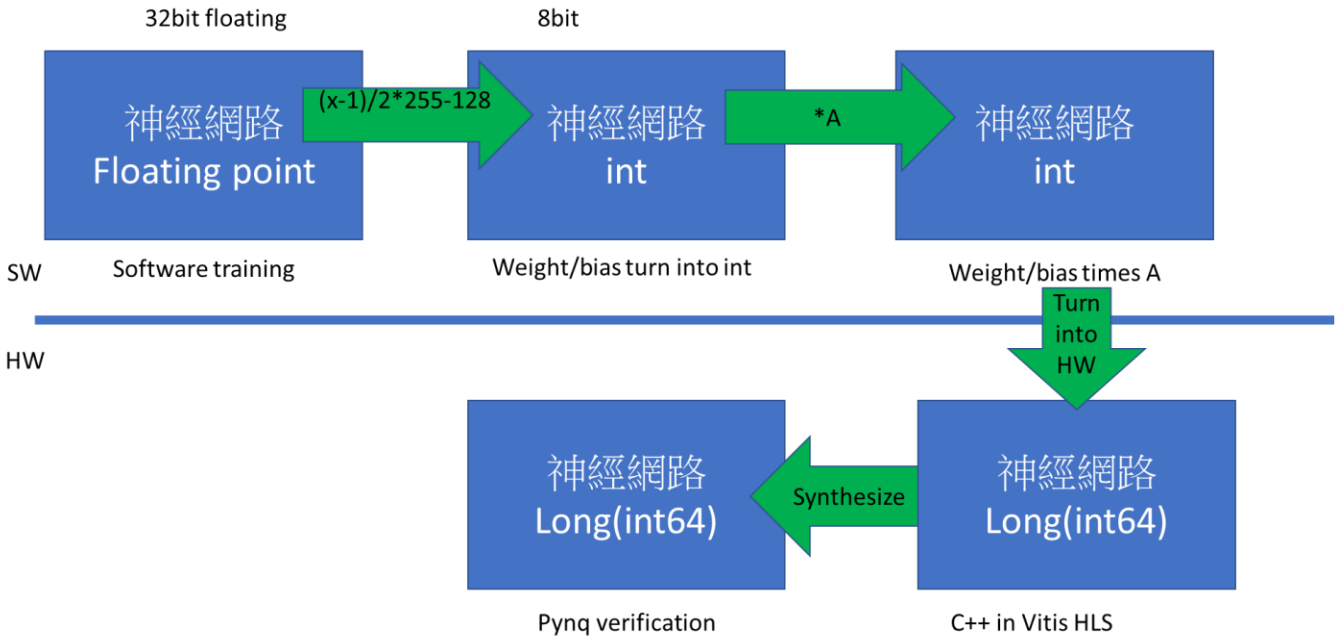
AN code Hardware System

為了將 AN code 嵌入整個神經網路，我做了以下實驗。

以 64bit 為 datatype，A=131 合成硬體。

選 64bit 當作 datatype 的原因

以目前的的轉換方式為例



在將 parameter 轉乘 8bit int 再乘上 A，計算最大的數值範圍不會超過 50bit，用 32bit 太小所以用 64bit 當作儲存 parameter 的資料型態。

選 A=131 的原因

由於要確認傳輸資料(codeword)是否正確，選定(A-1)/2 要能夠包含 64bit 的數字，故選最為接近的數字 131。

實驗結果

	BRAM_18K	DSP	FF	LUT	acc	timeit*
No encoded	138	192	46819	27692	0.966	40.5
Encoded	295	352	62733	34545		
Encoded no pipeline	203	204	43166	24795	0.967	40.3

* timeit: 跑 10 loops 取最快前三名做平均，當作此運算單元的運算速度，越短越好。

```
hit=0
start_time=time.time()
for i in range(1000):
    inBuffer0[:]=img[i]
    dma.sendchannel.transfer(inBuffer0)
    dma.recvchannel.transfer(outBuffer)
    ol.top_0.write(0x00, 0x01)
    while(ol.top_0.register_map.CTRL.AP_DONE==0):
        pass
    if(outBuffer.argmax()==label[i]):
        hit+=1
    #print("times: ",i+1,"hit: ",hit)
print("acc",hit/1000)
print("spend time",time.time()-start_time)
```

```
acc 0.967
spend time 40.6138596534729
```

Encoed

```
hit=0
start_time=time.time()
for i in range(1000):
    inBuffer0[:]=img[i]
    dma.sendchannel.transfer(inBuffer0)
    dma.recvchannel.transfer(outBuffer)
    ol.top_0.write(0x00, 0x01)
    while(ol.top_0.register_map.CTRL.AP_DONE==0):
        pass
    if(outBuffer.argmax()==label[i]):
        hit+=1
    #print("times: ",i+1,"hit: ",hit)
print("acc",hit/1000)
print("spend time",time.time()-start_time)
```

```
acc 0.966
spend time 40.62453269958496
```

No encoded

比較 AN decoder 與 Barrett reduction decoder

為了觀察使用 barret reduction 與使用除法運算單元(/)的硬體差異，我做了以下實驗:

分別只合成 an decoder 與 barrett reduction decoder 觀察使用硬體資源與 latency。

Pipeline(Yes)	Latency(ns)	DSP	FF	LUT

`void andecoder(DataType codeword[10]){`
`  andecoder_label_unroll:`
`  for(int i=0;i<10;i++){`
`  codeword[i]=codeword[i]/131;`
`  }`
`}`
	190.00	9
		100%
			1031
			100%
				717
				100%

`void barrett_decode(DataType codeword[10]){`
`  //no error ,do not need residue  r`
`  DataType r,q;`
`  barrett_decode_label0_unroll:`
`  for (int i=0;i<10;i++){`
`  std::tie(r,q)=dec_131(codeword[i]);`
`  codeword[i]=q;`
`  }`
`}`
	240.00	8
		88.8%
			1140
			100.7%
				689
				96.1%

比較 unroll 之後硬體差距。

Unroll(Yes)	Latency(ns)	DSP	FF	LUT

<pre>void andecoder(DataType codeword[10]){
  andecoder_label_unroll:
  for(int i=0;i<10;i++){
  codeword[i]=codeword[i]/131;
  }
}</pre>
	120.00	90
		100%
			7711	6155

<pre>void barrett_decode(DataType codeword[10]){
  //no error ,do not need residue r
  DataType r,q;
  barrett_decode_label0_unroll:
  for (int i=0;i<10;i++){
  std::tie(r,q)=dec_131(codeword[i]);
  codeword[i]=q;
  }
}</pre>
	170.00	80
		88.8%
			9158
			118.7%
				5583
				90.7%

問題與討論

由於再做 barrett reduction decoder 時，barrett 運算涵蓋範圍要包含到 64bit 以內的數字，所以 w 會很大，17189311554849 為 44bit。所以在硬體內做乘法運算時要做 explicit casting 才能夠確保位移時能夠拿到正確的數值。

經實驗測試能夠使用 ap_int<96>代替 128bit。 節省一些 LUT 和 FF。

```
std::tuple<DataType, DataType>  dec_131(DataType x){
    int_128     q=((int_128)x*17189311554849); //explicit casting
                q=q>>51;

    DataType    r= x-(long)q*131;
    if(r<131)
        return std::make_tuple(r,q);
    else
        return std::make_tuple(r-131,q+1);

}
```