FIR_example

➢ Introduction

Finite Impulse Response (FIR) Filter

In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time.

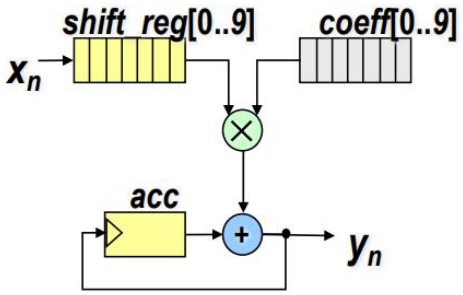
$$y[n] = \sum_{i=0}^{N} b_i x[n-i]$$

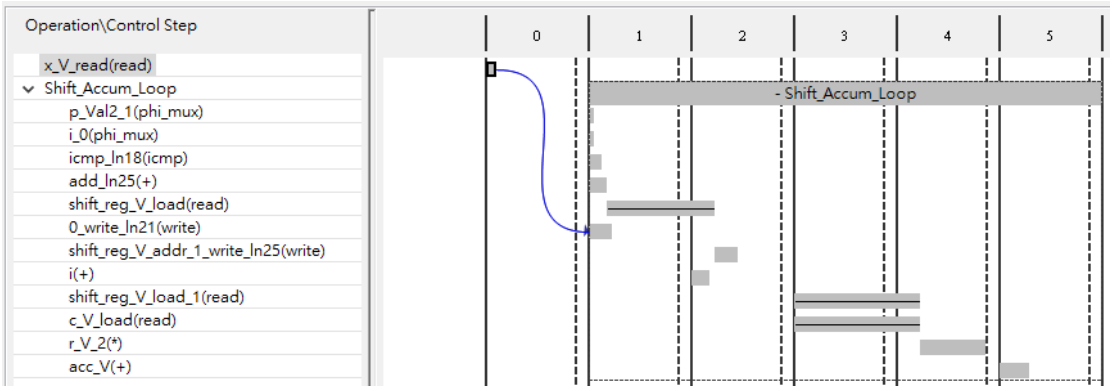
$x[n]$	input signal
$y[n]$	output signal
N	filter order
b_i	ith filter coefficient

I did some optimizations of FIR Operation in this lab

➢ Solution 1 – Baseline

Baseline architecture





There're totally 16 loops, which each one spends 5 cycles to complete.

Thus, latency = 5*16+1 cycles

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.702 ns	1.25 ns

Latency

Summary

	min	max	min	max	min	max	Type
	81	81	0.810 us	0.810 us	81	81	none

Detail

Instance

N/A

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	1	-	-	-
Expression	-	-	0	77	-
FIFO	-	-	-	-	-
Instance	2	-	188	242	-
Memory	0	-	36	5	0
Multiplexer	-	-	-	113	-
Register	-	-	147	-	-
Total	2	1	371	437	0
Available	280	220	106400	53200	0
Utilization (%)	~0	~0	~0	~0	0

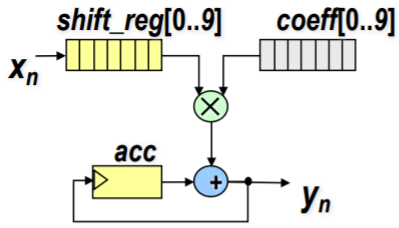
Detail

Instance

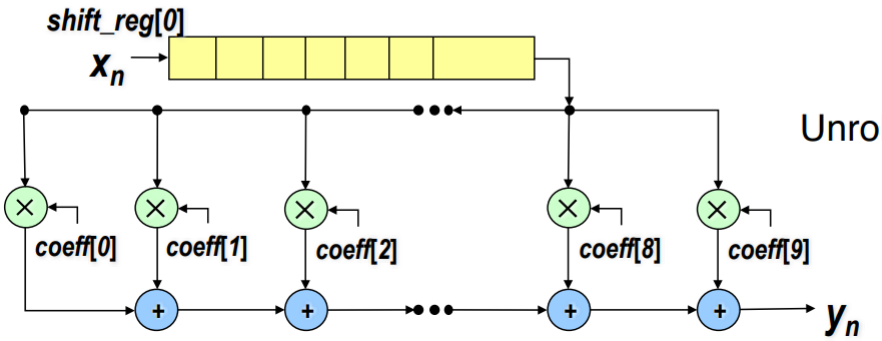
Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
fir_filter_AXILiteS_s_axi_U	fir_filter_AXILiteS_s_axi	2	0	188	242	0
Total	1	2	0	188	242	0

- **Solution 2 – Unroll Loops**

  Architecture after Unrolling



Default



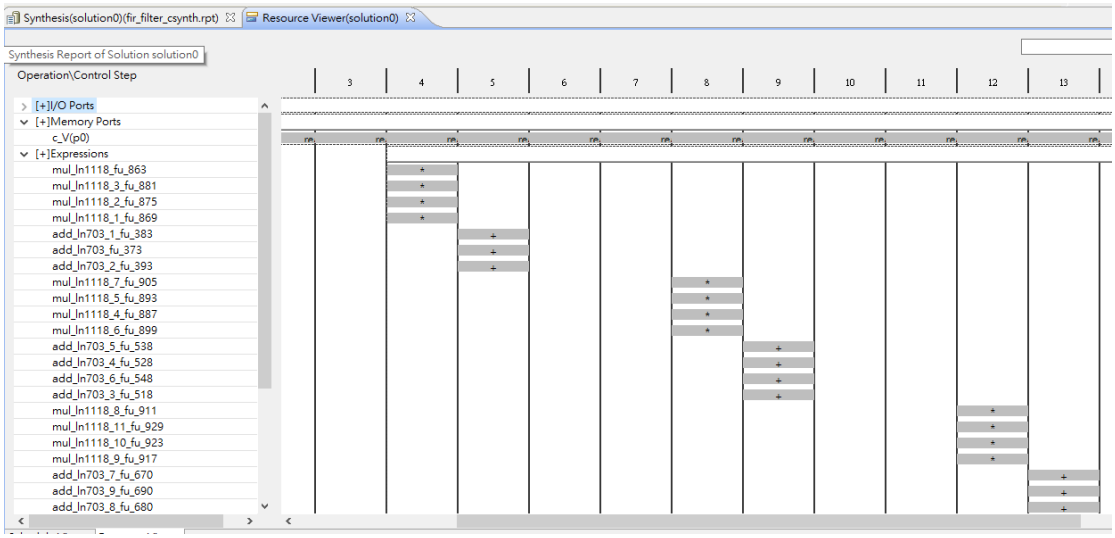
Unrolled

```
14    Shift_Accum_Loop: for (int i=N-1;i>=0;i--)
15    {
16  #pragma HLS LOOP_TRIPCOUNT min=1 max=16 avg=8
17
18      if (i==0)
19      {
20        //acc+=x*c[0];
21        shift_reg[0]=x;
22      }
23      else
24      {// shift the registers
25       shift_reg[i]=shift_reg[i-1];
26       //acc+=shift_reg[i]*c[i];
27      }
28      mult = shift_reg[i]*c[i];
29      acc  = acc + mult;
30
31    }
```

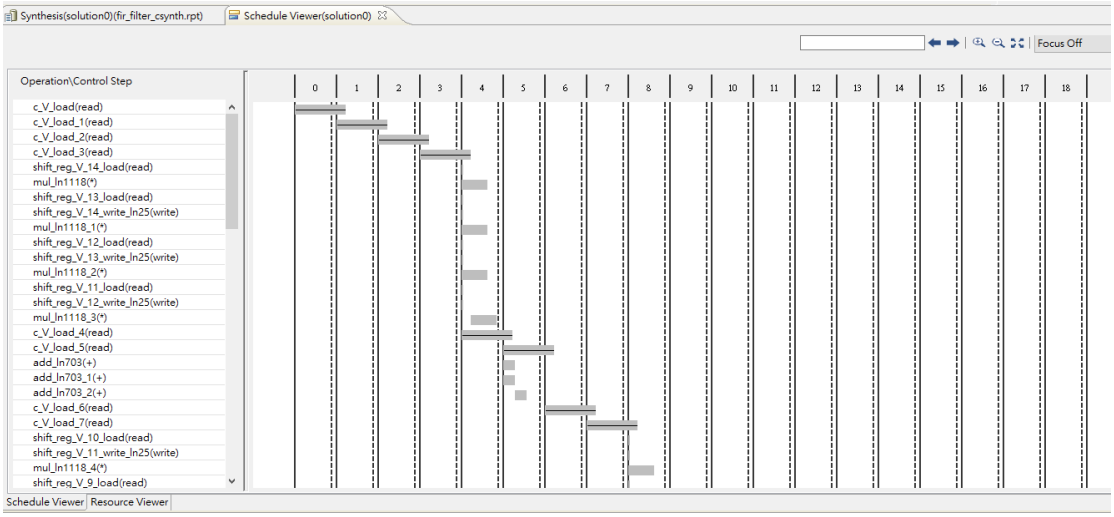
```
// unrolled shift registers
shift_reg[9] = shift_reg[8];
shift_reg[8] = shift_reg[7];
shift_reg[7] = shift_reg[6];
…
shift_reg[1] = shift_reg[0];
```

```
// unrolled multiply-accumulate
acc += shift_reg[0] * coeff[0];
acc += shift_reg[1] * coeff[1];
acc += shift_reg[2] * coeff[2];
…
acc += shift_reg[9] * coeff[9];
```

The adders and multipliers have multiple copies.



Loops is unrolled



The latency is reduced to 16+2, but the number of DSP increases

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.716	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
18	18	18	18	none

Detail

Instance

N/A

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	16	-	-	-
Expression	-	-	0	731	-
FIFO	-	-	-	-	-
Instance	2	-	188	242	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	182	-
Register	-	-	1073	-	-
Total	2	16	1261	1155	0
Available	280	220	106400	53200	0
Utilization (%)	~0	7	1	2	0

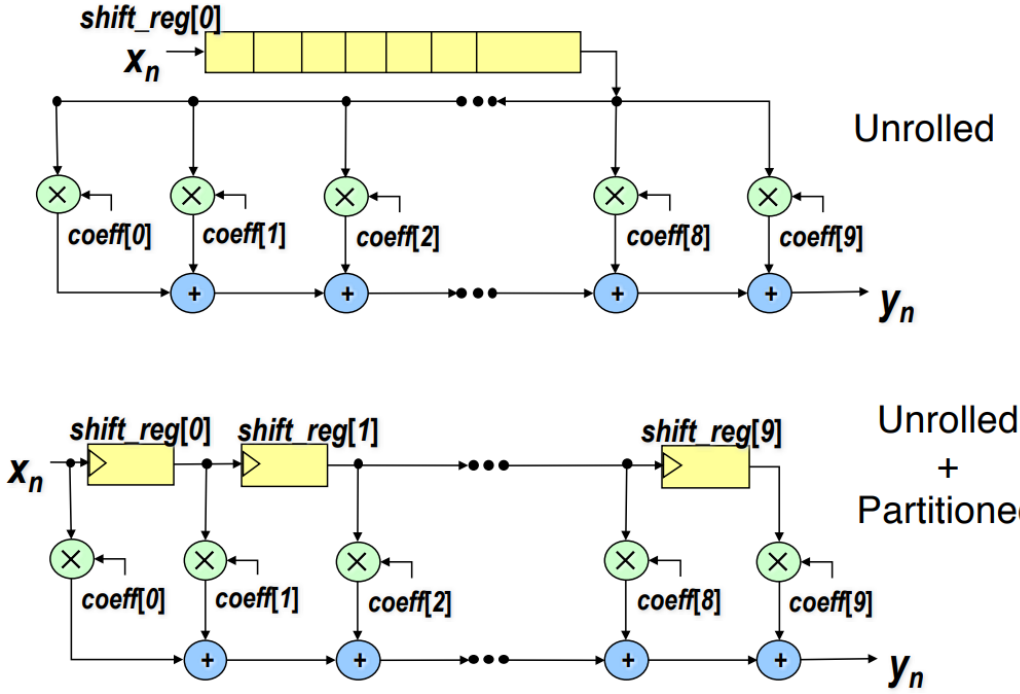
Detail

Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
fir_filter_AXILiteS_s_axi_U	fir_filter_AXILiteS_s_axi	2	0	188	242	0
Total	1	2	0	188	242	0

➢ **Solution 3– Unroll Loops + Array partitions on shift_regs**

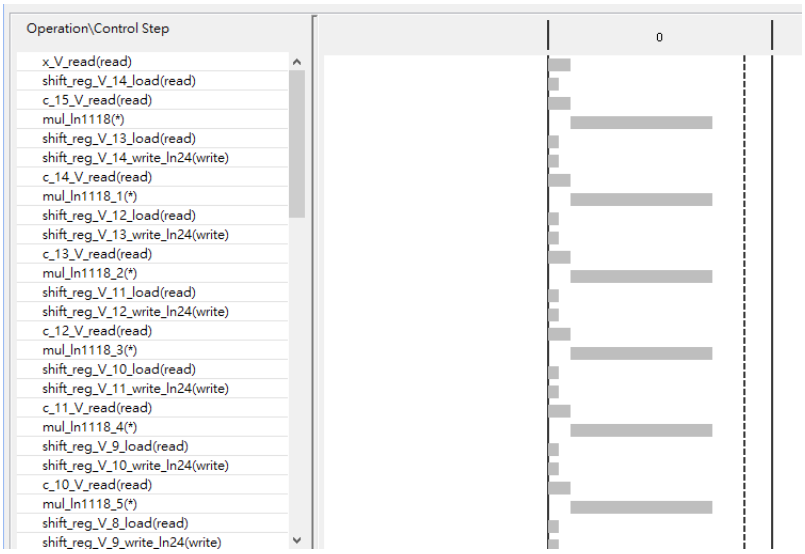
Architecture after Partitioning

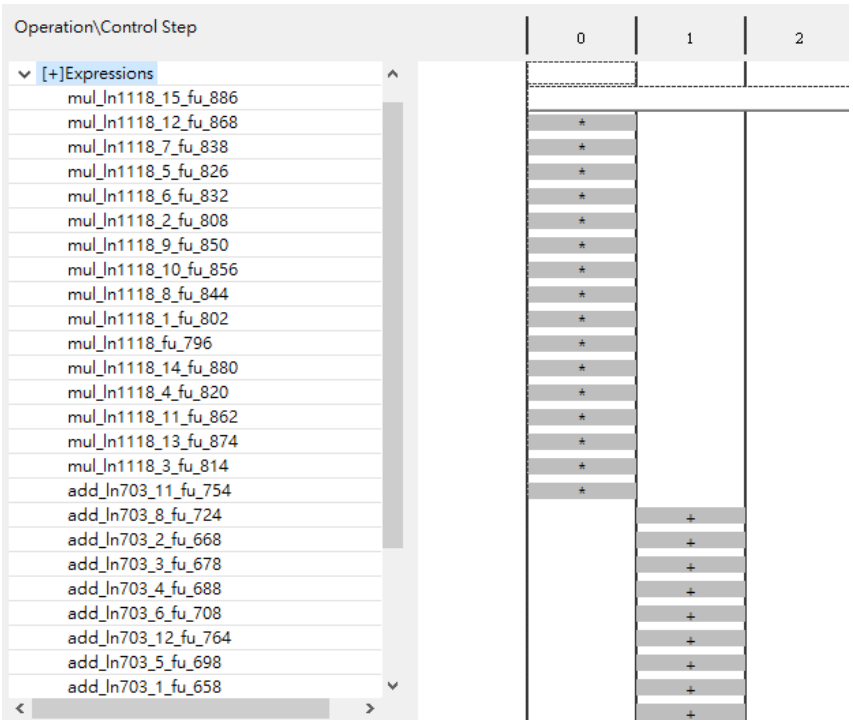


Set directives

```
c
% HLS ARRAY_PARTITION variable=c complete dim=1
shift_reg
% HLS ARRAY_PARTITION variable=shift_reg complete dim=1
```

NOTICE: both c and shift_reg must set directives





The latency reduce to 2

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.716	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2	2	2	2	none

Detail

Instance

Loop

Utilization Estimates

Summary

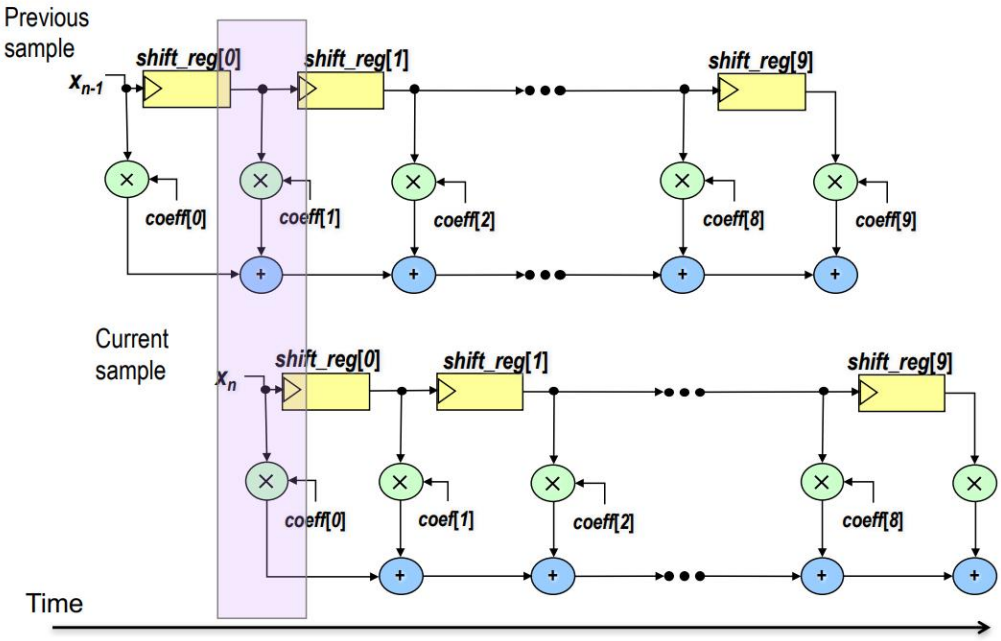
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	16	-	-	-
Expression	-	-	0	731	-
FIFO	-	-	-	-	-
Instance	0	-	492	748	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	21	-
Register	-	-	927	-	-
Total	0	16	1419	1500	0
Available	280	220	106400	53200	0
Utilization (%)	0	7	1	2	0

Detail

Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
fir_filter_AXILiteS_s_axi_U	fir_filter_AXILiteS_s_axi	0	0	492	748	0
Total	1	0	0	492	748	0

➢ Solution 4 – Pipeline Loop




```
14      Shift_Accum_Loop: for (int i=N-1;i>=0;i--)
15      {
16  #pragma HLS LOOP_TRIPCOUNT min=1 max=16 avg=8
17  #pragma HLS pipeline II=1
18          if (i==0)
```

The latency reduce to 2 due to pipeline pragma

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.716	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2	2	1	1	function

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	16	-	-	-
Expression	-	-	0	735	-
FIFO	-	-	-	-	-
Instance	0	-	492	748	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	927	-	-
Total	0	16	1419	1483	0
Available	280	220	106400	53200	0
Utilization (%)	0	7	1	2	0

Detail

Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
fir_filter_AXILiteS_s_axi_U	fir_filter_AXILiteS_s_axi	0	0	492	748	0
Total	1	0	0	492	748	0

C-sim

```
output[  58]=   4.46709       reference[  58]=   4.46725
output[  59]=   4.02063       reference[  59]=   4.02077
output[  60]=   4.73773       reference[  60]=   4.73785
output[  61]=   4.09267       reference[  61]=   4.09282
output[  62]=   4.45626       reference[  62]=   4.45640
output[  63]=   3.64550       reference[  63]=   3.64564
TOTAL ERROR =0.131182

TEST PASSED!
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

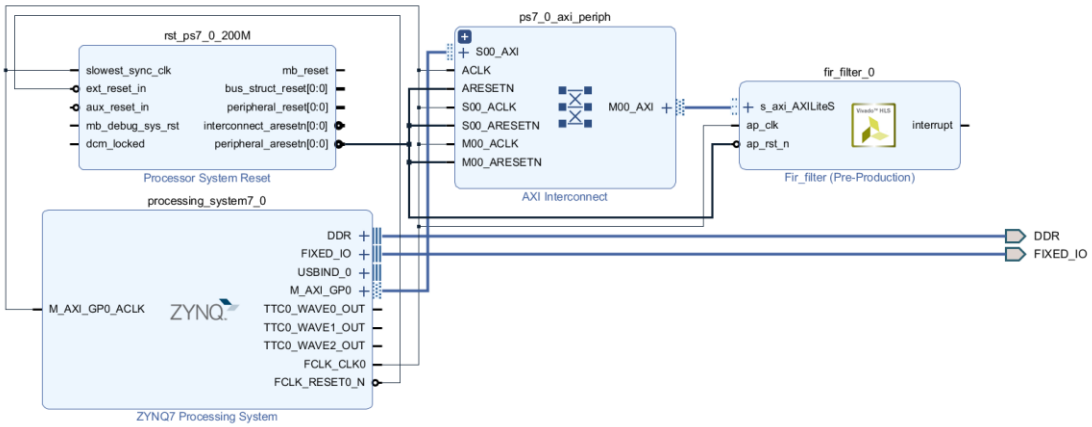
Co-sim

```
output[  61]=   4.09267       reference[  61]=   4.09282
output[  62]=   4.45626       reference[  62]=   4.45640
output[  63]=   3.64550       reference[  63]=   3.64564
TOTAL ERROR =0.131182

TEST PASSED!
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
Finished C/RTL cosimulation.
```

- **System level bring-up (Pynq or U50)**

Block diagram



Fixed point data type is used in my design, so I transfer fixed point to integer first and send it by s_axilite interface. After receiving return result, we need to transfer back fixed point and compare with reference.

```
16   #include "ap_fixed.h"
17
18   typedef ap_fixed<18,2>  coef_t;
19   typedef ap_fixed<48,12> out_data_t;
20   typedef ap_fixed<18,2>  inp_data_t;
21   typedef ap_fixed<48,12> acc_t;
```

```
3   out_data_t fir_filter (inp_data_t x,   coef_t c[N])
4   {
5     #pragma HLS INTERFACE s_axilite port = x
6     #pragma HLS INTERFACE s_axilite port = c
7     #pragma HLS INTERFACE s_axilite port=return
```

Host program

```
fiSamples.close()
numTaps = 16
f18Taps = allocate(shape=(numTaps,), dtype=np.int32)

f18Taps = [0.180617, 0.045051, 0.723173, 0.347438, 0.660617, 0.383869, 0.627347, 0.021650, 0.910570,
0.800559, 0.745847, 0.813113, 0.383306, 0.617279, 0.575495, 0.530052]
new_f18Taps = [math.floor(i * 2**16) for i in f18Taps]

timeKernelStart = time()
#COEF

output = allocate(shape=(numSamples,), dtype=np.float64)

for i in range(numSamples):
    ipFIRF18.write(0x1c, int(temp0[i]))
    for j in range(numTaps): # 0~7
        ipFIRF18.write(0x24 + j * 8, new_f18Taps[j])
    ipFIRF18.write(0x00, 0x1)
    a = ipFIRF18.read(0x10)
    b = ipFIRF18.read(0x14)
    output[i] = (b*2**32+a)*2**(-36)
    while (ipFIRF18.read(0x00) & 0x4) == 0x0:
        continue
tot_diff = 0
for i in range(numSamples):
    print("output = %.5f"%output[i], " ref = %.5f"%reference[i])
    diff = abs(output[i]-reference[i])
    tot_diff += diff
if (tot_diff < 1.0):
    print("TEST PASSED")
else:
    print("TEST FAILED")
timeKernelEnd = time()

print("============================")
print("Exit process")
```

```
output = 4.25798  ref = 4.25811
output = 3.46025  ref = 3.46037
output = 4.13056  ref = 4.13068
output = 3.81068  ref = 3.81079
output = 4.46850  ref = 4.46862
TEST PASSED
============================
Exit process
```

- **Github submission**