tags: MSOC

MSOC RSE Report

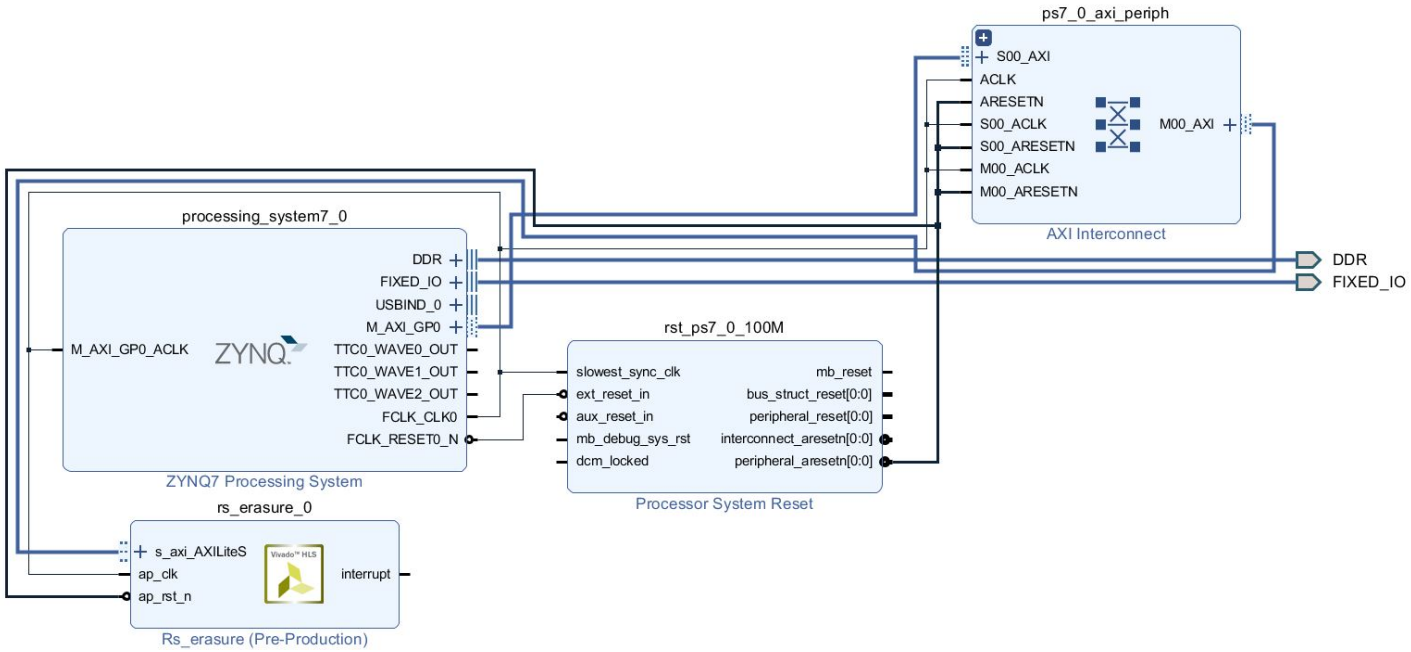
Objective

After completing this lab, we will be able to learn:

- [Additional] Analyze the latency and hardware cost by implementing the operation in different structure
- [Additional] Further optimize the baseline hls solution (pipeline, array partition, code restructuring...)
- [Additional] Use Vivado-HLS API to generate a Reed Solomon Erasure RTL IP with AXI_M interface from C code.
- [Additional] Build a PYNQ Host program and verify the design functionality on FPGA.

System diagram

This lab intends to develop a **Reed Solomon Erasurer** through the AXI_M & AXILite interface.



Interface

This design utilize the **AXILite interface** to transfer the input and output data. The following table shows the nnio of this kernel.

Addr. (+Base addr.)	bit	Usage
0x00	0	ap_start
0x00	1	ap_done
0x00	2	ap_idle
0x00	3	ap_ready
0x10-0x28		char array c
0x30-0x88		char array d
0x90		survival_pattern
0x98		codeidx

Notice that this corresponds to the interface of solution 4.

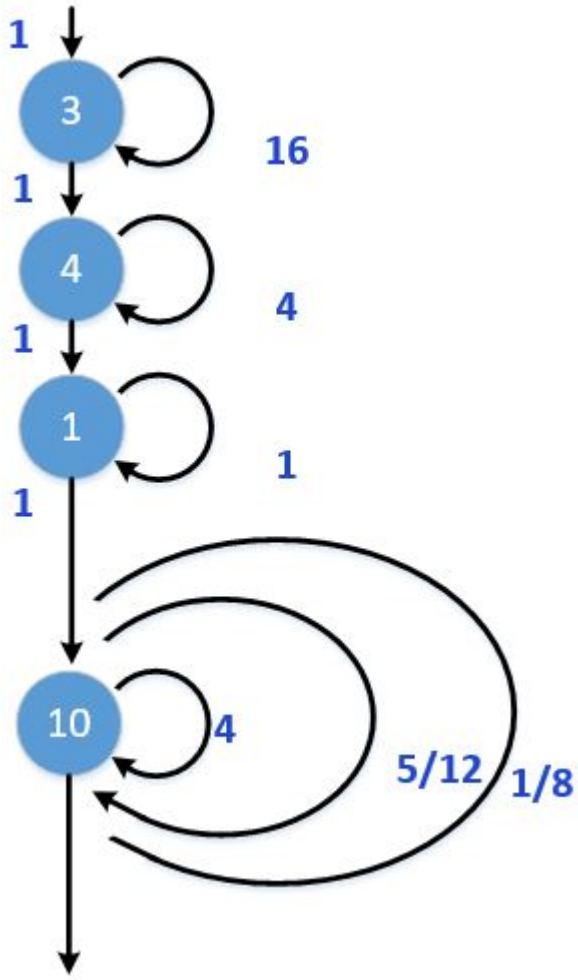
HLS implementation & optimization

What we have optimized will be marked **Bold**.

Solution 1 (Original design)

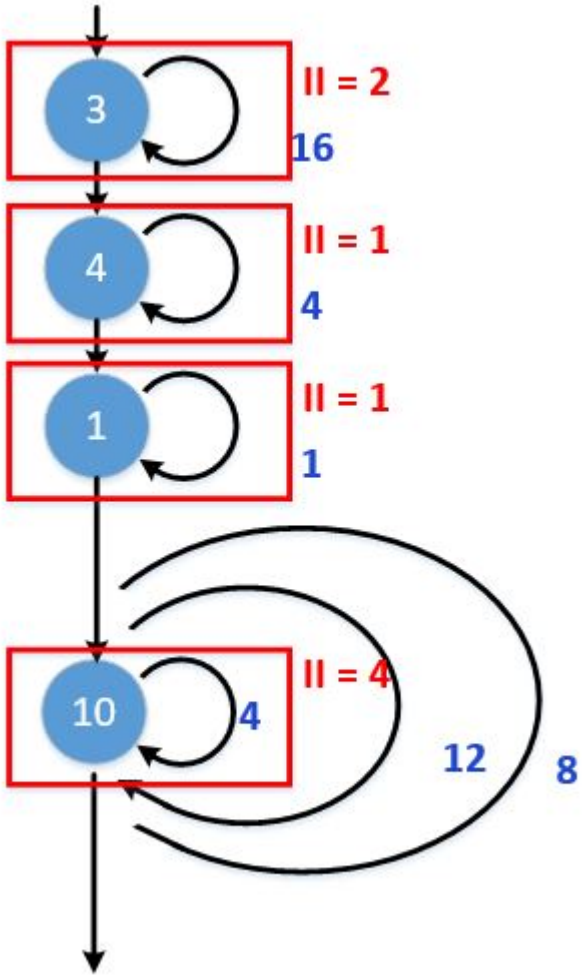
The flow digram is shown in the figure below (the delay of the loop is specify in the node, and the overhead of the loop is specify in the front of '/'). We can easily calculate the total latency **3x16+1+4x48+1+4x1+1+(((10x4+6x2)x12+1x2)x8)+10=5265** (10 is the overhead for the

interface processing). **The additional one for the latency is due to simultanenous read/write issues** (specify in Lab2's report).



Solution 2 (Naive Pipeline)

The simplest way to speed up without increasing too much resource is to pipeline all the inner loops. And the resulting pipeline II is shown in the flow chart below (The pipelined loop is highlighted by read blocks).



Even this naive pipeline idea can reduce the latency from 5264 cycles to 1650 cycles without increasing too much hardware resource.

Solution 3 (Code restructuring to further reduce II)

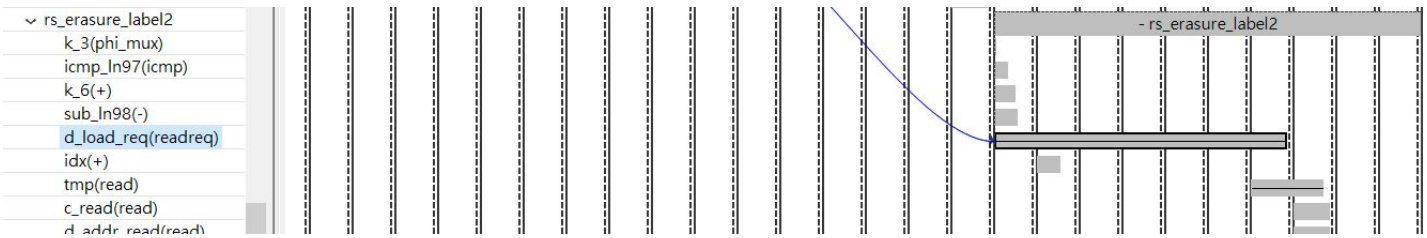
1.

The II=2 loop can't be further reduce since the read process from a "size 64" array requires latency larger than one cycle.

2.

The second part is to reduce the II=4 Loop.

We find that this is cause by the bandwidth of reading the data(see figure below).



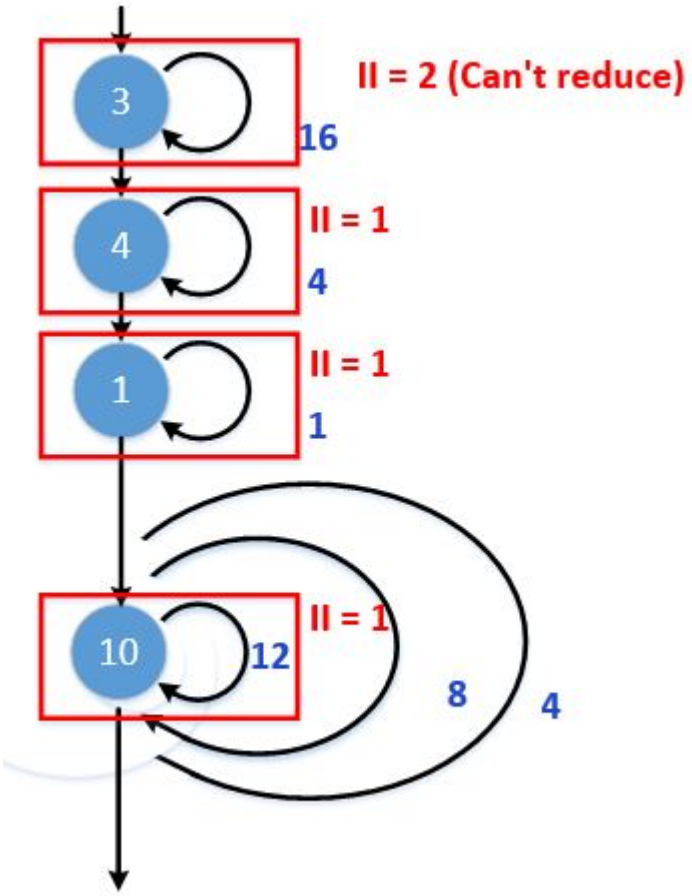
However, this step is redundant. The code really want to do is to accumulate the values into the array. We can instead auumulate by a temp variable “temp_read” and store the result after the full accumulation process. The following figure shows what we have done (Notice that the order of the loop is switched in order to fulfill our idea).

```
for(i=0; i<GF_ORDER; i++){
        // loop for all 8 data
        for(j=0; j<NUM_TAPS; j++)
            // loop for all 4 code symbols
            rs_erasure_label2:for(k=0; k<NUM_EQUATION; k++){
                unsigned char idx = k*NUM_TAPS+j;
                unsigned char  tmp = r[idx];
                // update c
                c[k] = c[k] ^  ( ((d[j]>>i)&1)? tmp : 0 );
                // update ram
                r[idx] = ((tmp>>7)&1) ?  ((tmp<<1)^gf_poly) : (tmp<<1);

            }
}
```

Loop order switching
&
Accum rewrite

```
for(k=0; k<NUM_EQUATION; k++){
    char tmp_read = 0;
    for(i=0; i<GF_ORDER; i++){
        // loop for all 8 data
        rs_erasure_label2:for(j=0; j<NUM_TAPS; j++){
            // loop for all 4 code symbols
            unsigned char idx = k*NUM_TAPS+j;
            unsigned char  tmp = r[idx];
            // update c
            tmp_read = tmp_read ^  ( ((d[j]>>i)&1)? tmp : 0 );
            // update ram
            r[idx] = ((tmp>>7)&1) ?  ((tmp<<1)^gf_poly) : (tmp<<1);

        }
    }
    c[k] = tmp_read;
}
```

The final flow chart is as follows:



After our optimization, the cycles is further reduce from 1650 to 956 cycls without adding too much hardware resource.

Solution 4 (Fully Unroll & Array partition)

The last solution pipeline the whole function and array partition the input & outpu array, so all the operation in the for loop will be parallize as much as possible.
The resulting total latency can be drastically reduce to 42 cycles with hardware resource as a tradeoff.

We try to implement this solution on board (all the interface is set AXILite).

Comparison Table

The final latency table:

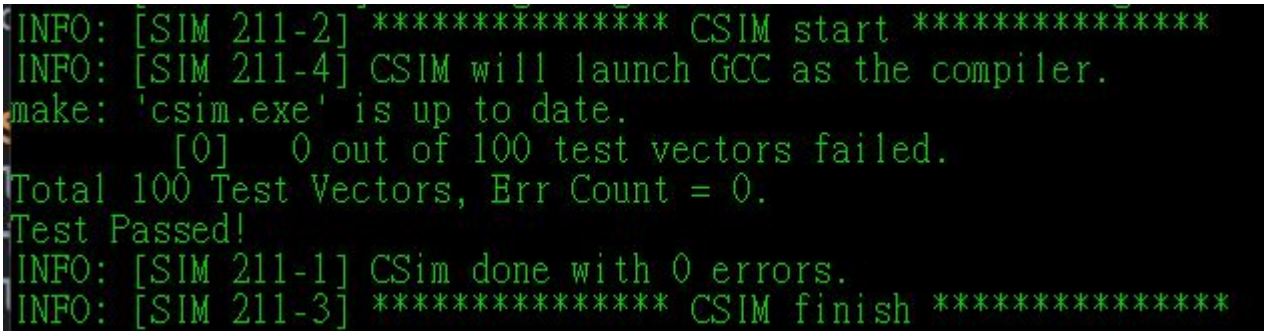
		solution1	solution2	solution3	solution4
Latency (cycles)	min	5264	1650	956	42
	max	5264	1650	956	42

The final utilization table:

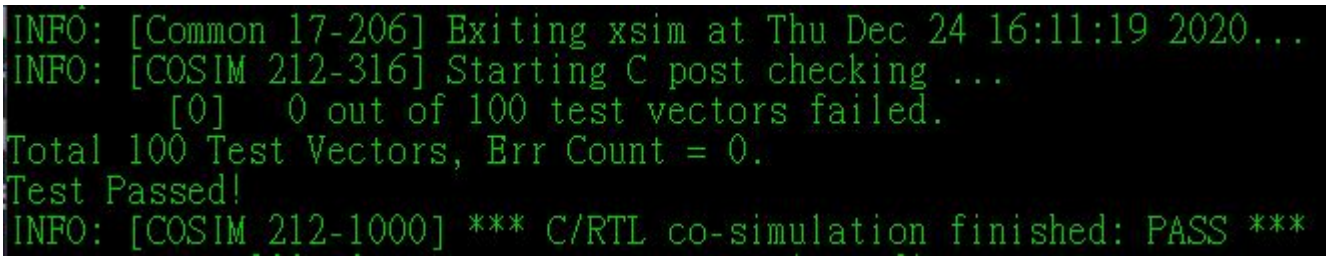
	solution1	solution2	solution3	solution4
BRAM_18K	132	132	132	96
DSP48E	0	0	0	0
FF	1578	1914	2066	12977
LUT	2390	2710	2729	20518
URAM	0	0	0	0

Csim and Cosim results

Csim:



Cosim:



PYNQ implementation

Our host program write 100 input datas streams into the kernel and compare all the output stream with the golen data. The interconnection and interface is shown in the beginning of the report.

```
151 127 121  44 | 151 127 121  44
199 221 240 239 | 199 221 240 239
205 131 226 179 | 205 131 226 179
 52 124 184 105 |  52 124 184 105
 62 140 143 177 |  62 140 143 177
 68 195 152 113 |  68 195 152 113
106  31 154 240 | 106  31 154 240
172  38   8 169 | 172  38   8 169
217 105 206 186 | 217 105 206 186
 93 114  56  30 |  93 114  56  30
 76 180  84 255 |  76 180  84 255
187 141  73  21 | 187 141  73  21
235 193 217 201 | 235 193 217 201
Total 100 Test Vectors, Err Count = 0.

Test Passed!

============================
Exit process
```