tags: `MSOC`

MSOC CORDIC-SQRT Report

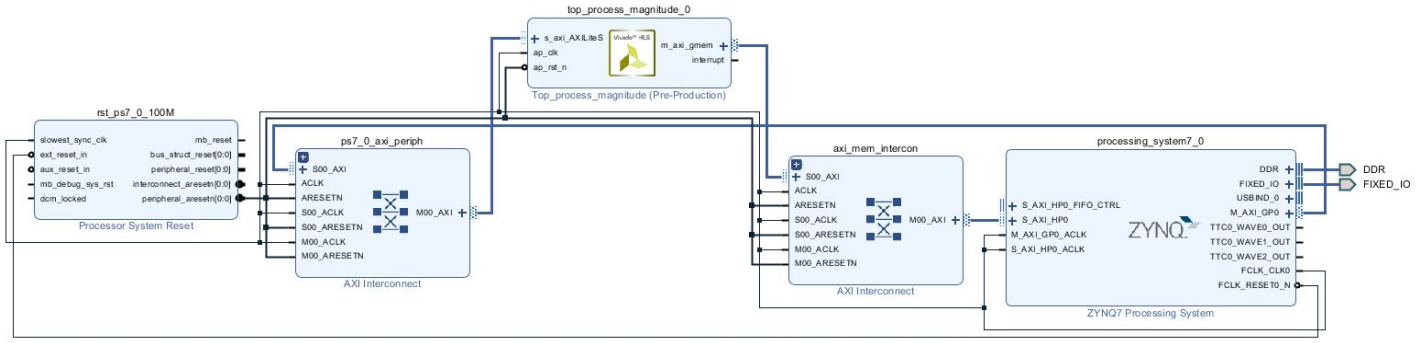
Objective

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

- Analyze the latency and hardware cost by implementing the operation in different datatype (Double, Float, Int)
- Analyze the latency and hardware cost by implementing the operation by CORDIC.
- [Additional] Further optimize the baseline hls solution.
- [Additional] Use Vivado-HLS API to generate a L2-norm Calculator 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 **L2-norm Calculator** through the AXI_M interface.



Interface

This design utilize the **AXIM interface** to transfer the input and output data. The following table shows the usage for some address.

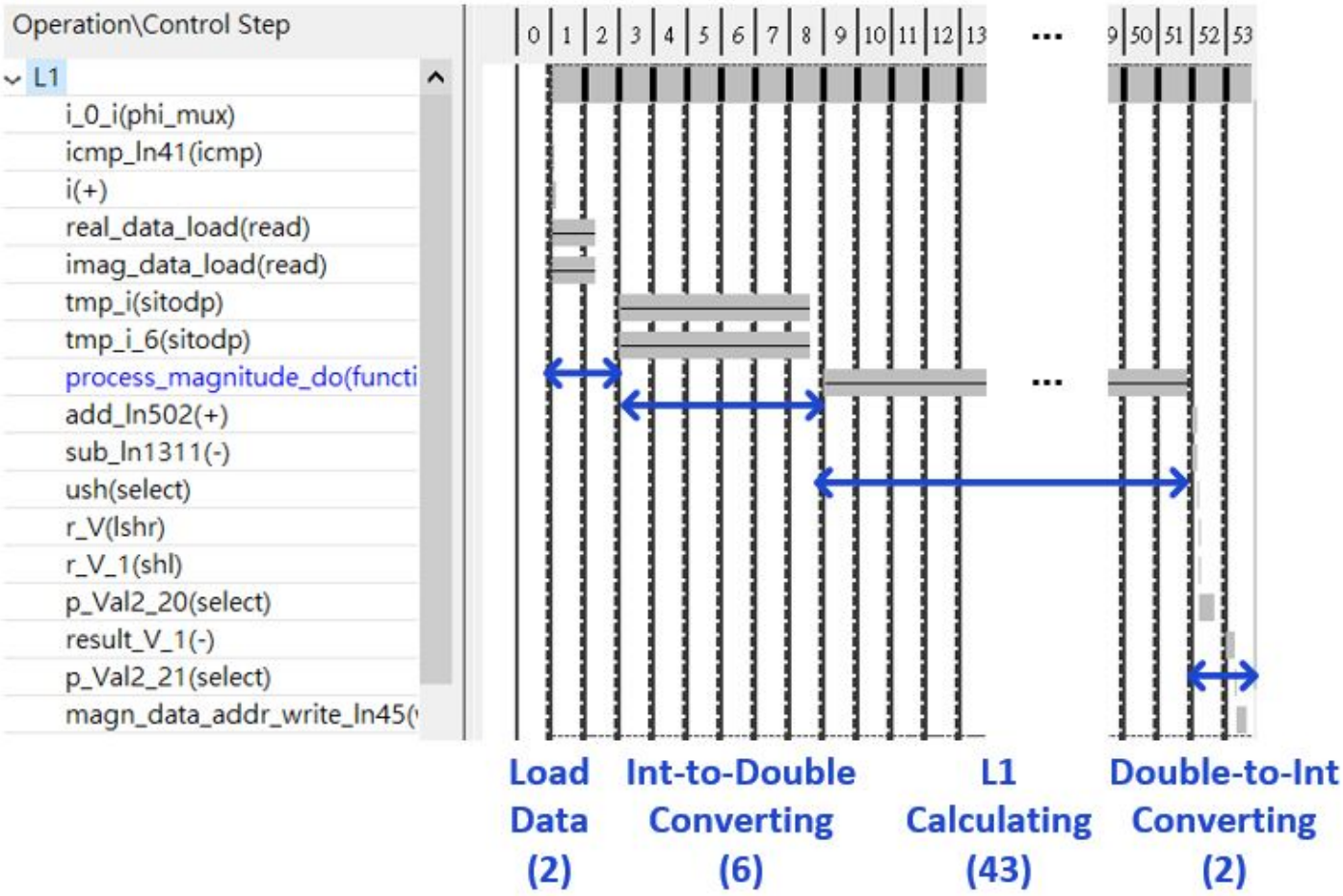
Addr. (+Base addr.)	bit	Usage
0x00	0	ap_start
0x00	1	ap_done
0x00	2	ap_idle
0x00	3	ap_ready
0x00	7	auto_restart
0x04	0	Global interrupt
0x10		real data addr.
0x18		imag data addr.
0x20		out data addr.
0x80		transfer length
0x40~0x7f		Coefficient

HLS implementation

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

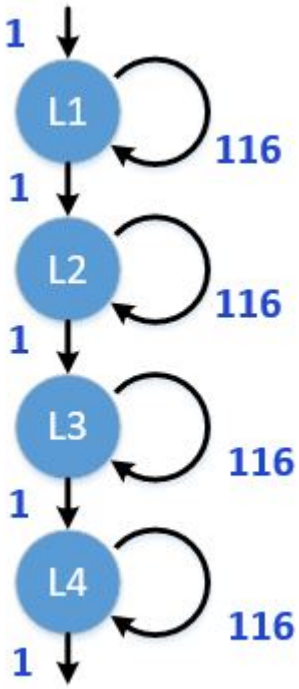
Solution 0 (64-bit double datatype L2 calculation)

The following figure shows the top-level timeline (We've also mark the function and latency for each part). Notice that since the orignal input data is of integer type, **an "integer-to-double" converter is required.**



Notice that the iteration latency is 2+6+43+2=53, the pipeline II is 1, and the loop trip count is 64, thus the latency for each loop is 53+64-1=116.

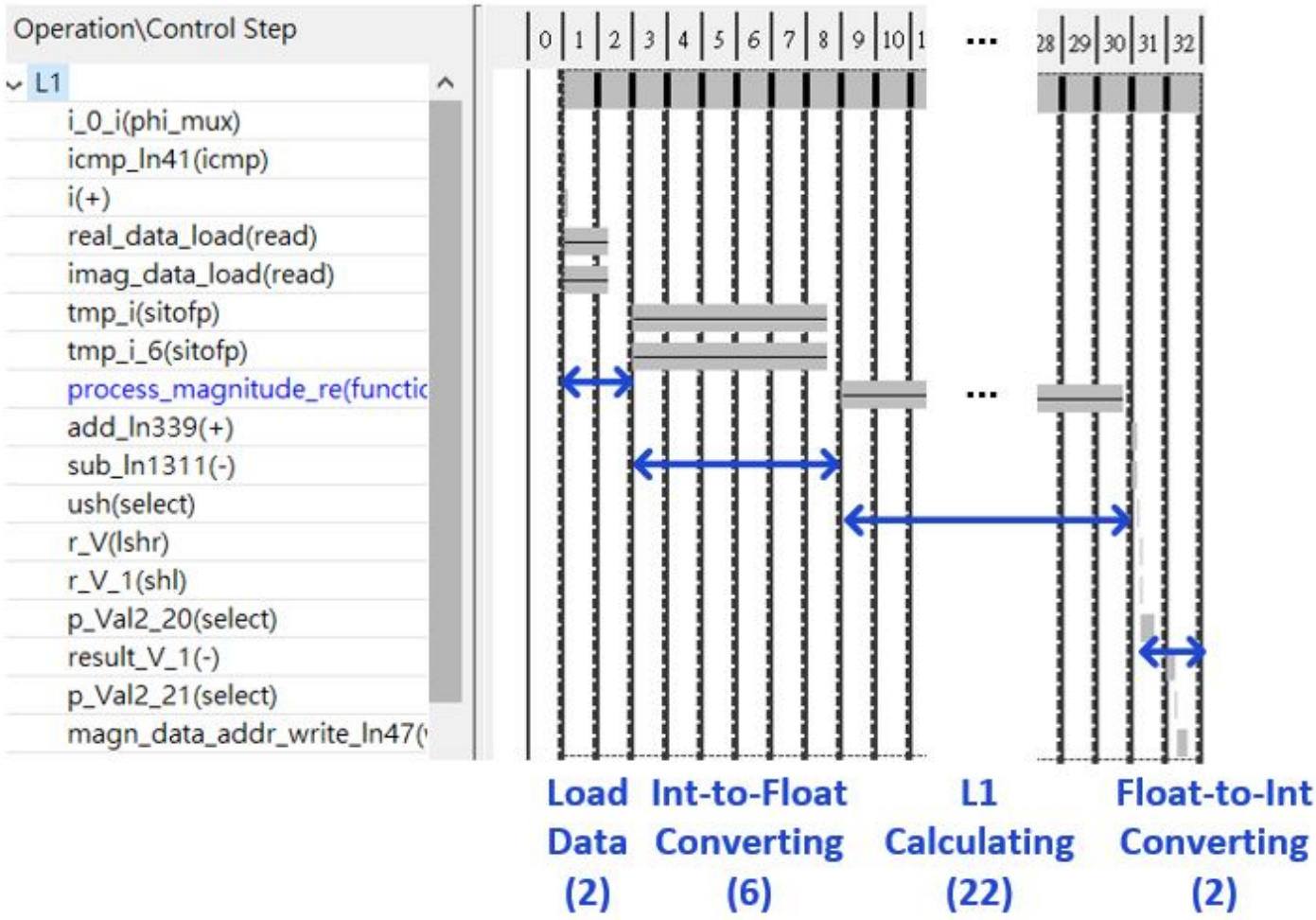
We can know that **the total latency is 116x4+4=468** (this can be more clear by the following flow chart). Actually, the latency must plus 5, an additional 1 is minus due to the simultaneous read/write operation issue (we have shown in report Lab2-1)



Notice that since L1, L2, L3, L4 do the same operation and is without dependency, we can further optimize it (shown in solution 5).

Solution 1 (32-bit floating point datatype L2 calculation)

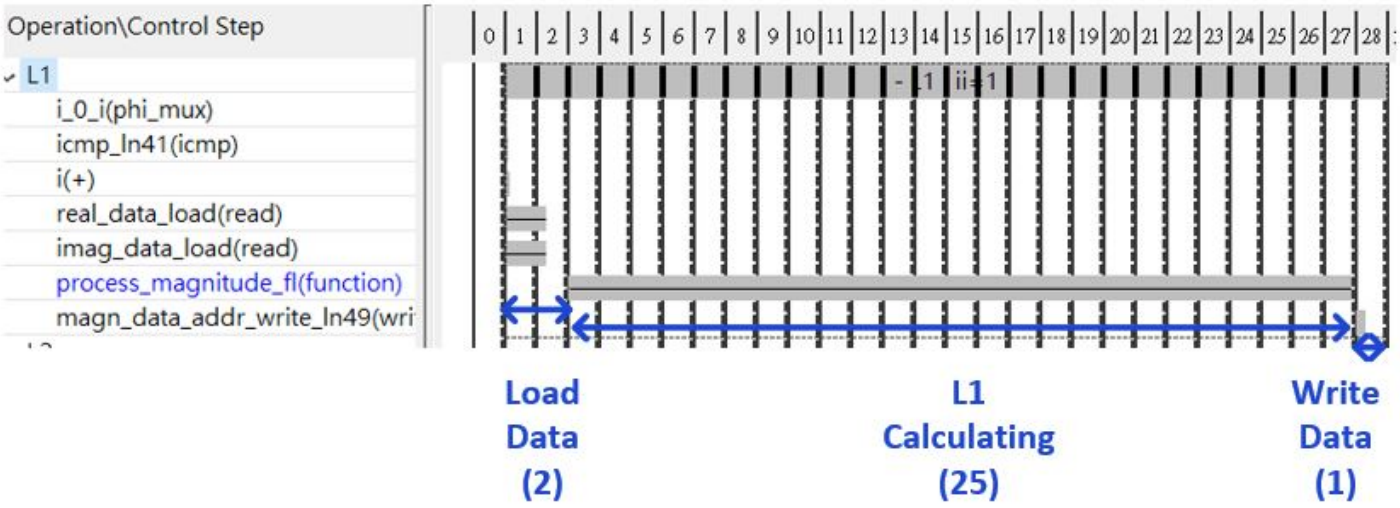
The following figure shows the top-level timeline. Similar to solution 0, **an "integer-to-float" converter is required.**



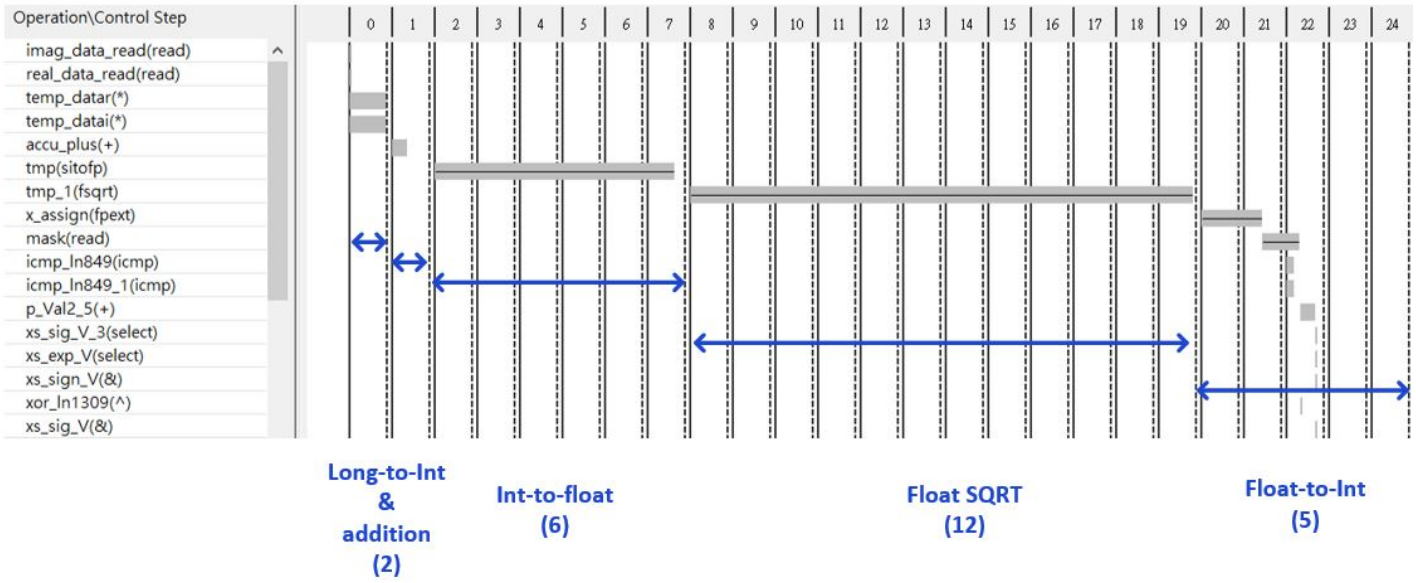
Similar to Solution 1, the iteration latency is 2+6+22+2=32, the pipeline II is 1, and the loop trip count is 64, the latency for each loop is 32+64-1=95, and **the total latency is 95x4+4=384**.

Solution 2 (integer datatype L2 calculation)

The following figure shows the top-level timeline. Notice that since the original input data is of integer type, **the datatype converter is not required in the top level design.**



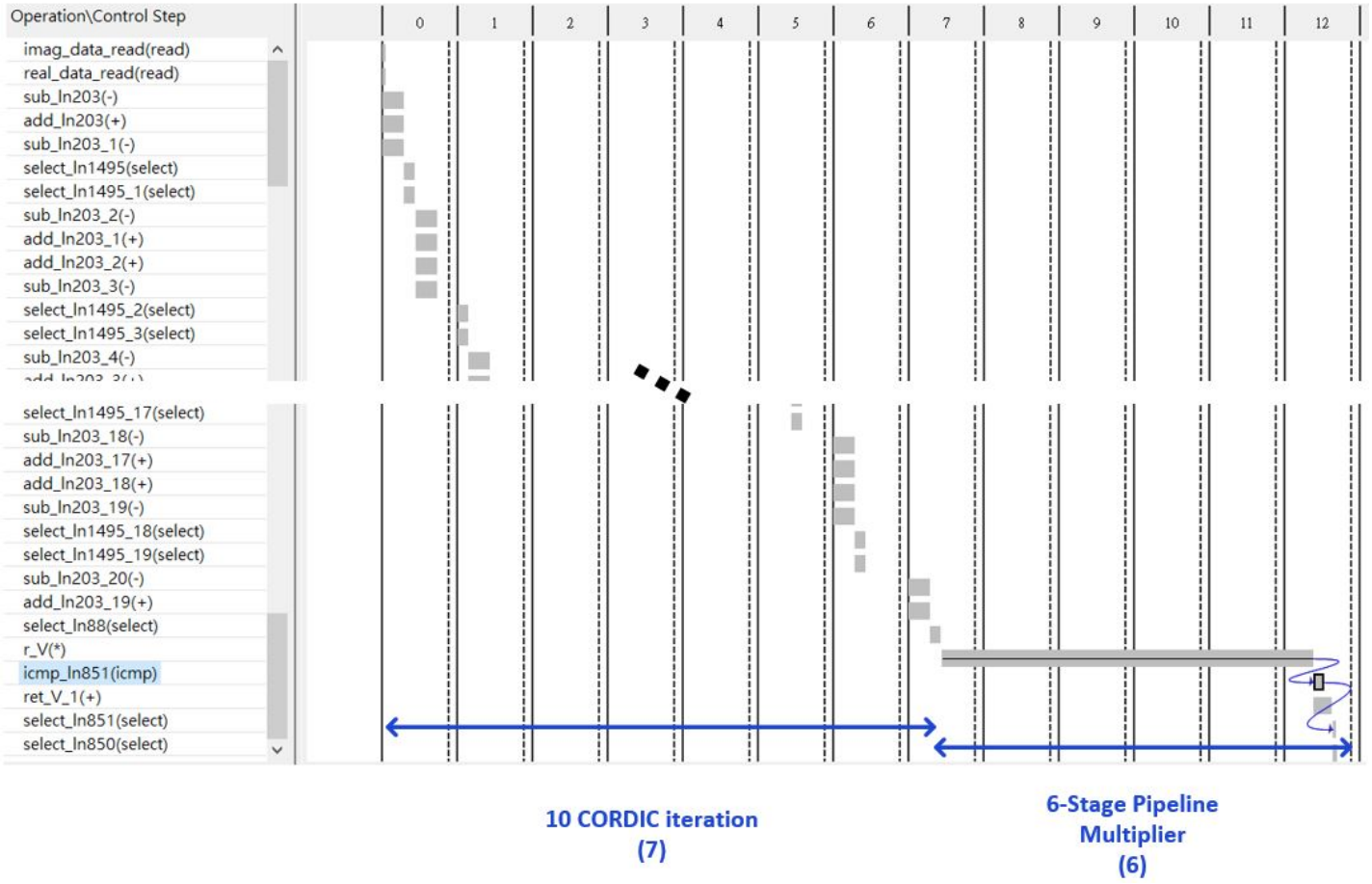
However, the sqrt calculation is still done by floating point, so actually the floating-and-int converter is hidden in the “process_magnitude” submodule.



Similar to the case in solution 1 & 2, **the total latency is 4x(2+25+1+64-1)+4=368.**

Solution 3 (L2 calculation by (40,32) fixed point CORDIC)

The only difference between this solution and solution 2 is the latency of the inner submodule “process_magnitude”. The latency of the submodule is 13 in this case, therefore **the total latency is 4x(1+14+2+64-1)+4=324.**



Notice that **since the "#pragma HLS RESOURCE variable=x2 core=MUL6S" is added, the compensation multiplier is synthesize by a 6-stage multiplier.**

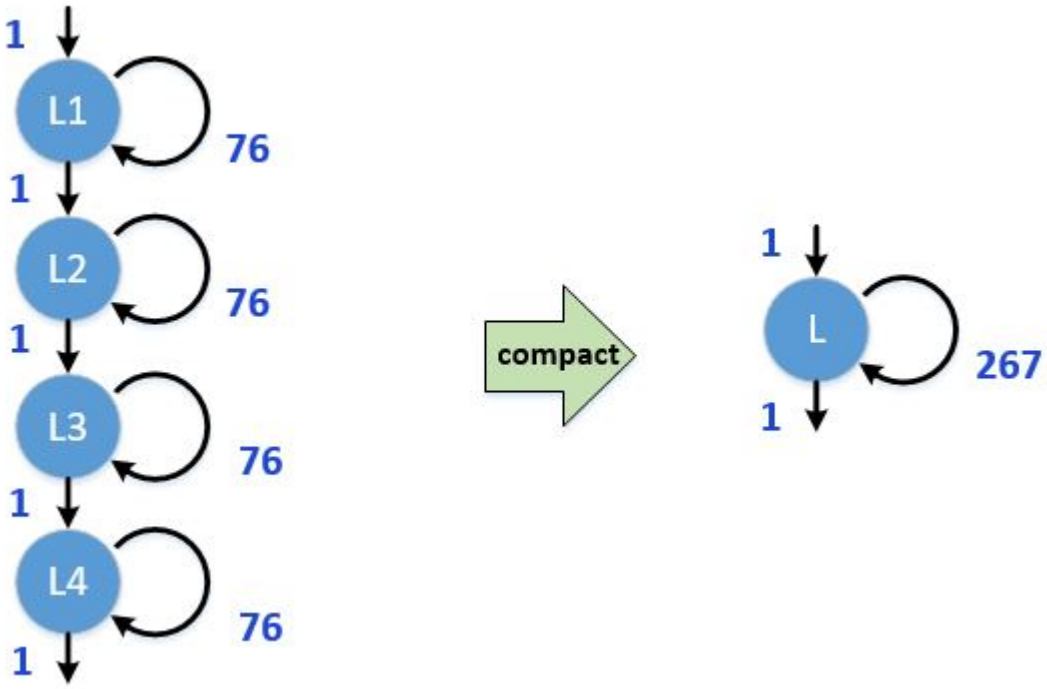
An observation is that **Solution 2 & 3 do sqrt under rougly the same latency, but the prior has much overhead on datatype conversion.**

Solution 4 (L2 calculation by (24,18) fixed point CORDIC)

We find that the word length of fixed point datatype can be further reduce to (24,18) without lossing to much accuracy. By doing this, we can further reduce the latency of the submodule to 10 (due to lower latency of add/sub in CORDIC); therefore, **the total latency is 4x(1+10+2+64-1)+4=308.**

Solution 5 (Solution 4 + Loop compact)

Actulally the 4 loop can be placed more compact (see figure below).



This will reduce the overall latency to 10+256-1+4=269. Moreover, the hardware cost can reduce since in original solution, 4-loops requires 4x registers and multiplexers.

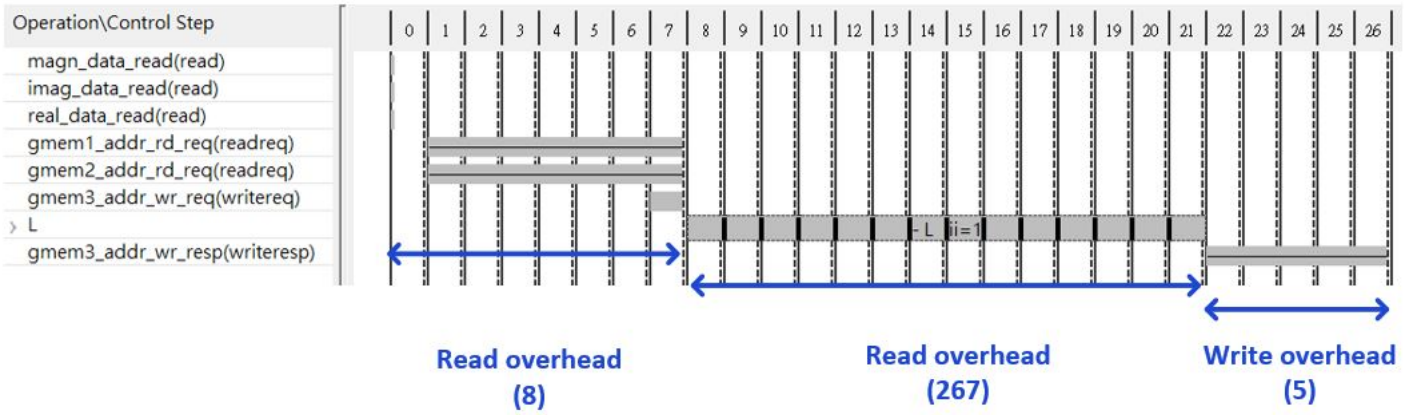
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	32	-
FIFO	-	-	-	-	-
Instance	-	2	471	1849	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	48	-
Register	0	-	199	64	-
Total	0	2	670	1993	0
Available	280	220	106400	53200	0
Utilization (%)	0	~0	~0	3	0

compact

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	124	-
FIFO	-	-	-	-	-
Instance	-	2	490	1858	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	263	-
Register	0	-	677	256	-
Total	0	2	1167	2501	0
Available	280	220	106400	53200	0
Utilization (%)	0	~0	1	4	0

Solution 6 (Solution 5 + AXI_M interface)

The following figure shows the timeline. We can see that extra 13 cycles is introduced by input/output overhead, so **the total latency is 281.**



Also, **the resource will increase due to the large size array decoder.**

Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
grp_process_magnitude_co_fu_167	process_magnitude_co	0	2	518	1858	0
top_process_magnitude_AXILiteS_s_axi_U	top_process_magnitude_AXILiteS_s_axi	0	0	150	232	0
top_process_magnitude_gmem1_m_axi_U	top_process_magnitude_gmem1_m_axi	2	0	512	580	0
top_process_magnitude_gmem2_m_axi_U	top_process_magnitude_gmem2_m_axi	2	0	512	580	0
top_process_magnitude_gmem3_m_axi_U	top_process_magnitude_gmem3_m_axi	2	0	512	580	0
Total		5	6	2	2204	3830	0

Comparison Table

The final latency table:

Latency

		solution0	solution1	solution2	solution3	solution4	solution5	solution6
Latency (cycles)	min	468	384	368	324	308	269	281
	max	468	384	368	324	308	269	281

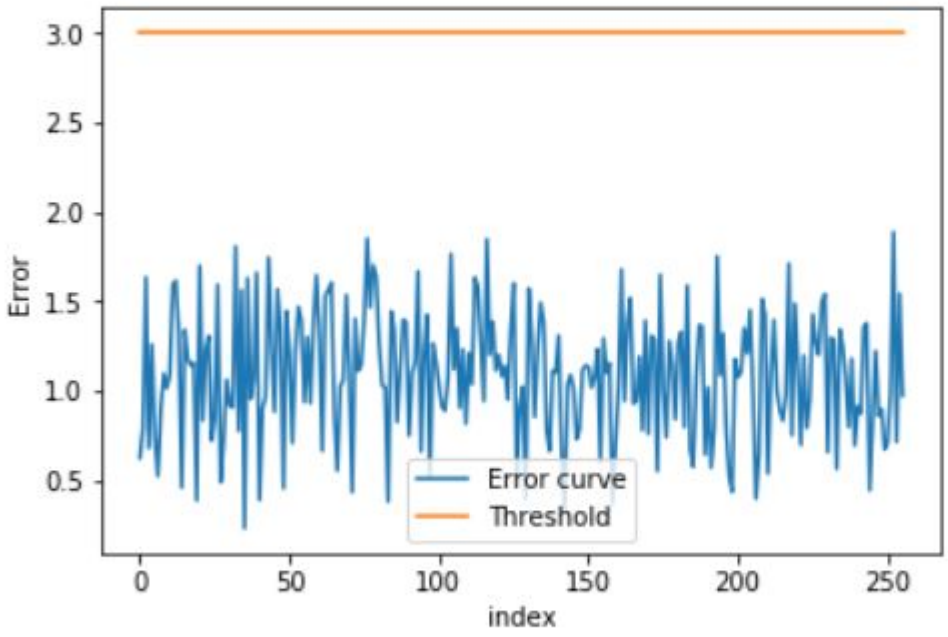
The final utilization table:

	solution0	solution1	solution2	solution3	solution4	solution5	solution6
BRAM_18K	0	0	2	0	0	0	6
DSP48E	25	8	8	5	2	2	2
FF	5309	2778	2377	2090	1223	726	2478
LUT	9787	5239	3360	3525	2501	1993	4056
URAM	0	0	0	0	0	0	0

PYNQ implementation

Our host program random generates 256 datas and plot the error curve between Double Floating and CORDIC<24,18>.

```
Entry: /usr/lib/python3/dist-packages/ipykernel_launcher.py
System argument(s): 3
Start of "/usr/lib/python3/dist-packages/ipykernel_launcher.py"
Kernel execution time: 0.00022745132446289062 s
```




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

Encountered bugs

- Since the bitwidth of (packed) data on axi master must be power of 2, **we must modified the input and output data format to 32-bit** (since 16-bit is not enough).
- **Add "-ldflags {{-Wl,–stack=268435456}}"** in Csim and Cosim to increase the stack size; otherwise, the csim and cosim won't pass!
- **Bundle real and imag data on different gmem; otherwise, it will be default bundled on the same gmem, which will make II=2 and thus increase 2x total latency.**