FPGA Firmware Design and Verification for the ATLAS Liquid Argon Calorimeter Trigger Processor

Melissa Aguiar, Marcos Oliveira and Lucca Viccini

12/06/2024

Agenda

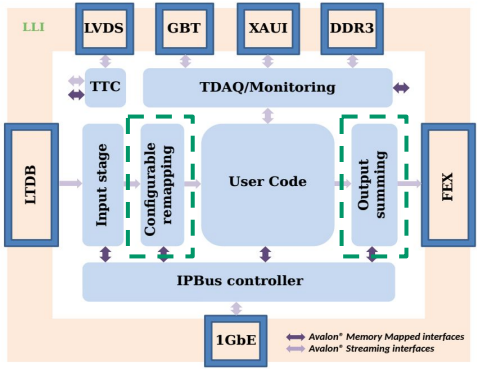
- Motivation
- Design flow
- Design examples
  - Partial-mesh switch matrix
  - Recurrent operations
  - Expensive operations
  - Clock retargeting
- Summary

Motivation

- It took ~8 years to develop the LAr backend firmware currently running at P1:
  - Logic implemented using time-division multiplexing (TDM);
  - Timing closure is very challenging and the area utilization is high.

- Exploration studies showed that a new architecture with parallel logic and design abstraction based on HLS can improve the overall design (provide lower logic usage, latency, better code readability, timing closure, design verification, see [1]).
  - The resource sharing is implemented automatically only when needed using HLS.

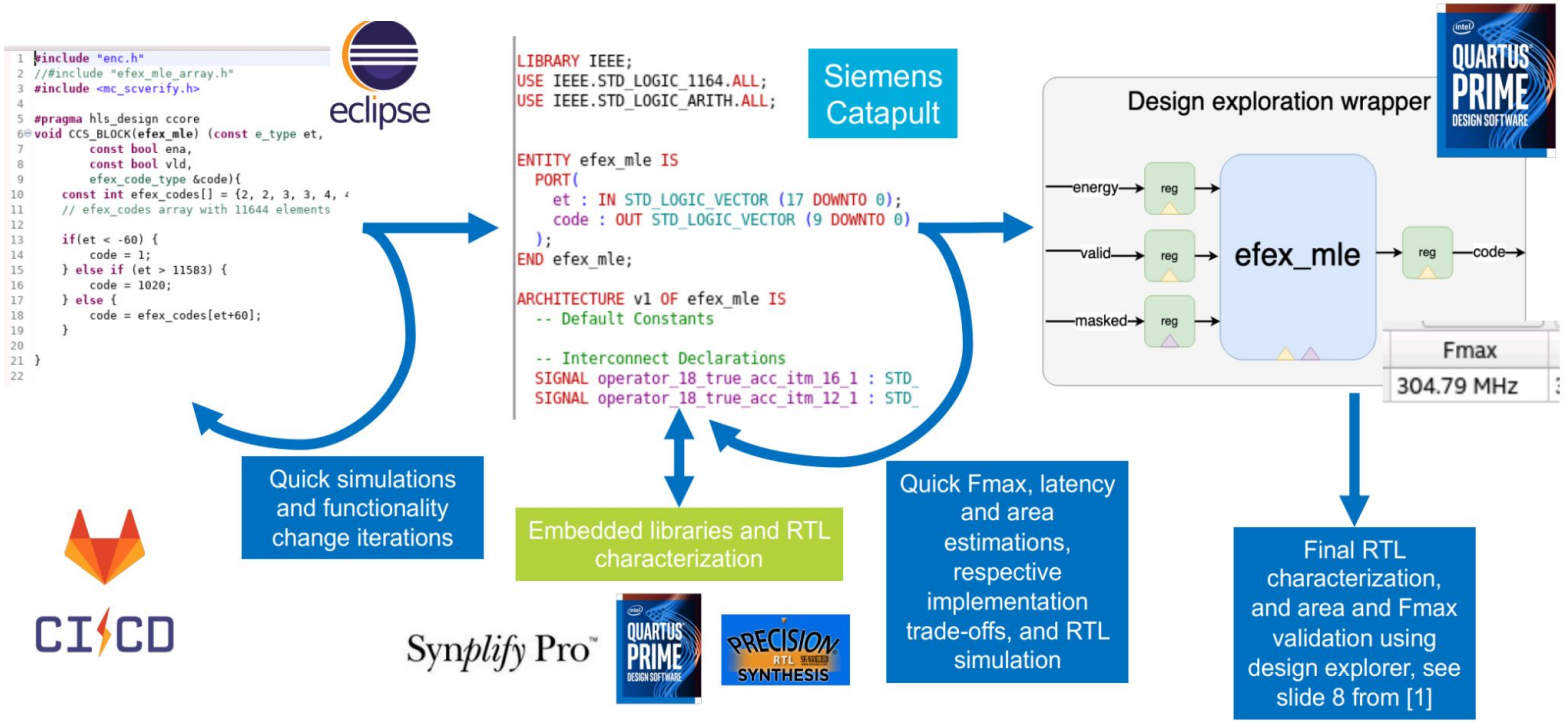
- We decided to take these studies further and redesign two blocks of the current firmware in view of:
  - Reducing total area utilization by routing data more efficiently → allowing to reduce the number of filtering blocks (high area);
  - Learn more about leveraging parallelization versus serialization for our use cases;
  - Test if HLS can be beneficial for future detector upgrades.



LATOME Block Diagram.

[1] Silva Oliveira, Marcos Vinicius. 2022. “LATOME Firmware in the Long-Term.” https://indico.cern.ch/event/1226981/#84-latomefirmware-in-the-long.

Overview of the LATOME HLS Design Flow



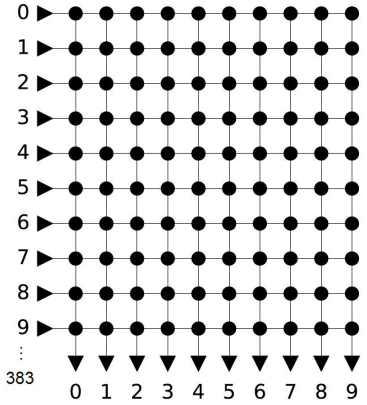
[1] Silva Oliveira, Marcos Vinícius. “LATOME Firmware in the Long Term.” 2023. https://indico.cern.ch/event/1260250/#84-latome-firmware-in-the-long.

Example A - Partial-mesh Switch Matrices

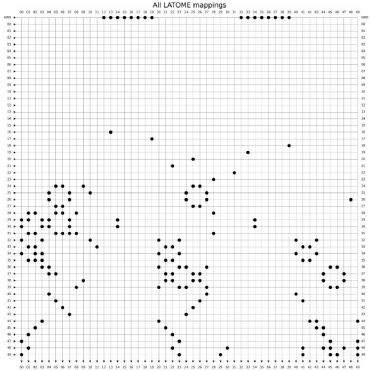
- Each board receives detector data:
  - 48 optical fibers → transfer data from 8 sensors using TDM.

- Original design kept unchanged the TDM interface and routed the data to the respective filtering blocks in multiple steps.

- The new architecture deserialize the input data and routes them to the filtering blocks in a single step implementation using a switch matrix.
  - The implementation of a full-mesh switch matrix requires 131% of the FPGA area;
  - We analysed the required routing for our 116 boards and implemented a sparse switch matrix that required only 4% of the FPGA area:
    - Comparable area to previous TDM solution;
    - Less than half of the latency.



full-mesh switch matrix example



partial-mesh (sparse) switch matrix example, for more examples see here

Example B - Recurrent Operations

- We need to repeat a multi-linear encoding operation 320 times.
- We write our HLS code without needing to know beforehand if and how resource sharing will be used.
  - We only write such operation in a loop and we can decide later if:
    i. Process all 320 inputs in parallel using dedicated blocks for each input;
    ii. Reusing some of encoding blocks to process multiple inputs using TDM.
- The area was reduced by 40% with reutilization at the price of nearly doubling the latency [1].
- Even in a late stage of the project, one can trade area by latency without changing source code.

Resource Sharing	Latency	Throughput	Slack (ns)	ALMs (%)
320 blocks	5	1	0.335	19,603 (4.5%)
64 blocks	9	5	0.257	11,673 (2.7%)
48 blocks	11	7	0.278	12,089 (2.8%)

Resource sharing options for the multi-linear encoding

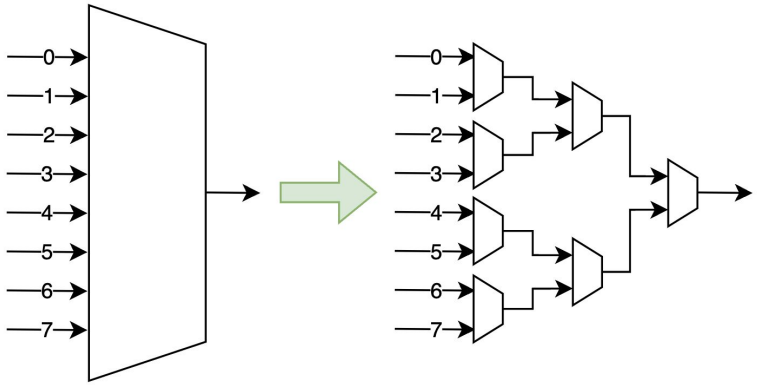
Area vs latency trade-off

[1] Aguiar, Melissa S. 2024. “LATOME HLS Implementation.” https://indico.cern.ch/event/1388391/#25-latome-hls-implementation.

Example C - Expensive Operations

- Sometimes HLS will fail to find a RTL primitive that matches the function you are asking for:
  - In these cases, you might need to decompose a single expensive operation into multiple cheaper operations.

- For example, Siemens Catapult HLS could not find primitives to implement our large multiplexers running at up to 280 MHz
  - We decomposed N-input multiplexer into (N-1) 2-input multiplexers, see illustrative example for N=8.



Decomposing a 8-input multiplexer to 7 2-input multiplexers

```
    o_loop : for (int o = 0; o < (1 << b); ++o) {
        index = (o << 1) + sel_off;
        x_int[o] = x_int[index];
    }
    sel_int = sel_int >> 1;
}
y = x_int[0];
```

Example D - Clock Retargeting

- After multiple blocks were integrated together, at a later stage of the design, we noticed that the switch matrix was failing to complete timing closure.

- The HLS tool was scheduling several operations in a single clock cycle:
  - We retargeted the HLS design to a higher clock frequency;
  - The scheduling step was forced to pipeline the design → at the price of higher latency.

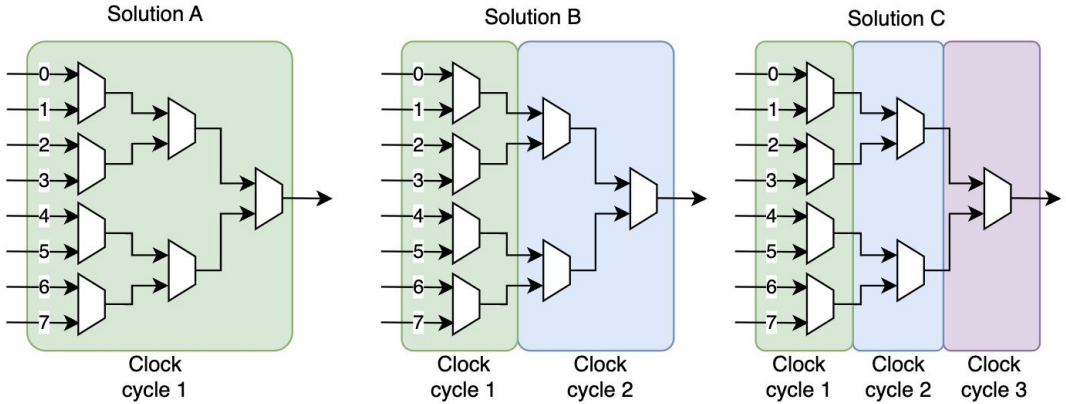
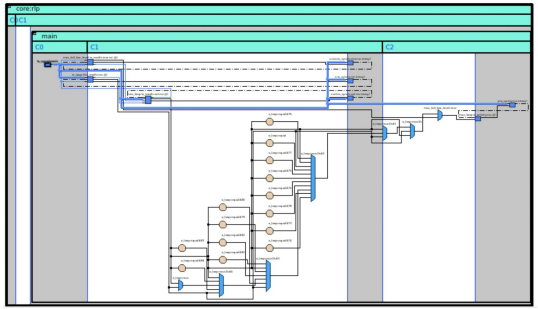


Illustration of 3 HLS solutions targeting 3 different clock frequencies



Catapult design analyzer showing muxes implemented in different clock cycles

Fmax vs latency trade-off

Summary

- This upgrade exercise is still under progress, but we already learned some lessons;
- Processing data using TDM is complex → in some cases it will be worthy to deserialize the data before processing;
- Describing logic in parallel and using HLS allows one to explore different resource sharing options automatically for pre-selected portions of the design using TDM:
  - Trading area by latency at any stage of the design without changing the source code.
- Reusing a block multiple times using TDM do not always lead to lower area:
  - In some cases lower resource sharing or no resource sharing at all is cheaper.
- HLS might not always find the primitives you need, but you may be able to succeed by decomposing a large task to multiple smaller ones.
- Describing logic in parallel enables clock retargeting at later stages of the design, and this can be done quickly and automatically using HLS:
  - Trading Fmax by latency at any stage without changing the source code.


ufjf
UNIVERSIDADE
FEDERAL DE JUIZ DE FORA

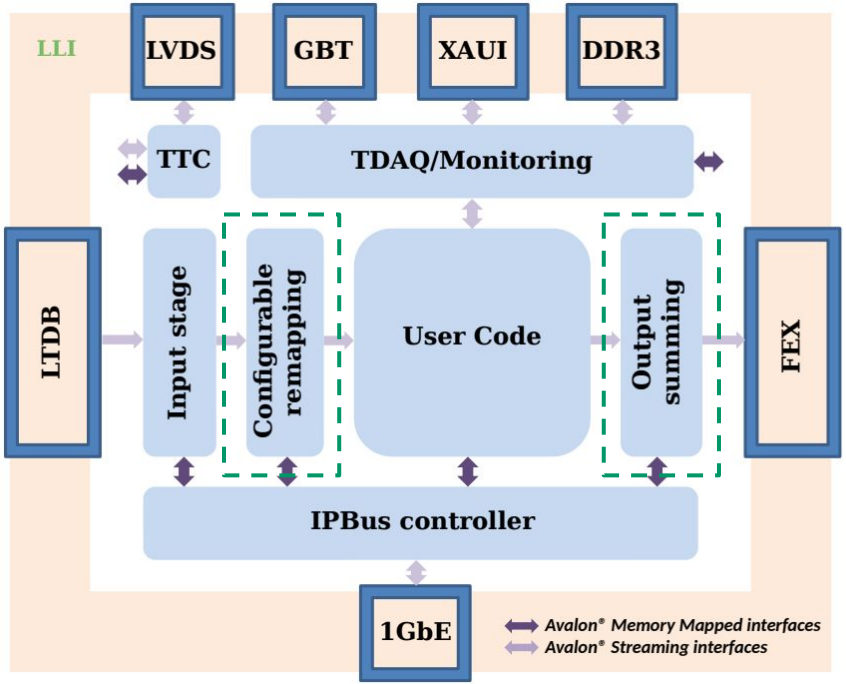

ATLAS
EXPERIMENT


CERN

Thank you!

Backup Slides

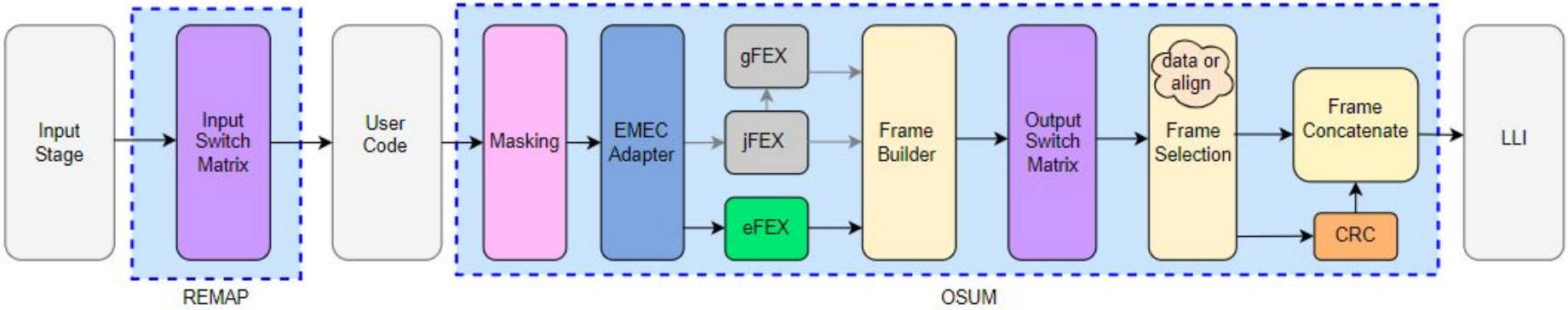
LATOME Firmware

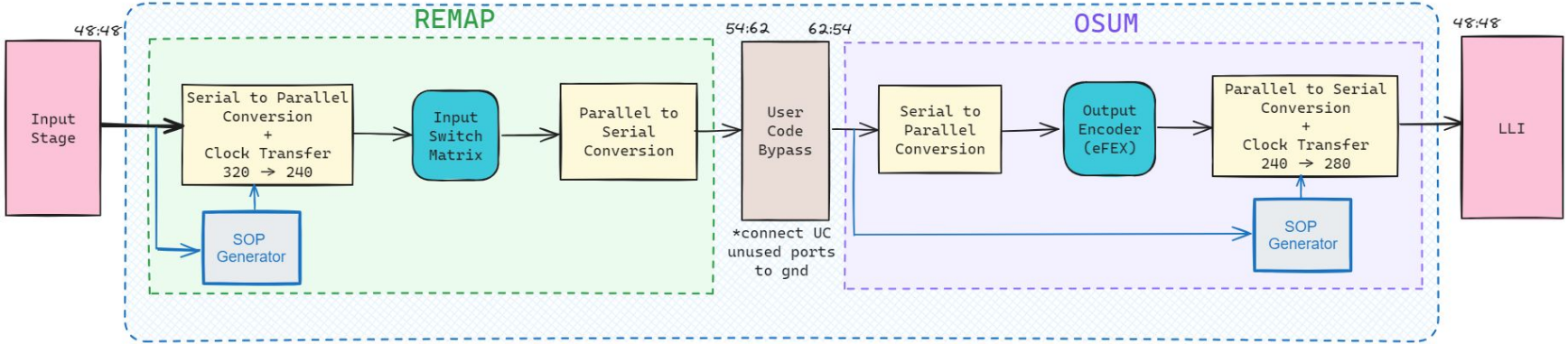


LATOME Block Diagram.

[1] Aad, Georges et al. “ATLAS LAr Calorimeter trigger electronics phase I upgrade: LATOME Firmware Specification”, 2023. Documentation link.

LATOME HLS Implementation





EMEC Adapter Block

- This block distributes the total energy of 6 super-cells into 4 super-cells, keeping the total energy conserved.
  - It is necessary because some of the **EMEC_HEC** towers feature 11 SCs.

```
26    if (ena) {
27        y[0]     = x_internal[0] + (x_internal[1] >> 1);
28        y[1]     = x_internal[2] + (x_internal[1] - (x_internal[1] >> 1));
29        y[2]     = x_internal[3] + (x_internal[4] >> 1);
30        y[3]     = x_internal[5] + (x_internal[4] - (x_internal[4] >> 1));
31        y[4]     = 0;
32        y[5]     = 0;
33
34        vld_o[0] = vld_i[0] or vld_i[1];
35        vld_o[1] = vld_i[1] or vld_i[2];
36        vld_o[2] = vld_i[3] or vld_i[4];
37        vld_o[3] = vld_i[4] or vld_i[5];
38        vld_o[4] = true;
39        vld_o[5] = true;
40    }
```

emec_adapter.h

$$E'_{F1} = E_{F1} + \lfloor E_{F2}/2 \rfloor$$
$$E'_{F2} = E_{F3} + (E_{F2} - \lfloor E_{F2}/2 \rfloor)$$
$$E'_{F3} = E_{F4} + \lfloor E_{F5}/2 \rfloor$$
$$E'_{F4} = E_{F6} + (E_{F5} - \lfloor E_{F5}/2 \rfloor)$$

Bits (in | out)	Saturation Protection	Latency	Throughput	Slack	Estimated Area
18 | 18	ON	1	1	1.69	260.23
18 | 18	OFF	1	1	2.09	120.23
18 | 19	ON	1	1	1.69	272.23
18 | 19	OFF	1	1	2.09	124.23
18 | 20	ON	1	1	1.69	260.23
18 | 20	OFF	1	1	0.89	280.23

Quantization studies for EMEC adapter

```
76  static int emec_cfg[EMEC_E][EMEC_N] =
77  {
78          {49, 50, 51, 52, 48, 88},
79          {55, 56, 57, 58, 54, 94},
80          {69, 70, 71, 72, 68, 108},
81          {75, 76, 77, 78, 74, 114},
82          {249, 250, 251, 252, 248, 208},
83          {255, 256, 257, 258, 254, 214},
84          {269, 270, 271, 272, 268, 228},
85          {275, 276, 277, 278, 274, 234}
86  };
```

Configuration matrix from LATOME System Level Development (specifies which supercells will be used in each case)

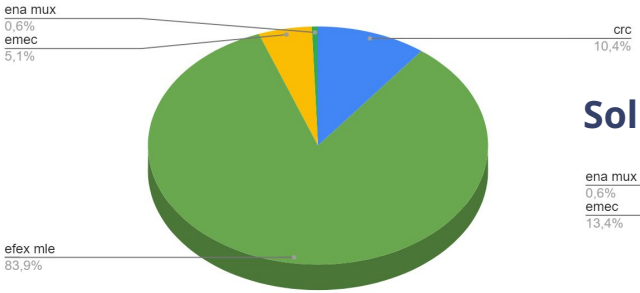
Resource Sharing Design Exploration

We can trade latency by area!

Solution	Latency	Throughput	Slack (ns)	ALMs (%)
A	5	1	0.335	19,603 (4.5%)
B	9	5	0.257	11,673 (2.7%)
C	11	7	0.278	12,089 (2.8%)

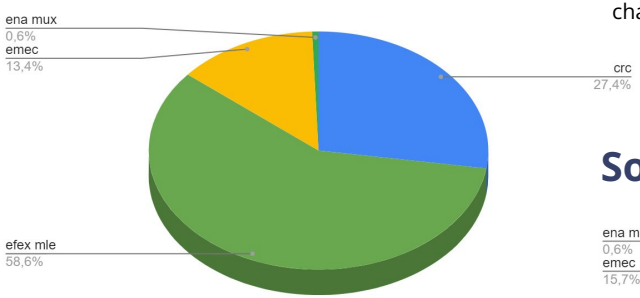
Preliminar performance results for each of the solutions after final characterization using Intel Quartus Prime

Solution A (320 MLEs)



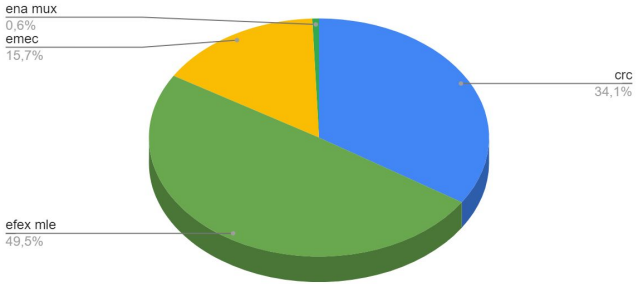
Comparing ALMs for the 4 main blocks of the Output Encoder

Solution B (64 MLEs)



Comparing ALMs for the 4 main blocks of the Output Encoder

Solution C (48 MLEs)



Comparing ALMs for the 4 main blocks of the Output Encoder

The same code can be targeted to different resource sharing options and clock frequencies!

The MLE block is the one that takes the most largest portion of area!

- **Solution A:** the design can run at any frequency multiple of 40 MHz;
- **Solution B:** the frequency of the design can be 240 MHz or 280 MHz;
- **Solution C:** due to the constraints imposed by resource sharing, the frequency can be only 280 MHz.