UM–SJTU Joint Institute

System-on-Chip Design (ECE4810J)

Laboratory Report

Lab 2. Design with high level languages on Arty Z7 SoC development platform

Group 2

Name: Haochen Wu	ID: 518021910558
Name: Yihua Liu	ID: 518021910998
Name: Siyuan Zhang	ID: 518370910180

Date: October 14, 2021

Contents

1 Overview

In this lab, we will go through two exercises to create design with high level languages (instead of Verilog). The goals of this lab are to:

- Get familiar with the concept of High-level Synthesis (HLS)
- Learn to use HLS to create an IP block in C/C++
- Learn about differences between programming with HLS vs. Verilog in terms of design quality
- Get started with the Arty Z-7 board as a Python accelerator and be able to run example Jupyter notebook with the board

2 Creating and using custom IP blocks using High-Level Synthesis

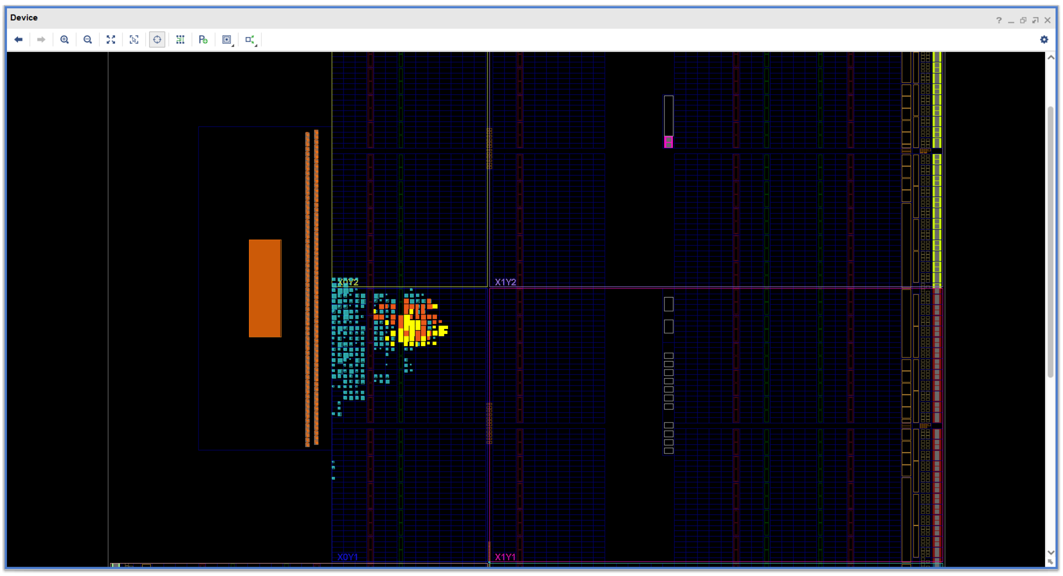


Figure 1. The resource utilization of the HLS multiplier (screenshot).

The red leaf cells are RTL multiplier, and the yellow leaf cells are HLS multiplier. By comparing their area, we can find that the RTL implementation costs less resources, probably due to RTL version is more logically optimized and compact.

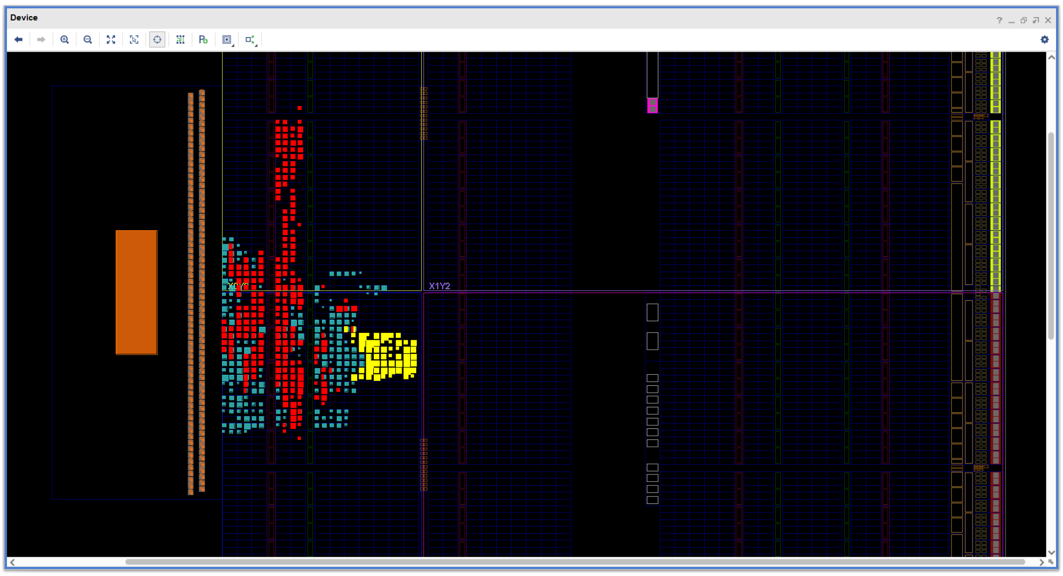


Figure 2. The resource utilization of the HLS divider (screenshot).

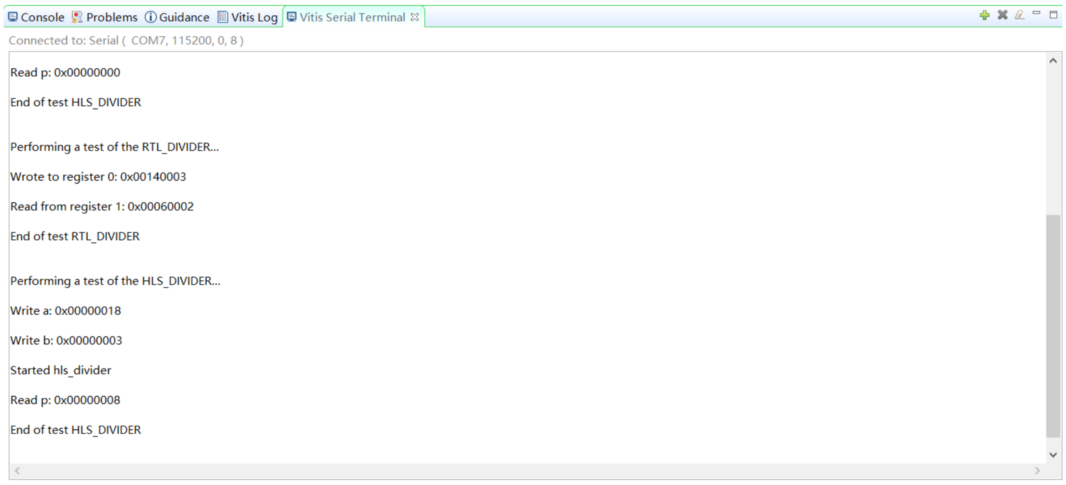
For resource utilization of divider, the HLS consumes less area than RTL version. It's probably because RTL divider we write is not quite efficient, which costs so many resource usage.

```c
#include <stdio.h>
unsigned int hls_divider(unsigned short int a, unsigned short int b);
int main() {
        unsigned short int a, b;
        unsigned int q;
        a = 20;
        b = 3;
        q = 0;
        printf("initialized variables: a=%d, b=%d, q=%d \n", a, b, q);
        q = hls_divider(a, b);
        printf("testing hls_divider: %d / %d = %d \n", a, b, q);
        return 0;
}
```

Listing 1. Source code of the resource utilization of the HLS divider (test bench).

```
unsigned int hls_divider(unsigned short int a, unsigned short int b) {
#pragma HLS INTERFACe s_axilite port=return bundle=CRTLS
#pragma HLS INTERFACe s_axilite port=a bundle=CRTLS
#pragma HLS INTERFACe s_axilite port=b bundle=CRTLS
        unsigned int q;
        q = a / b;
        return q;
}
```

Listing 2. Source code of the resource utilization of the HLS divider (source).



```
Connected to: Serial ( COM7, 115200, 0, 8 )

Read p: 0x00000000
End of test HLS_DIVIDER

Performing a test of the RTL_DIVIDER...
Wrote to register 0: 0x00140003
Read from register 1: 0x00060002
End of test RTL_DIVIDER

Performing a test of the HLS_DIVIDER...
Write a: 0x00000018
Write b: 0x00000003
Started hls_divider
Read p: 0x00000008
End of test HLS_DIVIDER
```

Figure 3. SDK terminal printout of the HLS divider.

For RTL divider, we concatenate the quotient and the remainder together to form a 32-bit number to read from register 1. Since 20/3=6......2, the RTL can be verified as working properly. For HLS divider, we don't pertain this design, where the result is merely the qoutient, so 24/3=8.

```
unsigned int hls_ALU(unsigned int a, unsigned int b, unsigned int c) {
#pragma HLS INTERFACe s_axilite port=return bundle=CRTLS
#pragma HLS INTERFACe s_axilite port=a bundle=CRTLS
#pragma HLS INTERFACe s_axilite port=b bundle=CRTLS
#pragma HLS INTERFACe s_axilite port=c bundle=CRTLS
 unsigned long int result;
 if (c==0){result =  a + b;}
 if (c==1){result = a - b;}
 if (c==2){result = a * b;}
 if(c==3){result = a / b;}

 return result;
}
```

Listing 3. Source code of ALU.

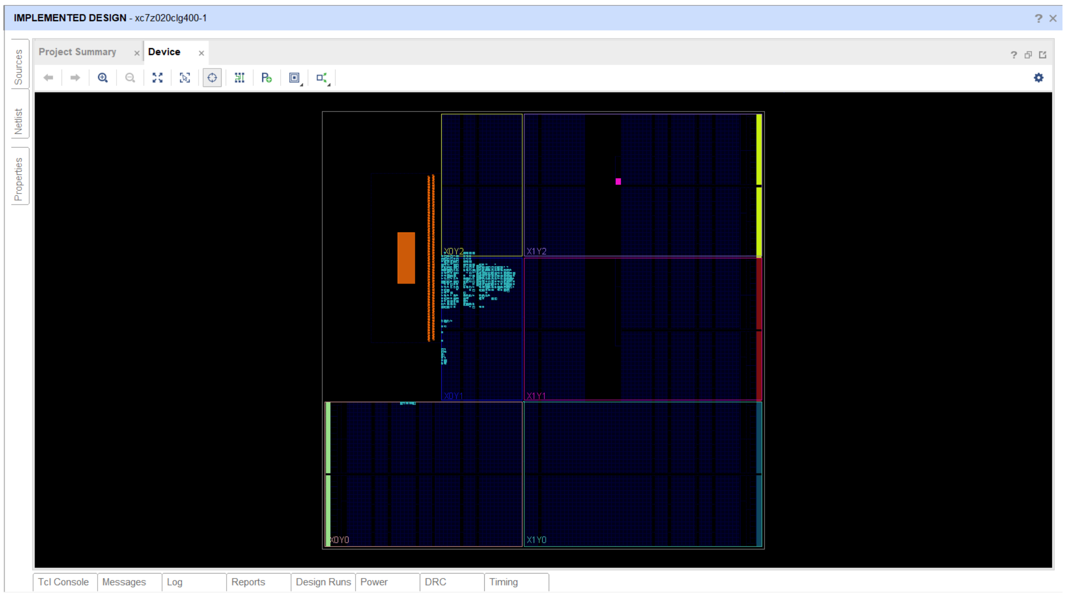
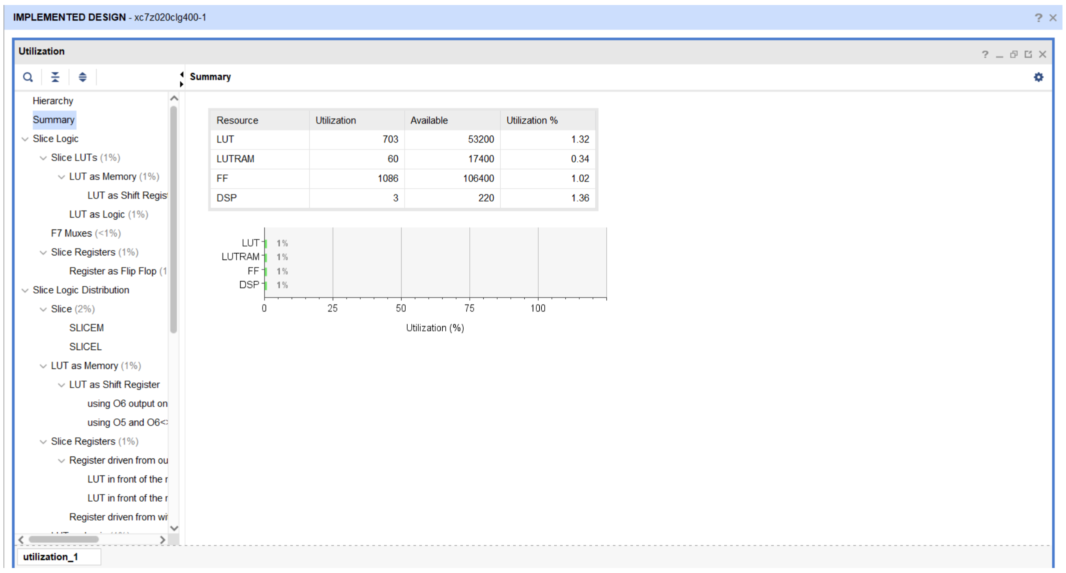


Figure 4. The resource utilization of ALU.


IMPLEMENTED DESIGN - xc7z020clg400-1

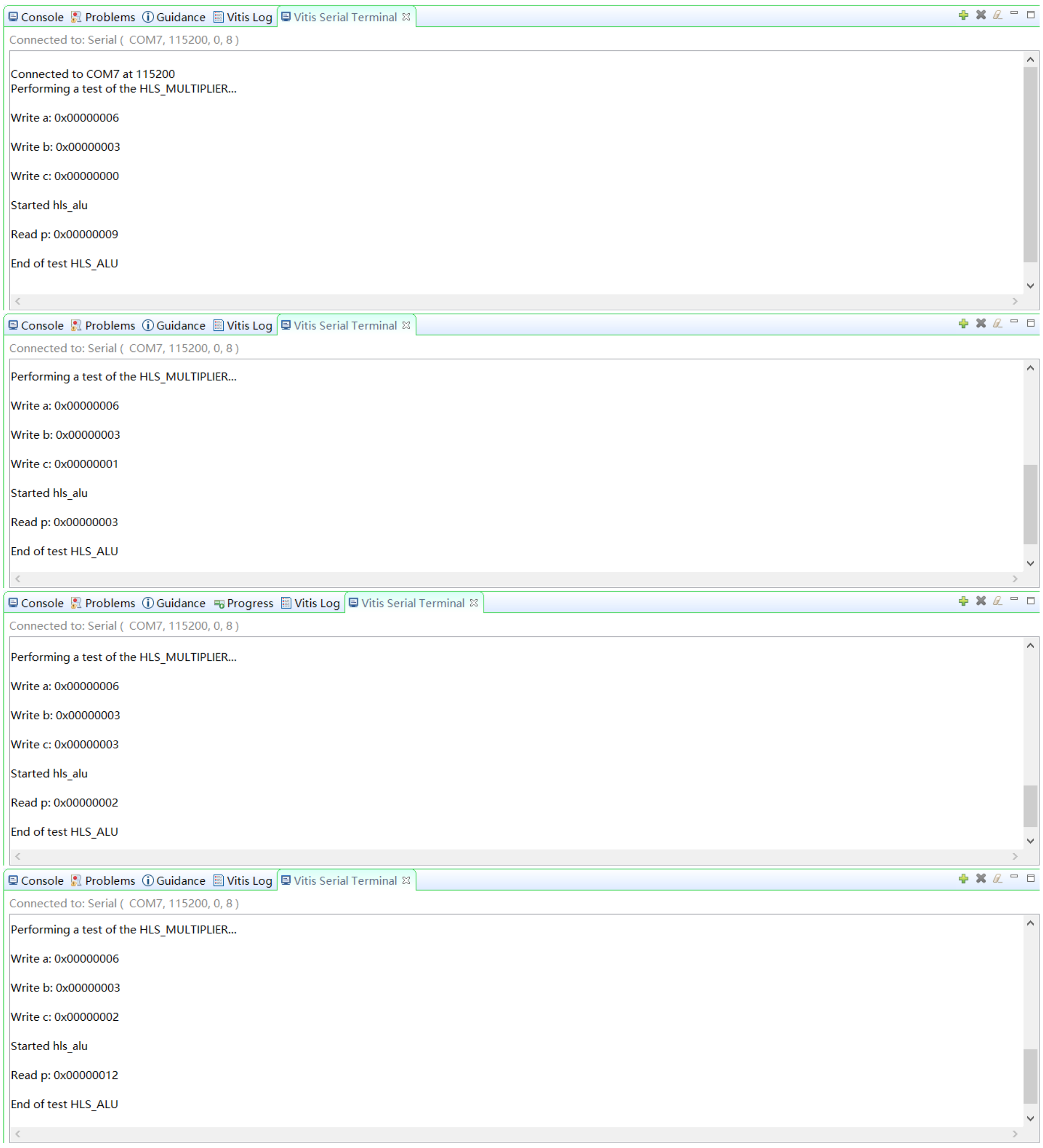
Utilization

Hierarchy
Summary
Slice Logic
Slice LUTs (1%)
LUT as Memory (1%)
LUT as Shift Regis
LUT as Logic (1%)
F7 Muxes (<1%)
Slice Registers (1%)
Register as Flip Flop (1
Slice Logic Distribution
Slice (2%)
SLICEM
SLICEL
LUT as Memory (1%)
LUT as Shift Register
using O6 output on
using O5 and O6<
Slice Registers (1%)
Register driven from ou
LUT in front of the r
LUT in front of the r
Register driven from wi

Summary

Resource	Utilization	Available	Utilization %
LUT	703	53200	1.32
LUTRAM	60	17400	0.34
FF	1086	106400	1.02
DSP	3	220	1.36

Figure 5. The resource utilization of ALU (summary).



```
Connected to: Serial ( COM7, 115200, 0, 8 )

Connected to COM7 at 115200
Performing a test of the HLS_MULTIPLIER...

Write a: 0x00000006

Write b: 0x00000003

Write c: 0x00000000

Started hls_alu

Read p: 0x00000009

End of test HLS_ALU
```

```
Connected to: Serial ( COM7, 115200, 0, 8 )

Performing a test of the HLS_MULTIPLIER...

Write a: 0x00000006

Write b: 0x00000003

Write c: 0x00000001

Started hls_alu

Read p: 0x00000003

End of test HLS_ALU
```

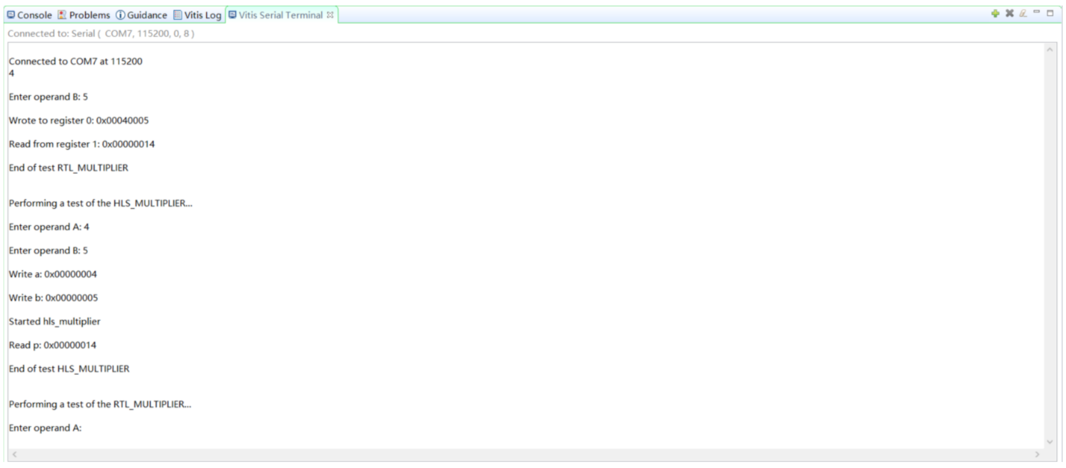
```
Connected to: Serial ( COM7, 115200, 0, 8 )

Performing a test of the HLS_MULTIPLIER...

Write a: 0x00000006

Write b: 0x00000003

Write c: 0x00000003

Started hls_alu

Read p: 0x00000002

End of test HLS_ALU
```

```
Connected to: Serial ( COM7, 115200, 0, 8 )

Performing a test of the HLS_MULTIPLIER...

Write a: 0x00000006

Write b: 0x00000003

Write c: 0x00000002

Started hls_alu

Read p: 0x00000012

End of test HLS_ALU
```

Figure 6. SDK terminal printout of ALU.

For four arithmetic operations of ALU, we test them respectively. Code 0 indicates summation, which is 6+3=0. Code 1 indicates subtraction, which is 6-3=3. Code 2 indicates multiplication, which is 6*3=18=0x12. Code 2 indicates division, which is 6/3=2.

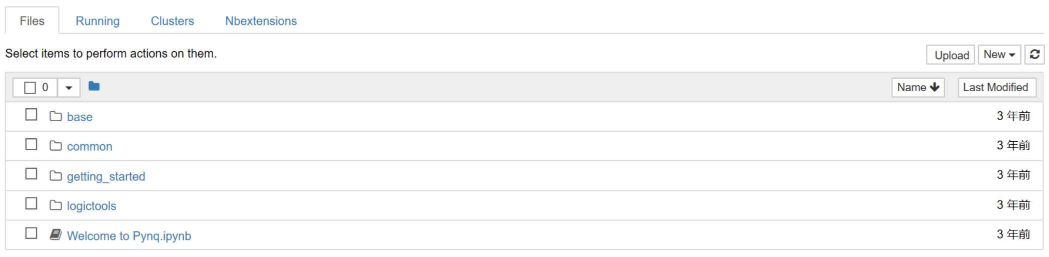
3 Programming the FPGA and Booting Software from the SD card



```
Connected to: Serial ( COM7, 115200, 0, 8 )

Connected to COM7 at 115200
4

Enter operand B: 5

Wrote to register 0: 0x00040005

Read from register 1: 0x00000014

End of test RTL_MULTIPLIER


Performing a test of the HLS_MULTIPLIER...

Enter operand A: 4

Enter operand B: 5

Write a: 0x00000004

Write b: 0x00000005

Started hls_multiplier

Read p: 0x00000014

End of test HLS_MULTIPLIER


Performing a test of the RTL_MULTIPLIER...

Enter operand A:
```

Figure 7. The screenshot from the SDK terminal or PUTTY terminal.

4 Programming Python on Arty Z7-20



Select items to perform actions on them.

Name	Last Modified
base	3 年前
common	3 年前
getting_started	3 年前
logictools	3 年前
Welcome to Pynq.ipynb	3 年前

Figure 8. The screenshot of the Jupyter notebook.

5 Questions

1. What was the resource utilization of the rtl implementation vs. HLS implementation? If they are different, can you guess why there are differences?

   From previous results of RTL and HLS multiplier and divider in part 2, we find that RTL and HLS modules use different amount of building blocks on FPGA. Generally, the RTL implementation should be more compact comparing to HLS implementation. For instance of multiplier in Figure 1, the RTL implementation covers smaller area compared to HLS implementation. The reason is we can specific RTL hardware details and RTL module is logically optimized. However, for HLS implementation, there might be some redundant circuit elements.

   Nevertheless, from our utilization report of RTL and HLS divider, we notice that HLS divider in fact consumes less recourse, which probably because our RTL implementation is not logically optimized enough. Besides, the high-level synthesis process itself also carries out some optimizations.

2. Imagine that you have to create an IP block implementing 10 different sorting algorithms. Would you rather use a hardware description language or High-level Synthesis? Why?

   We would use high-level synthsis rather than hardware description language. The reason is that to implement an IP block with 10 different sorting algorithms manually by hardware description language is very complex and costs a lot of time on hardware logic, however, using high-level synthesis, it is much easier, and we can focus on implementing sorting algorithms themselves rather than hardware logic.

3. Read more about PYNQ, and answer the following questions.

   (a) What is a Jupyter notebook? Why do we use it?
   Jupyter notebook is a platform which support Python and relating numerous packages we can easily dowload and use. It use blocks to execute codes step by step. Because we need to implement Pynq designs and Jupyter notebook can have a nice interface with the Arty-7z board through Ethernet connection to PC.

   (b) What are overlays? What are the benefits of using overlays?
   Overlays are hardware designs made by hardware engineers for software engineers to use. Overlays are just like package in software programming, it can be used efficiently in other software design without having any knowledge about the hardware. It can be used to accelerate software or support the hardware platform for certain software. Convenience and modularity is the most benefit.

   (c) Please list available overlays on the Arty-Z7 board.
   On the Arty-Z7, it supports BaseOverlay and Logictools Overlay. Also users can custom their own overlays.

   (d) Comparing part 2 where you design with C/C++ vs. part 4 (PYNQ framework with python), what are the major differences?
   Design in C/C++ can accomplish more complicated and advanced function and it is more extensible. Besides, design with C/C++ is mature. In contrast, Pynq can take full advantage of package written in python and FPGA, which can shorten the design period and make it easier for design.