Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

INTRODUCTION TO SoC

Practical session - Semester 251

WEEK 1

Introduction & Arithmetic

1 Introduction

1.1 Aims

- Get familiar with Vivado software.
- Get familiar with the FPGA Arty-Z7 board.
- Practice in designing digital logic circuits with Verilog.
- Understand the hierarchical design principle.
- Practice in writing test benches for a designed module.
- Get familiar with UART integration on FPGA
- Understand FSM models and the principles of designing logic circuits with FSM.

1.2 Preparation

- Read the laboratory materials before class.
- Revise Verilog basics.
- Each group prepare at least one laptop with Vivado software installed.
- Download prebuilt RISC-V toolchain.
- Recommend using Linux for the Lab session.

1.3 Documents and lab materials

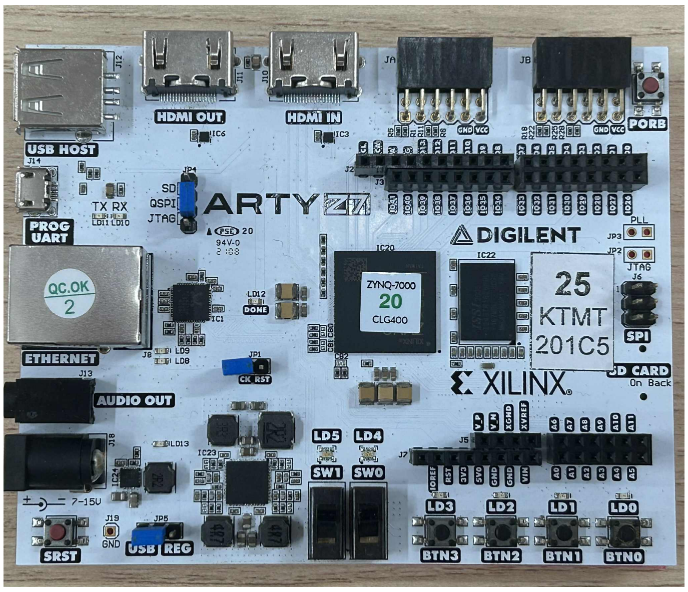


Figure 1: Arty-Z7 FPGA board

- M. Morris Mano, Michael D. Ciletti, Digital System with an Introduction to the Verilog HDL, VHDL, and SystemVerilog, Pearson Education, Inc., 2017
- Lecture slides
- *Arty-Z7-20-Master.xdc*: Arty-Z7 constraint file.
- *Guide_for_Installing_Vivado.pdf*: Guide for installing Vivado and getting started with Vivado and Arty-Z7.

1.4 Procedure

For each exercise (also for further labs):

- Read the requirements, then determine the input/output signals of your circuits.
- Make a design idea of the circuit, then use Verilog to model the circuit.
- Analysis & Synthesis the circuit with Vivado software.
- Write a test bench to simulate the circuit on Vivado Simulator.
- Generate the bitstream and program the Arty-Z7 to evaluate the circuit.

1.5 Report requirements

- Lab exercises will be reviewed directly in class.
- Write a PDF report (with circuit/simulation screenshots inserted).
- Must have group ID, group members' names, and student IDs in the report.
- Must explain the flow of code, capture the waveform, and the simulation results on board Arty-Z7
- Submit on BK-elearning by deadline

2 Exercises

Arty-Z7 extension board will be used for the following exercise:

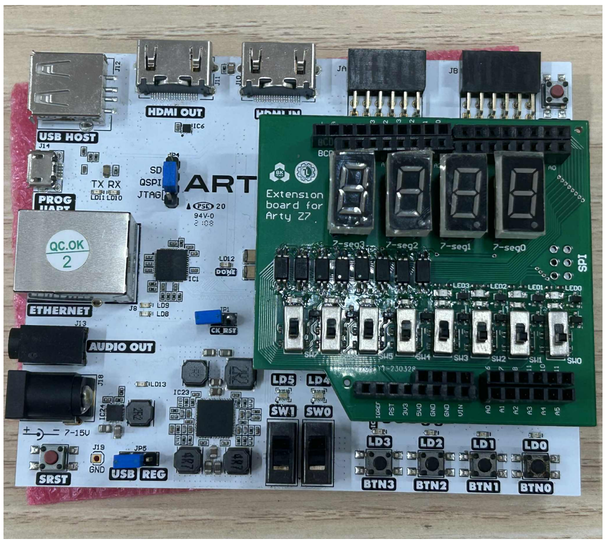


Figure 2: Arty-Z7 with extension board

2.1 Exercise 1

Traffic Light Control System

A **traffic light** is a device used to control the flow of vehicles at intersections with heavy traffic (typically at busy three-way or four-way junctions). It is an essential system that not only ensures the safety of vehicles but also helps reduce traffic congestion during peak hours. Traffic lights can operate automatically or be manually controlled by traffic police.

The traffic light system consists of three signal colors:

- **Red**: When the red light is on, all vehicles must stop before the stop line (except for right-turn vehicles and emergency vehicles performing their duties).
- **Green**: When the green light is on, all vehicles are allowed to go.
- **Yellow**: The yellow light indicates a transition between the green and red signals.

2.1.1 System Design Requirements

- Use **RGB LEDs** to simulate the traffic light system (e.g., red light – red LED, green light – green LED, yellow light – blue LED).
- Use **7-segment displays** to show the countdown timer for each light.
- Use the **buttons** and **switches** on the FPGA development kit (Arty Z7 or equivalent) to set the timing values for the lights and perform system configuration or mode changes.

2.1.2 System Functional Requirements

- Simulate the operation of two traffic light poles for two intersecting lanes, each consisting of three lights in the order: **Red, Yellow, Green**, ensuring proper coordination between the two poles.

2.2 Exercise 2

Seven-Segment LED Decoration System

The **Seven-Segment LED Decoration System** is a popular display system commonly used in electronic signboards on streets and in building or company decoration systems. The operating principle of this system is to create motion or shifting effects for characters displayed on seven-segment LED modules. A seven-segment LED decoration system can support multiple display modes.

2.2.1 System Design Requirements

- Use the **seven-segment LEDs** on the FPGA board to display LED effects.
- Use the **switches** and **buttons** on the FPGA development kit (Arty Z7 or equivalent) to modify and adjust the system operation.
- Use individual **single-color LEDs** to provide additional visual indicators or support for the system.

2.2.2 System Functional Requirements

- The system must support at least **two display effects** using the seven-segment LEDs. Some examples of possible effects include:
  - **Effect 1:** The seven-segment LEDs display the text "CE" scrolling from left to right repeatedly.
  - **Effect 2:** The seven-segment LEDs display the text "CE". When the text reaches the right edge, it will shift left; on the other hand, when the text reaches the left edge, it will shift right.

2.3 Exercise 3

Change mode String bit LED circuit.

Use Verilog HDL to model a state machine for a circuit that changes the display mode of a bit string.

Initially, LEDs show the default 4-bit random string, which is performed by a reset signal, for example, string: 0011. And buttons in the board will set the display mode as follow:

- Button 0: Mode Reset: Show the default 4-bit string on LEDs.
- Button 1: Mode Circular Shift Left Ring : Shift 4-bit string to left in a ring every 1s.
- Button 2: Mode Circular Shift Right Ring: Shift 4-bit string to right in a ring every 1s.
- Button 3: Pause: Pause the current shifting string.

Write a test bench to simulate the circuit and test the circuit on the Arty-Z7 board. In addition, students need to use the UART output to send the current LED string to the PC.

3 Divider

- In this homework, you will create a key component of the RISC-V processor: the **divider unit**.
- RISC-V contains four division/remainder instructions:`div`, `divu`, `rem`, and `remu`, which perform signed and unsigned divide and remainder operations.
- You will build a module that performs **unsigned** division and remainder operations.
- In a later homework, when you build a complete RISC-V processor, you will add extra logic to support the **signed** division and remainder operations.

3.1 Division Algorithm (Software)

The module takes as input two 32-bit data values: the `dividend` and `divisor`. It outputs two 32-bit values: the `quotient` and `remainder`.

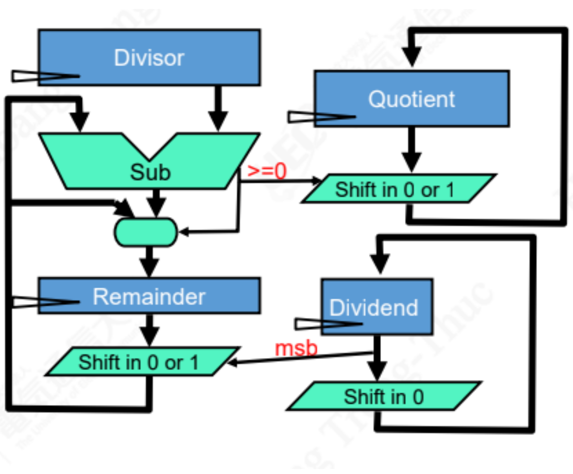


Figure 3: Sequential Divisor

Use the following algorithm, written in C, as a model for your hardware implementation:

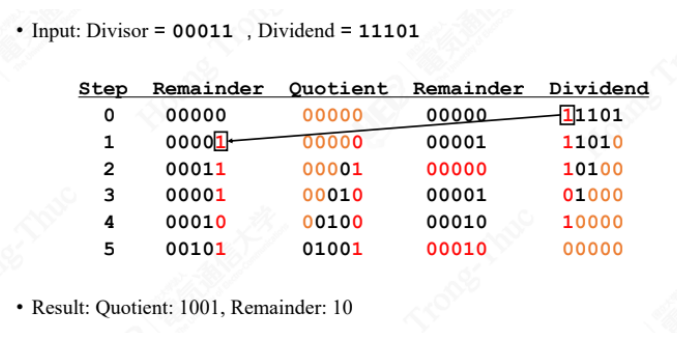
```c
int divide(int dividend, int divisor) {
    int quotient = 0;
    int remainder = 0;

    for (int i = 0; i < 32; i++) {
        remainder = (remainder << 1) | ((dividend >> 31) & 0x1);
        if (remainder < divisor) {
            quotient = (quotient << 1);
        } else {
            quotient = (quotient << 1) | 0x1;
            remainder = remainder - divisor;
        }
        dividend = dividend << 1;
    }

    return quotient;
}
```

Listing 1: Division Algorithm

Your circuit will compute the quotient and remainder in the same way, but in a **single cycle using combinational logic only**.



• Input: Divisor = 00011 , Dividend = 11101

Step	Remainder	Quotient	Remainder	Dividend
0	00000	00000	00000	11101
1	00001	00000	00001	11010
2	00011	00001	00000	10100
3	00001	00010	00001	01000
4	00010	00100	00010	10000
5	00101	01001	00010	00000

• Result: Quotient: 1001, Remainder: 10

Figure 4: Divide example

3.2 Corner Case: Divide-by-Zero

A divisor of zero is a special case. **You do not need to handle this case** in your modules.

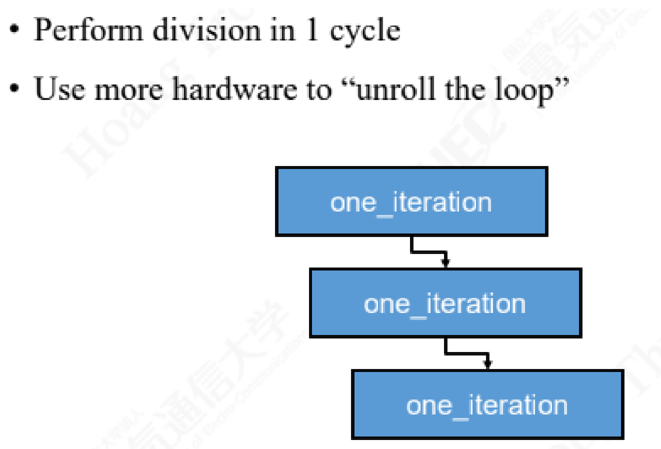
In a later homework, you will need to handle this and other corner cases according to the RISC-V ISA specification.

3.3 Verilog Restrictions

- You **may not** use Verilog's / or % operators in your code.

3.4 Implementation Instructions

1. Begin by writing the `divu_1iter` module that performs **one iteration** of the division algorithm.
2. Then, instantiate this module 32 times to construct the full `divider_unsigned` module.
3. Carefully consider how each output value is computed from the inputs at each step.



• Perform division in 1 cycle

• Use more hardware to "unroll the loop"

Figure 5: Combinational divider

3.5 Testing

We have provided tests for both the `divu_1iter` and `divider_unsigned` modules.

To run tests, use the following command:

```
MAKEFLAGS=-j4 pytest --exitfirst --capture=no testbench.py
```

This command runs the tests for both modules. Only a few simple test cases are provided in the `testbench.py` files.

You are encouraged to use these files as templates to add your own test cases, especially for edge and corner cases.

3.6 Submission Instructions

Submit the following files:

- `divu_1iter.v`
- `divider_unsigned.v`

- In addition to the source codes, students must also prepare a PDF file to explain the codes, the reasoning behind them, and show the results.
- The results include screenshots of the waveforms and photos of the board output (if the assignment involves an FPGA board).
- Email tile format: **[Introduction to SoC] [student ID] Practice #no.**
- Send source codes and PDF report to:
  hoangtt@uec.ac.jp, cc: anhpkn@hcmut.edu.vn, tnthinh@hcmut.edu.vn