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

INTRODUCTION TO SoC

Practical session - Semester 251

WEEK 4

Multi-Cycle Datapath

1 Introduction

1.1 Aims

- Get familiar with Vivado software.
- Get familiar with the FPGA Arty-Z7 board.
- Get familiar with RISC-V toolchain
- Understand the hierarchical design principle.
- Practice in writing test benches for a designed module.

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

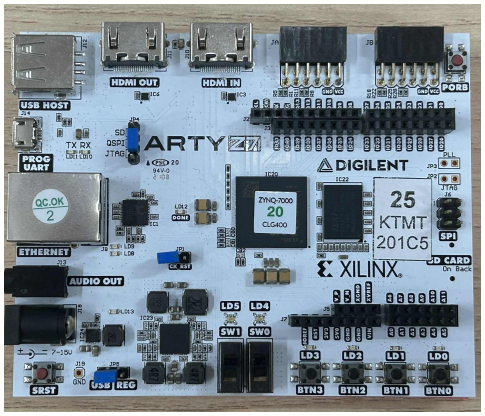

Figure 1: Arty-Z7 FPGA board

- 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 explain the flow of code, capture the waveform, and the simulation results on board Arty-Z7
- Submit on BK-elearning by deadline

2 Multi-Cycle Datapath

Pipeline Divider and Multi-Cycle Datapath.

This homework has two main components:

- First, pipeline your divider from homework #1 to improve its critical path.
- Then, integrate your faster divider with your processor from homework #3.

2.1 Multi-Cycle Datapath

The *multicycle microarchitecture* executes instructions in a series of shorter cycles. Simpler instructions execute in fewer cycles than complicated ones. Moreover, the *multicycle microarchitecture* reduces the hardware cost by reusing expensive hardware blocks, such as adders and memories.

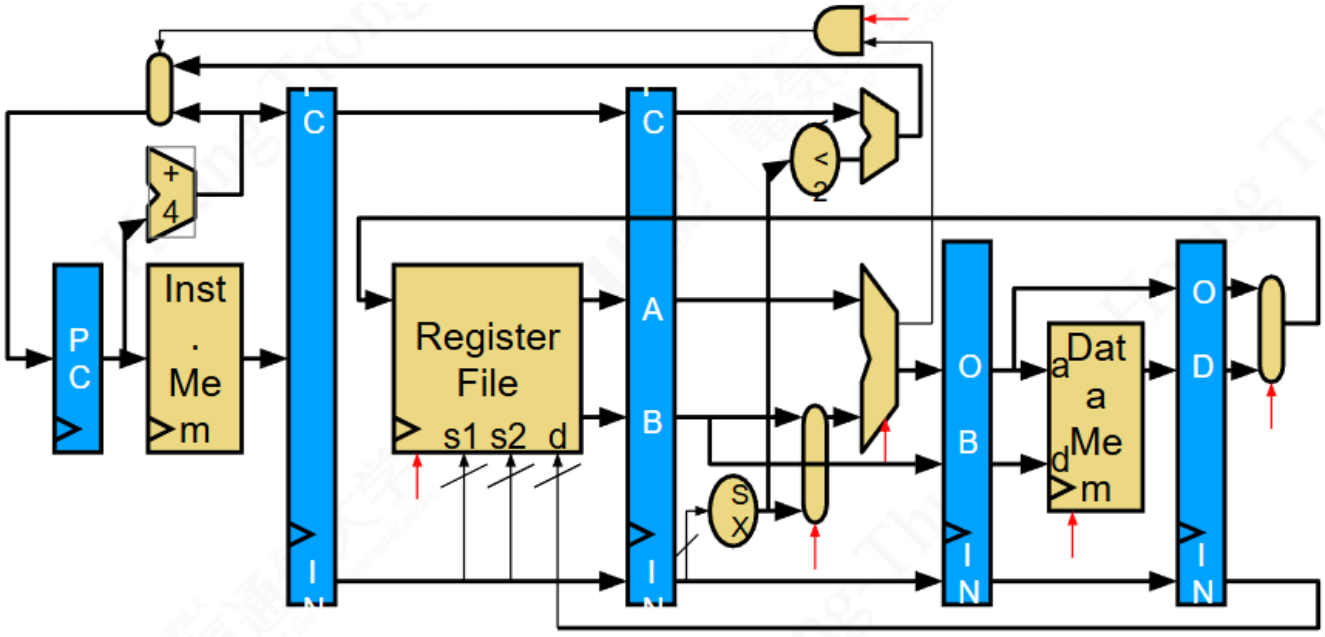

Figure 2: Multi-Cycle Datapath

For example, the adder may be used on different cycles for several purposes while carrying out a single instruction. The multicycle microprocessor accomplishes this by introducing several nonarchitectural registers to hold intermediate results. The multicycle processor executes only one instruction at a time, but each instruction takes multiple clock cycles. This processor requires only a single memory, accessing it on one cycle to fetch the instruction and on another to read or write data. Because they use less hardware than single-cycle processors, multicycle processors were the historical choice for inexpensive systems.

2.2 Your Design

2.2.1 8-stage Pipelined Divider

You will need to design a critical path which means improve your divider from your homework #1. The critical path is the maximum-delay path between two registers. Adding more registers can help by reducing this critical path, which in turn boosts clock frequency and overall throughput. However, inserting extra registers also increases the latency of a single divide instruction, creating a trade-off between performance and individual c

Figure 3: Your pipelined divider

Your `divu_1iter.v` code can be recycled. You should start from your `divider_unsigned.v` and convert it to have 8 pipeline stages.

Each stage should complete 4 *"iterations"* of the division algorithm, and your divider should support starting a new division operation on each cycle. The *'stall'* input can be ignored for now.

2.2.2 Datapath Integration

After your divider is working, you will need to integrate it into your datapath.

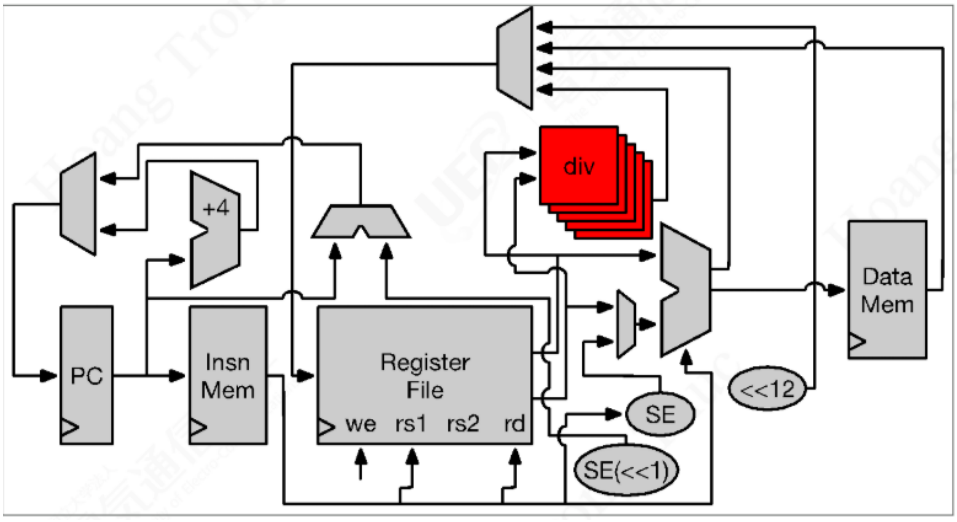

Figure 4: Your critical datapath design

You can start by copying your homework #3 singlecycle to `DatapathMultiCycle.v`.

Note that the CLA code from homework #2 is still needed here.

Your datapath is almost entirely unchanged from homework #3, except that *'div'*, *'divu'*, *'rem'*, and *'remu'* instructions will take **8 cycles** instead of one.

We'll refer to these 4 kinds of instructions as "**divide operations**" for simplicity.

Your divider module is pipelined and so, theoretically, you could start back-to-back independent divide operations on consecutive cycles for improved performance.

However, your datapath will not (yet) take advantage of this and so consecutive divide operations will take 8 cycles each.

Thus, *k* consecutive divide operations will take *8k* cycles.

All told, the datapath changes amount to around a dozen lines of code.

For this homework, you will again run synthesis and place-and-route to see how much your improved divider design affects overall frequency.

2.3 Submitting your code

You cannot use the '-', '/' or '%' operators in your code.

However, the '-' operator is ok in the divider.

For this homework, you will again run synthesis and place-and-route to see how much your improved divider design affects overall frequency. Also, you will need to show the waveform for us.

Submit your `DividerUnsignedPipelined.v` and `DatapathMultiCycle.v` file together with clock frequency and LUT usage reports.

- 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