Philip Schiffrin (pjs2186)
Akira Baruah (akb2158)
Chaiwen Chou (cc3636)
Sean Liu (sl3497)

NES-PACS 6502

Design Document

Introduction

Our project is the emulation of the 6502 microprocessor on Altera's Cyclone V FPGA. We were originally interested in pursuing this project because we wanted to emulate the Nintendo Entertainment System. However, upon further investigation, emulation of the entire system seemed like too broad of a project and we decided to pursue emulating the 2A03, which is the modified 6502 processor found inside the NES. The only major modification on the 2A03 is the removal of the binary coded decimal mode which was originally found on the 6502; otherwise, the processors have the same internal design.

This is an exciting project to work on both because it allows us to design a complex, synchronous system in system verilog and because it gives us a fundamental insight into the nature of a processor. The 6502 was revolutionary at its time and is cited as one of the building blocks for the modern pipelined processor.

Processor Architecture

The architecture of our 6502 emulation follows the overall design of the original processor while modifying the implementation of internal buses and the clock. While the original processor used bidirectional buses to communicate both between internal components and with the off-board RAM, our use of the FPGA, which does not allow for bidirectional buses, requires a modified design.

For external communication between the processor and RAM/peripherals, we will have two separate 8-bit registers, one for data-in and one for data-out. Internally, all registers read from and write back to each bus to which it is connected to in the original design (Figure 1.1). However, writing to the bus is controlled by a multiplexer, so that only one register can write to a bus on any given cycle. Each register will read from the buses to which it is connected on each cycle, but the use of that value is controlled by an enable signal which is output by the overall control logic. If the signal on a given bus is irrelevant for a particular register, the enable signal will be low and the value will be thrown away. Any register that needs the value will have their enable signal go high.

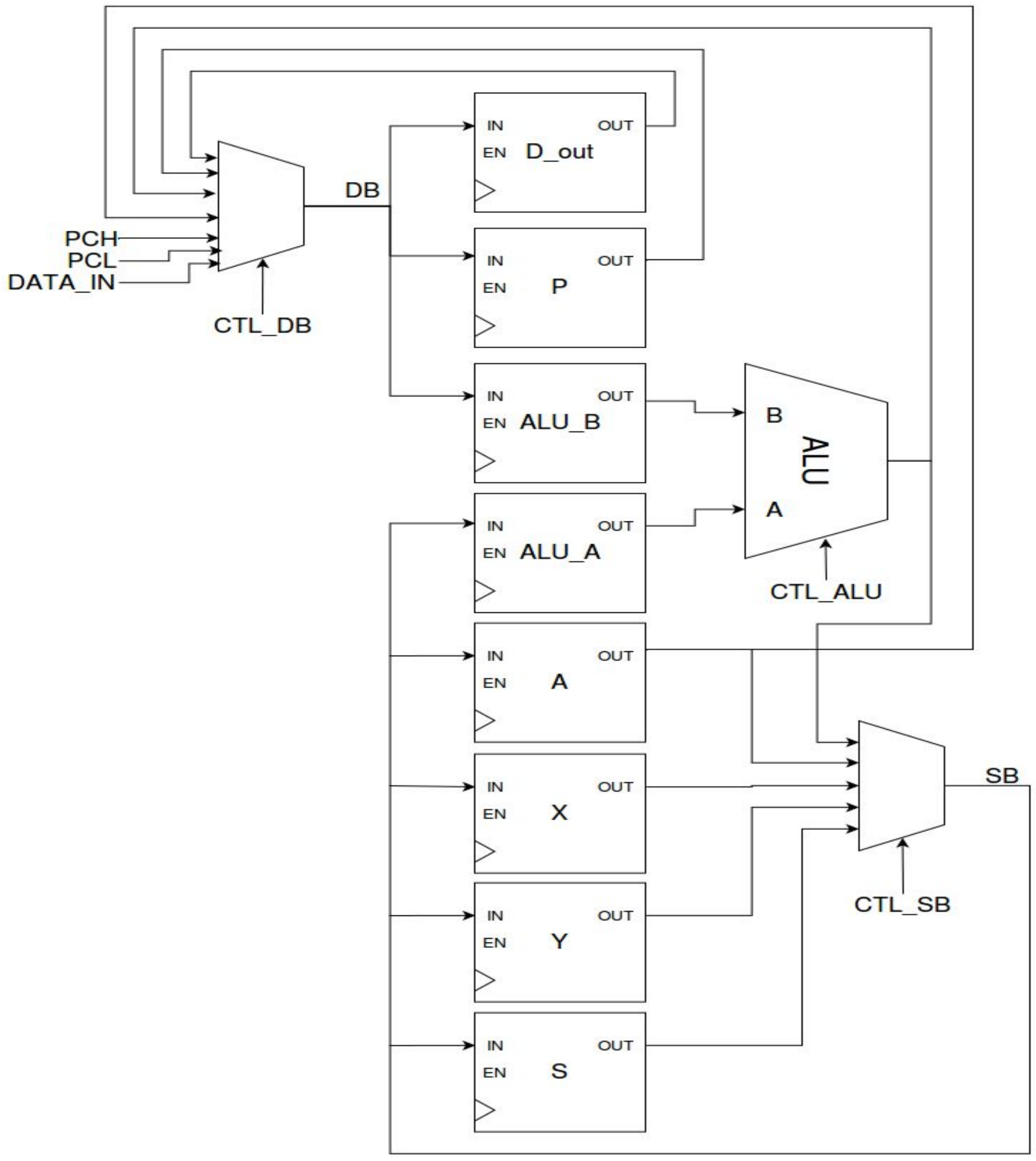

Figure 1.1: Processor registers and control logic

For timing, the original 6502 divides one clock into two phases, each of which controls certain components within the processor. Specifically, the data is read from the bus in one phase and an address is written in another phase. Our design combines the two phases into a single clock cycle, where each cycle performs decoding of the opcode, i.e the setting of each control signal for the current operation, and begin execution of the given instruction. What beginning execution refers to will depend on the action needed for the given instruction, but includes use of the ALU for operations that require any computation. For example, in the single byte instruction TAX, the value of the accumulator is loaded into the X index register. Figure 1.2 shows how the opcode arrives and is translated on T0 and the value in the accumulator is put on the SB bus on T1 via the en_X signal going high. Finally, in the next instruction's T0, the value is read by the X index register from the bus.

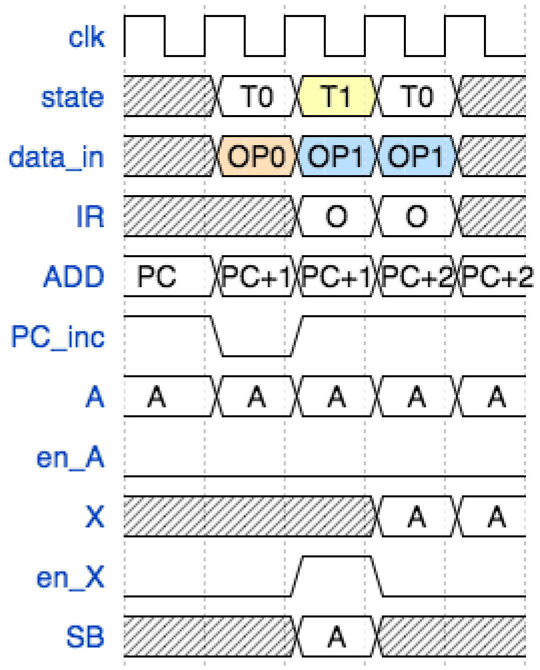

Figure 1.2: Timing for TAX instruction

The following diagrams indicate the timing for communication between the processor and the block ram.

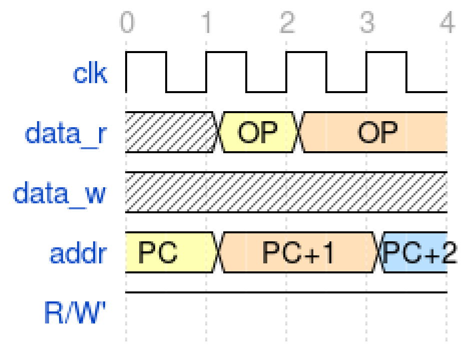

Figure 1.3: Single byte instruction timing

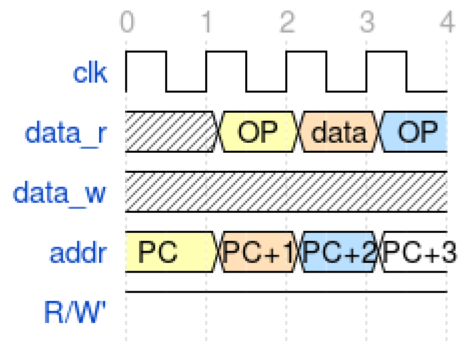

Figure 1.4: Two byte instruction timing

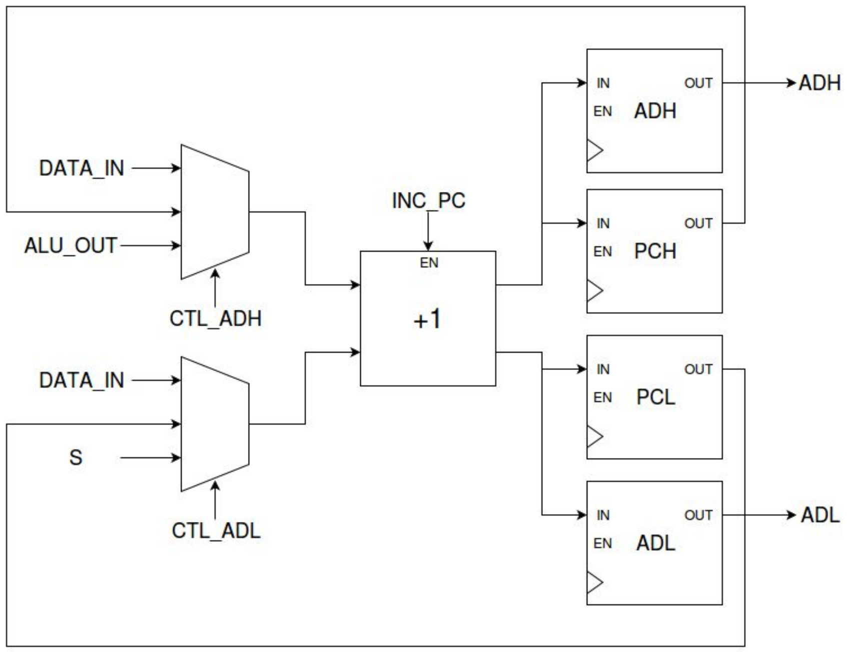

Figure 1.5: Address registers and program counter

Our processor will communicate with a 65kB block ram implemented as a module on the FPGA similar to the memory module implemented in lab 1. The module takes a 16 bit address as input as well as a write enable bit and, on each rising edge of the clock, puts one byte stored at the given address on an output which is sent to the data_in register in the processor.

Software Interface

We will implement a basic software interface to communicate between linux running on the ARM processor and our 6502 implementation running on the FPGA. On a high level, the software interface will allow the user to start and stop the processor, place 6502 assembly at a memory location that is addressable by the processor, and receive processor output (which can be printed to the screen by the user program). In this way the user can write a program in 6502 assembly, place the instructions in memory, run the instruction on the processor, and receive any output produced.

Milestones

Milestone 1:

- Create simulations using Verilator for each submodule separately. Control logic partially complete.
- Verify that each submodule produces proper values for hard coded inputs

Milestone 2:

- Finish simulations for each submodule.
- Simulate timing between submodules... DEBUG!
- Begin synthesizing modules on the FPGA

Milestone 3:

- Finish synthesizing modules on the FPGA
- Write software interface to communicate between user and processor

Final Presentation:

- Processor successfully executes 6502 assembly
- Software interface allows user to place assembly code at proper memory address
- User can start and stop processor and view output