Will Krska and Conor Naughton

EC551 Fall 2022

Lab 1

Our overall design is shown in the following block diagram:



As shown, the design can be broken down into three main segments, a VGA block, a datapath and an instruction read only memory (ROM). The following is a description of these three segments.

**Instruction ROM**

This block operates prior to the datapath beginning it’s execution and it’s purpose is to fill the memory with instructions. It only take the system clock and a reset signal as inputs and it outputs the address that instruction is to be sent to in memory as well as the data which is to be written to that address. It operates via a finite state machine, the first state of which checks to see if reset is enabled, then moves to a state which enacts a slight wait (16 clock cycles) then finally moves to the state in which the instruction writing occurs.

**Datapath**

Our design is a pipelined datapath which can be broken down into four stages. Between each stage are registers which hold the values which are output in that stage and are to be used by modules in the next stage and operate on each positive edge of the clock cycle. These registers take as input, most of the outputs of the modules which operate in that cycle as well as the clock and reset. Upon receiving a reset signal, all values would be zeroed out in these registers (this reset functionality is used in nearly all modules within our design). In addition, there is a finite state machine which runs prior to the datapath actually beginning operation which allows the instruction set to be loaded into memory. This is controlled using an ap\_start flag which begins the datapath.

Instruction Fetch

The first is an instruction fetch stage in which the fetch unit pulls the instruction from memory based on the present value of the program counter which resides within the seventh location of the register file. It then outputs the instruction as well as the new program counter based on whether the jump flag is being held high.

Instruction Decode

The next stage is the instruction decoder stage wherein the decoder module takes in the current instruction from the fetch unit and breaks it down into the operation and depending on the operation, either register or memory locations or an immediate value.

The register file also operates within this stage and takes in two read select as well as a write select and VGA display location for the user defined location which is to be displayed via VGA. The register file consists of 6 locations as well as the program counter each of which contain 16 bits of data. It also has two enable inputs for writing to register locations as well as writing to the PC and corresponding inputs for the write data and the new PC value. The read functions (two register locations as well as the PC update and the location which is going to be displayed via VGA) are performed asynchronously whenever the read values or the PC change. In addition, on every clock cycle, there is logic to check whether the reset flag is active in which case all register locations are rest to a value of 0.

The final module to operate in this stage is the memory unit which has two address inputs, one for the data which is being used for ALU operations and one for the instruction which is being fetched by the fetch unit based on the program counter value. It also takes as input the memory write enable flag as well as the address and data which is to be written if needed. It’s only outputs are the instruction and ALU data which are assigned immediately upon any change to those input addresses.

Execution Stage

The third stage is the execution stage which occurs solely in the ALU. The ALU takes as input the four bit operation code, as well as the data from the register file, the memory unit and potentially an immediate from the decoder which may be used based on the operation code. It’s outputs are the result of the operation which was taken in as input as well as two flags, jump and halt which indicate if those were the instructions received on that cycle. The ALU operates using a case statement which identifies the operation based on the decoder output and assigns the ALU output accordingly.

Writeback Stage

The final stage in our datapath is the writeback stage in which the output of the ALU is sent back to the register file and written to the address specified within the instruction. The key to this design working is that the read of the register file is performed asynchronously whenever the decoder unit outputs the instruction while the writeback is only performed on the positive edge of the clock.

**VGA Block**

Our VGA controller consists of a top module which takes in the 100MHz system clock and divides it down to 25MHz and feeds that to a VGA controller module which does the horizontal and vertical pixel counting. Both the pixel counters are triggered on the positive edge of the 25MHz clock. The counter is used to identify vertical and horizontal position and to ensure the pixels being generated are within the active pixels or drawing area. Once the counter moves past the drawing area, there is logic to restart the counter and to zero out the pixels while it restarts the count. Within the top module, the inputs are the system clock, a reset signal as well as the contents of a user defined register. To display those contents, there is a finite state machine which runs within a loop which extracts the bits of the register contents four bits at a time so that they can be matched up with a hex value. Once the equivalent hex is identified, a starting address for that hex digit is placed into a 32-bit variable used for containing the starting addresses for all four hex values to be displayed.

These initial addresses are used by an additional nested font ROM module which identifies the bits to be used by the VGA controller to represent a particular ASCII character. The top module takes the counters as the output of the controller and on every system clock edge enters a block where the first step is to identify which hex character, we are displaying based on the horizontal position as defined by the counter. Knowing that the register data is 16 bits, we split the horizontal range into quadrants to display the four hex characters required to represent that data. In addition, the ASCII characters are separated into 16 lines of binary digits and so we divide the vertical range by 16 and the next step is to identify which segment of the vertical drawing range we are in and set the address which will be extracted from the font ROM module appropriately (i.e if we are in the fifth of the sixteen vertical segments, the address will be four past the initial address of the appropriate hex digit). Finally, by taking the modulus of the horizontal counter, we identify where within the given hex digit we are and pull the particular bits from the ASCII line for display.