




## CSE490/590 Spring2011

## Project 2

*There are five options for the project 2. Please read each option carefully. The project 2 should be done individually. These options should be implemented with Verilog and your FPGA board. For using a keyboard as your input, you need to connect it to your FPGA board via the ps-2 port. If you choose option 4), you may have extra credits. Also if you choose option 5), please let us know in advance for approval.*

- 1) Design a baseball game using the four-digit seven-segment display. This is a small baseball game based on 4 numbers. The goal of the game is that a user needs to guess and get the correct four numbers. First, the user needs to input randomly four numbers in sequence (e.g., 5372) on the keyboard. You may add another way to generate four input numbers, which is that the user may use the “random” function in your Verilog code to generate 4-random numbers. After the user finishes entering 4 numbers, the four-digit segment display should show “PLAY” (i.e., ). Second, the user needs to guess the first number (in the above example, 5) of the 4 numbers. If the user guesses correctly and presses “5” on the keyboard, the first digit of segment display should show the first number “5” because it matched. In the next step, the user needs to guess the second number (in the above example, 3) of four numbers. If the user incorrectly guesses and presses “4”, the four-digit display should show “H” in the second digit (i.e., 5H) which means that it didn’t match. After that, the user can try up to 3 times per game. If the user fails over 3 times, the four-digit display should show “over” (i.e., ). When the user correctly guesses all 4-numbers successfully, the four-digit display should show “strike” (i.e., ). Since “strike” is a 5-letter word, the display should rotate “strike” as a text animation, e.g., “stri” first, followed by “trik” and “rike”. You are free to choose any other style for the animation.

Grading Criteria: Total 60 pts (40 pts functionality demo + 10 pts code + 10 pts documentation)

- a) Overall game (20 pts)
  - A. Keyboard functionalities (5 pts)
  - B. Game functionalities (saving the sequence & match the sequence) (15 pts)
- b) Displaying the text “play”, “over” and “H” (5 pts)
- c) .bit, .mcs and all .v files (5 pts)
- d) Displaying the text “strike” with rotation (10 pts)
- e) Code organization & implementation & proper comments (10 pts)
- f) Document organization & quality (please include a list of objectives and indicate “what you did” & “what you did not”) (10 pts)

- 2) Design a digital clock that uses the four-digit segment display of the board. The user needs to use a keyboard as an input. In addition, this digital clock has two more functions: i) alarm alert and ii) stop watch. First, the user should be able to enter the initial time (e.g., 08 20 or 20 20) by the keyboard. The 08 20 means 8:20am and 20 20 means 8:20pm. In the four-digit display, the leftmost two digits represent hours and the rightmost two digits represent minutes. The user should be able to set or change the time by pressing “m” on the keyboard. Then the four-digit display should start to blink while showing the current time. At this point, the user can enter the new

time by pressing 4 numbers in sequence on the keyboard. Note that when you set the initial time or change time, the display should show the time continually. Second, the user should be able to set the alarm (e.g., 09 00) by pressing “a” on the keyboard. Then the time on the segment display should blink. At this point, the user should be able to set the alarm by pressing four numbers on the keyboard. After setting the alarm, the LED should flash and the segment display should blink when the alarm goes off. Third, for the stopwatch functionality, “s” should be used. When the user presses “s” on the keyboard, the time should blink and the user should be able to input four numbers, minutes first then seconds. While entering the four digits for the stopwatch, the segment display should show the numbers entered. The leftmost two digits should show the minutes, and the rightmost two digits should show the seconds. Pressing “enter” on the keyboard should start the stopwatch. Pressing “esc” should stop the stopwatch.

Grading Criteria: Total 60 pts (40 pts functionality demo + 10 pts code + 10 pts documentation)

- a) Overall digital clock (14 pts)
  - A. Keyboard functionalities (5 pts)
  - B. Clock operation (9 pts)
- b) Setting or changing the time, i.e., “m” (7 pts)
- c) Alarm function, i.e., “a” (7 pts)
- d) Stopwatch, i.e., “s” (7 pts)
- e) .bit, .mcs and all .v files (5 pts)
- f) Code organization & implementation & proper comments (10 pts)
- g) Document organization & quality (please include a list of objectives and indicate “what you did” & “what you did not”) (10 pts)

- 3) Design an advanced calculator that uses the four digits segment display of the board. This can be an extension of the project 1. However, this calculator should support more operators, i.e., +, -, \*, /, % (mod), and ^ (exp). First, the keyboard should be used for input. The user can enter a new number by pressing “i” first then the number itself on the keyboard. When the user presses “i”, the LED should flash which means that it is ready for another input. You need to support calculations up to 10 operators (e.g.,  $6^3-2$ ,  $5*2\%9$ ,  $5+4/2$ ,  $4*5*5$ ,  $4*2/4$ ,  $6+3-2*9/3$ ,  $8/2-4*1+1*1+1-2+2-1+3$ ). The precedence should be (highest to lowest), i) \*, /, %, and ^ (left-to-right) and ii) +, - (left-to-right). The results should be shown on the four-digit display. You also need to display negative numbers (e.g., -1 or -2) on the segment display. When you press enter, this means that you finish entering the numbers and the results should be displayed. Second, when the user presses “r”, the number on the segment display should reset to zero, so that the user can start a new calculation. Third, the display should be able to display large numbers (i.e., 1000000, 250000) by rotating the number on the display. For example, you can display 1000345 by first showing 1000 followed by 0003, then 0034, and finally 0345. Please refer to the project option 1) about the rotation. Note that you also need to show negative power operation (e.g.,  $10^{(-9)} = 10^{-9}$  on the segment display). You will need to show the calculations up to 9999999 and all the entire numbers need to be shown by rotation. You need to implement all the functionalities.

Grading Criteria: Total 60 pts (40 pts functionality demo + 10 pts code + 10 pts

documentation)

- a) Overall calculator (20 pts)
  - A. Keyboard functionalities (5 pts)
  - B. Correct precedence (3 pts)
  - C. Advanced calculator operation (12 pts)
- b) Displaying the output number in rotation and negative power operation (15 pts)
  - A. Rotation (8 pts)
  - B. Negative power operation (7 pts)
- c) .bit, .mcs and all .v files (5 pts)
- d) Code organization & implementation & proper comments (10 pts)
- e) Document organization & quality (please include a list of objectives and indicate “what you did” & “what you did not”) (10 pts)

- 4) Design a 32-bit single-cycle MIPS processor supporting 12 instructions. The 12 instructions are LW (load word), SW (store word), BEQZ (branch if equal to zero), BNEZ (branch if not equal to zero), ADDI (add immediate), SUBI (subtract immediate), J (jump), JAL (jump and link), ADD (add), SUB (subtract), AND (and), and OR (or). Please refer to the lecture notes for the formats of these instructions. If certain bits in the instructions are not necessary, you are free to do anything, e.g., your design can just ignore them, flash the LED, or throw an error. There should be 16 registers supported. The instruction memory should be able to store up to 64 instructions (i.e., 256B). The data memory should be able to store up to 64 32-bit words (i.e., 256B). In addition, the CPU should operate in four modes. The first mode is *preload*, where the user can preload the memory with instructions and data using the keyboard by typing the type of memory (“i” or “d”), the address (2 hex digits), and finally the instruction or data value. Instructions and their necessary fields should be typed using the keyboard. For registers, the names should be typed (e.g., type “a-d-d-r-0-r-1-r-2” for  $R0 = R1 + R2$ ). Data values should be entered as 8 hex digits. The second mode is *execution*, where the user can start the execution of the instructions in the memory from the first instruction to the last instruction by pressing “enter” on the keyboard. The third mode is *lockstep*, where the user can execute instructions in the memory one instruction at a time by pressing the space bar on the keyboard. The fourth mode is *inspection*, where the user can examine the content in the memory by typing the type of memory (“i” or “d”) and the address (2 hex digits). The four-digit display then should show the corresponding value in hex. The user should be able to switch between the four modes by using the push buttons on the board.

There are a few things to note:

- i. R15 is used for JAL to hold  $PC + 4$ .
- ii. The byte ordering is little-endian (the least significant byte comes in the smallest address).
- iii. In general, both the instruction memory and data memory are byte-addressable. For example, if you enter an address in the inspection mode, the display should show the content of a byte, not 4 bytes, even if instructions and data are 32 bits. An exception is the preload mode, which we will detail next.
- iv. 32-bit instructions and data occupy 4 bytes and the memories are byte-

addressable. According to (iii), this means that we need to enter each byte at a time, which is not convenient for preloading the memory contents. Thus, only for the preload mode, a user should be able to enter 4 bytes as described above (e.g., for an instruction 'a-d-d-r-1-r-2-r-3' and for integer data '0-0-0-0-0-0-1-0'). The address the user enters should be the smallest address of the 4 bytes (e.g., '0' for entering addresses 0-3).

Grading Criteria: Total 70 pts (50 pts functionality demo + 10 pts code + 10 pts document)

- a) Preload mode (15 pts)
  - b) Execution & lockstep modes (20 pts)
  - c) Inspection mode (10 pts)
  - d) .bit, .mcs and all .v files (5 pts)
  - e) Code organization & implementation & proper comments (10 pts)
  - f) Document organization & quality (please include a list of objectives and indicate "what you did" & "what you did not") (10 pts)
- 5) If you desire, you may come up with your own project (write-up), but your idea must be approved by Prof. Steve Ko and TA. If you want to do project 2 with your own idea, please let us know in advance.

**If you have any questions, please feel free to send an email to [jangyoun@buffalo.edu](mailto:jangyoun@buffalo.edu) or [srwshah@buffalo.edu](mailto:srwshah@buffalo.edu)**