University of British Columbia
Electrical and Computer Engineering
EECE 353

Laboratory 3 - Digital Systems Design

Learning Objectives

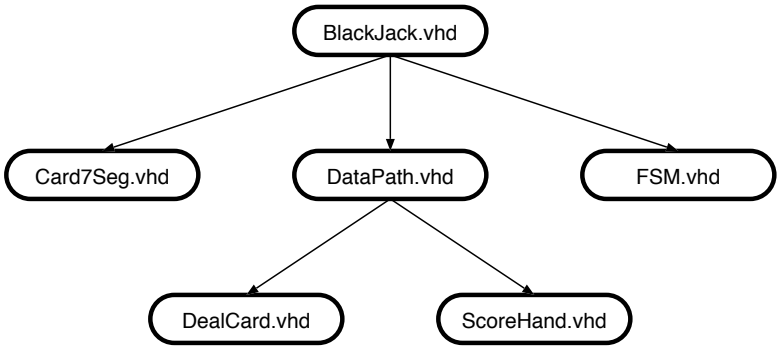
1. Design a non-trivial finite state machine and express it in VHDL.
2. Build complex combinational functions in VHDL.
3. Generate random numbers in hardware.
4. Interact with a clock.

Abstract

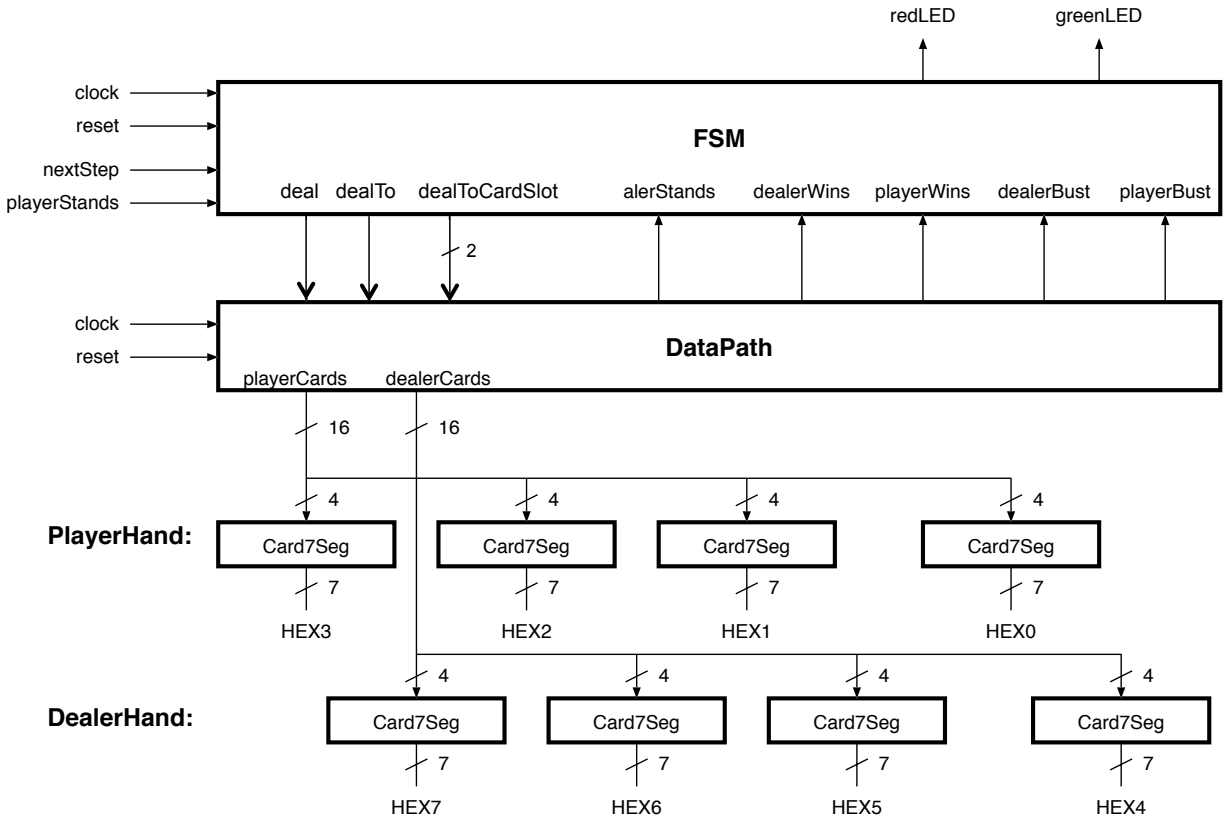
In this lab, you will design a one-player Blackjack card game using the DE2 board. The main part will be a complex FSM used to control the sequence of dealing cards to one player and the dealer. You will also develop a combinational function to drive the 7-segment display to indicate Jack, Queen, King and Ace cards, and a combinational function to count the value of a hand, where Aces can be either 1 or 11. As well, you will have a sequential part that determines the next card to be drawn. To solve this problem, you will use a variety of techniques in describing hardware with VHDL: FSMs, combinational logic, and variables.

Organization

You will create a hierarchical design, consisting of multiple VHDL files and design entities. The top-level filename you must use is **blackjack.vhd**, with lower-level entities organized as described below.



The top-level file will be provided to you. You must create the other files. A block diagram of the top-level design, which will be created inside **blackjack.vhd**, is shown below.



Not shown in the diagram: The **DataPath** will contain two instances of the **ScoreHand** entity (one for the player and one for the dealer), and one instance of the **DealCard** instance.

Requirements

The functional specification of the lab requirements is as follows:

1. Each card is represented by a 4-bit value. Ignore suites (hearts, diamonds, clubs, spades). Each value indicates:
   - 0 is "no card"
   - 2 through 10 are themselves
   - 1 is Ace, 11 is Jack, 12 is Queen, and 13 is King
   - 14, 15 are not used
2. The dealer and player can each hold a hand of up to 4 cards. No more than 4 cards are allowed per player. To remember an entire hand, you will need 16 bits. You will need separate register memories for both player and dealer hands.
3. Display the cards in a hand on the 7-segment display. At all times:
   - Hex3-Hex0: display the player hand
   - Hex7-Hex4: display the dealer hand
4. Each card is a 4-bit number that is displayed as a singe digit on the 7-segment display. Write **card7seg.vhd** to convert a 4-bit card value into an appropriate LED display pattern as follows:
   - The value 0 is "no card" and should be displayed as a blank (all LEDs off)
   - Ace as "A", 10 as "0", Jack as "J", Queen as "q", and King as "H"
   - 2 through 9 as themselves, making sure the numeral 9 appears differently than "q"

5. As a player, during your turn you can request either "hit" for another card, or "stand" for no more cards.
   - SW0 = '0' indicates "hit". SW0 = '1' indicates "stand".
6. The dealer must always follow the house rules (regardless of the player's hand):
   - If the current dealer hand totals 16 or less, the dealer takes another card
   - If the current dealer hand totals 17 or more, the dealer stands
7. Counting a hand is done in **scorehand.vhd** as a combinational circuit. You will need two instances of this component, one to count the dealer hand and one to count the player hand. This component takes a complete hand (16-bit value) as inputs, counts the cards, and produces a 5-bit total for the output. There are two additional 1-bit outputs: "stand" indicates the hand >= 17, and "bust" indicates the hand > 21. Note that "stand" will only be used by the dealer's instance to drive it's A.I. This routine must be written entirely in combinational logic (with no clock). To simplify things, the use of VARIABLE types is suggested. When counting, use a larger number of bits internally (6 bits is enough to represent the maximum possible sum of 44), but the final result should be reduced to 5 bits. Remember that each Ace takes on the value of '1' or '11', as needed, to give the maximum possible hand value that is still less than or equal to 21. **The logic for counting is complex, so make sure you give yourself enough time to think about it and do plenty of testing!** The use of a testbench and ModelSim is strongly encouraged.
   - The Jack, Queen, and King each represent a value of 10.
   - Each Ace represents the value of 11 or 1, individually as needed, to give a maximum possible hand total value that is less than or equal to 21.
   - If the final total hand value is >= 17, the **stand** output should be '1'.
   - If the final total hand value > 21, the hand is bust. In this case, force the hand value to 31 (all outputs will be '1'; this helps simplify testing if you are displaying on LEDs).
8. All of your sequential logic (FSMs, registers) **must use CLOCK_50 as the only clock**. **Use KEY0 as a combinational input** where pressing the key tells the game to take the "next step". As KEY0 is pressed, the game makes forward progress. Your FSM may sequence through several clock cycles to accomplish the current step (state), and then pause until the user presses KEY0 again. Usually, KEY0 means a new card will be given out (either to the dealer or the player).
9. The required sequence of steps are:
   a. **Start.** Give player a card.
   b. Give dealer a card.
   c. Give player a second card. If player stands, go to **DealersTurn**.
   d. Give player a third card. If player stands, or goes bust, go to **DealersTurn**.
   e. Give player a fourth card.
   f. **DealersTurn**. Give dealer a second card. If dealer stands, go to **Winner**.
   g. Give dealer a third card. If dealer stands, or goes bust, go to **Winner**.
   h. Give dealer a fourth card.
   i. **Winner.** Decide winner:
      - If both go bust, there is no winner, go to **EndGame**.
      - If dealer >= player, or player is bust, go to **DealerWins**.
      - If dealer < player, or dealer is bust, go to **PlayerWins**.
   j. **DealerWins.** Turn on LEDR[17:0]. Go to **EndGame.**
   k. **PlayerWins.** Turn on LEDG[7:0]. Go to **EndGame**.

l. **EndGame.** Wait forever.

10. To give the dealer or player a new random card, we will use a few simple tricks. First, assume we are dealing from an infinite deck, so it is equally likely to generate any card, no matter which cards have already been given out. Second, assume that when the player presses the "next step" key, an unpredictable amount of time has passed, representing a random delay. During this random delay interval, your **dealcard.vhd** should be continuously counting from the first card (Ace=1) to the last card (King=13), and then wrapping around to Ace=1 at a very high rate (eg, 50MHz). To obtain a random card, we simply sample the current value of this counter during one clock cycle after a random delay, ie when the user presses "next step".
11. Use KEY3 to reset the game and go to **Start** at any point.

DO NOT USE KEY0 AS A CLOCK SIGNAL! Your design must use only CLOCK_50 as the clock that controls all memory elements and advances you from state to state. ***You will lose marks if you do not follow this instruction.***

A Few More Notes About the FSM and Datapath

In the suggested structure, the FSM controls the datapath with with three control signals: **deal, dealTo,** and **dealToCardSlot**. The idea here is that **deal** specifies whether a card should be dealt during the current state ('1' to deal, '0' to not deal). When dealing, **dealTo** specifies whether to deal to the player ('1') or to the dealer ('0'), and **dealToCardSlot** is a 2-bit value specifying the card slot in the hand to which the card is being dealt. For example, since there can be up to 4 cards per hand, say we are dealing to the player to the third card slot in the player's hand: **deal** = '1', **dealTo** = '1', **dealToCardSlot** = '10'.

The FSM's inputs include the a set signals suppled by the top-level (**clock, reset, playerStands, nextStep**), and a set of status signals produced by the Datapath (**dealerStands, dealerWins, playerWins, dealerBusts,** and **PlayerBusts.** These are all generated by the instances of **ScoreHand)**.

Sample Design and Testing Process

Develop your solution by breaking it down piecewise. Build a small, simple piece and test it. Continue building and testing each piece one-by-one as you go along (do not leave all testing until the end!).

1. Write the **Card7Seg** decoder entity and architecture in **card7seg.vhd.** Instantiate one instance of **Card7Seg** inside the top-level **BlackJack** entity and connect it to SW[3:0] and HEX0. Compile and test for all card values. 1mark
2. Create the **DataPath** entity inside **datapath.vhd**. Don't worry about the insides of the architecture in this step. Instantiate the **DataPath** inside **BlackJack.** Instantiate all copies of **Card7Seg.** Connect the instances of **DataPath** and **Card7Seg**.
3. Write the **DealCard** entity and architecture in **dealcard.vhd** to continuously "shuffle" the deck and provide a new card every clock cycle on its output. Instantiated **DealCard** in the **DataPath** architecture. You may want to temporarily connect the **DealCard** instance in such a way as to be able to view its output on HEX0. Test that **DealCard** properly produces a random card sequence when KEY0 is pressed (**nextStep)** . 1mark

4. Write a simplified **ScoreHand** description in **scorehand.vhd** by treating Ace/Jack/Queen/King as having scores of 10/11/12/13 (this is to keep things simple at first; you will fix it later). At first, connect it to sixteen switches SW[15:0] and verify that it works as expected. While testing, continuously display the player's hand on HEX[3:0], and the hand total on LEDR[4:0]. 1mark
5. Modify **blackjack.vhd** to add the simple FSM. Create a simplified FSM in **fsm.vhd**, having it build dealer and player hands of 2 cards each. Display both hands on 7-segment displays. Display the score of the player and dealer hands on LEDR[4:0] and LEDG[4:0]. At this point, don't worry about a winner, or a hand going bust. The FSM should wait for KEY0 to be pressed once to deal each card, otherwise you will not get a random card. It should also respond to KEY3, which will reset the game. Compile and test. 2marks

You should aim to complete the above during the first week.

6. Modify **scorehand.vhd** to count Jack, Queen, and King as 10. If hand total exceeds 21, force hand total to be 31. Calculate "stand" and "bust" outputs; for testing this step only, temporarily display "stand" and "bust" on any green LED. Compile and test. 1mark
7. Modify **scorehand.vhd** to count Ace as 11 or 1, as required to achieve maximum score of 21 without bust. For example, a hand containing two Aces could treat one Ace=1 and another Ace=11 to get a perfect score of 21, whereas a hand with two Aces and a 10 would treat both Ace=1 to get a score of 12. 1mark
8. Connect your **DataPath** and **FSM** instances to each other(in **BlackJack**), and modify them so that they work together to run through the operations of the game.
9. Modify **blackjack.vhd** and/or **fsm.vhd** to calculate and display the winner, reset the hands, and restart. Indicate the winner as required using LEDG. Compile and test.
10. Modify **blackjack.vhd** and/or **fsm.vhd**, checking "stand" and "bust" for player and dealer. The player's "stand" signal comes from SW0, the dealer's comes from **scorehand.vhd**. Compile and test.
11. Make sure all requirements are met. 3 marks for final integration

Lab Demonstration and Marking

Demonstrate your BlackJack game to the TA.

There is no Challenge Task for this lab. If you can demo the correct behaviour of the entire system as a single unified design, you will receive full marks. 10/10

If you cannot demo a full solution, you can demo individual components or specific portions of the lab to receive part marks. The steps in the "Sample Design and Testing Process" have marks associated with them. It is up to you to convincingly demo the completion of each step to the TA within your time-slot to receive the marks.

VHDL Specifications

Use the top-level input/output specifications provided in the **blackjack.vhd** file. Do not add or remove any **top-level** ports. Start with the supplied blackjack.vhd skeleton file to see recommended portlists for the card7seg, datapath, and fsm components.