**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**
**UNIVERSITY OF BRITISH COLUMBIA**
**CPEN 311 – Digital Systems Design**
**2016/2017 Term 1**

# Lab 1: Simple Datapath and Controller: Baccarat

Working week: September 12-16, 2016
Marking week: September 19-23, 2016

**Work in Pairs: Choose your partners during the lab period on Sept 13th or 15th. If you do not have a partner, your TA can help pair people up during your lab period Sept 13th or 15th. Do not wait until Sept. 20th to partner up.**

In this lab, you will become familiar with the software and hardware we will be using in the labs, and use the hardware and software to implement a simple digital datapath. You will also be re-acquainted with VHDL or Verilog, which most of you will have been introduced to in CPEN 211 or CPEN 312.

We will be using two pieces of software for most of this course: Quartus II, which is produced by Altera, and ModelSim, which is produced by Mentor Graphics. There are several versions of ModelSim available; we will be using one that has been modified by Altera for use with Quartus II. You can download these programs (they are free), or use them on the departmental computers in the lab.

Like all labs in this course, this lab will span two weeks. The first week is intended to be a work-week, where you spend the three hours in the lab working on the tasks. A TA will be available to help you if you run into problems. The second week is primarily for marking (you should not expect help from the TA during the second week). You may have to do some work on your own (at home or in the lab outside of lab hours) as well. You most certainly can not finish the lab if you don't start until the marking week.

The end result of this lab will be a Baccarat engine. Baccarat is a card game played in casinos around the world. Baccarat is for "high rollers" -- James Bond is a fan of Baccarat. Designing this Baccarat engine will help you understand how a simple datapath can be constructed and controlled by a state machine; this is the foundation of all large digital circuits.

It may be tempting to write a software-like specification and hope that the synthesis tools can synthesize the design to hardware. It is unlikely that this approach would be successful. In this document, we will consider the circuit at the low-level hardware level, to ensure that you understand how the hardware works. This handout will take you through the steps needed to construct the circuit, starting from the basic blocks, and ending (hopefully) in a working circuit on your Altera board.

## PHASE 1: GETTING YOUR ENVIRONMENT READY

TASK 1.1: The first step is to install Quartus II and the Altera version of ModelSim on your home PC or laptop, or find a place you can run the software (it will be available on most of the computers available throughout the department). You probably already have the software installed from last year. If not, you have three options:

1. Install the software natively on your own Windows PC or laptop (this is likely what you did last year). You can find instructions in the document "Digital System Design using Altera Quartus II and ModelSim" which is on the Connect site. As described in the handout, the version of the software you install depends on the board you have. **If you are using a DE2 board, you should use Version 13.0 SP1** (*do not use Version 13.1 or later*, *since it does not support the device you will be targeting*). **If you are using a DE1-SoC or a DE0-CV board, you should use Version 15 or 15.1** (*do not use Version 13 or earlier*, *since it does not support the device you will be targeting*).
2. Alternatively, you can create a Virtual Box virtual machine. Inside the virtual machine, you would run Windows and Quartus II. This is especially useful for Mac users that are not running

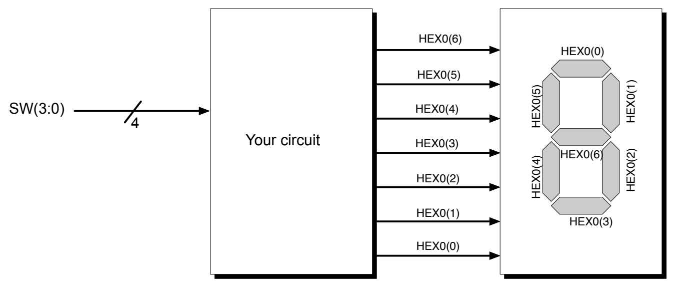
Windows natively. If you wish to set up such a virtual machine, see https://www.youtube.com/watch?v=tGrGsIDDJIo This video assumes you have a DE1-SoC; if you are using a DE2, you will need to make sure you install version 13.0 SP1 as described above. I have also had luck running Quartus II under Parallels, but we are not providing a tutorial for this option.

3. If none of these methods work for you (everyone's laptop has a slightly different configuration, and the class is too big to provide technical support for everyone's configuration), you can use the departmental computers which have Quartus II installed.

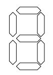
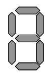
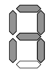
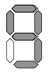
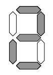
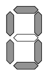
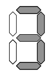
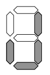
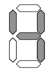
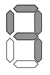
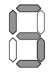
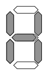
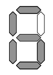
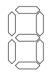
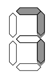
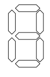
TASK 1.2: Work through the rest of the "Digital System Design using Altera Quartus II and ModelSim" tutorial document located in the Lab 1 folder of Connect. There are some download files for this tutorial, which you can find in Lab1->Download Files->Task1.2 on the Connect site (these files are all in VHDL; even if you are completing the course using Verilog, you can do this tutorial with VHDL files, since the overall flow is identical). This will re-introduce you to Quartus II and ModelSim, which are the tools we will use throughout the course. Even if you have used Quartus II and Modelsim, you might have forgotten some of the details, so it will save you a lot of time if you work through the tutorial carefully.

## PHASE 2: CREATE A SEVEN-SEGMENT LED DRIVER [2 marks]

To get you started quickly, in this Phase, you will create the simplest interesting combinational circuit I can think of. It will also be a component in the Baccarat game that you will create in Phase 3.



You will design a combinational block with four inputs and seven outputs, as shown to the right. In this phase, the four inputs will be connected to switches 3 to 0 on your board, and the seven outputs are connected to a single seven-segment display on your board.

| SW(3 downto 0) | Output | SW(3 downto 0) | Output |
|---|---|---|---|
| 0000 (blank) |  | 1000 |  |
| 0001 (Ace) |  | 1001 |  |
| 0010 |  | 1010 |  |
| 0011 |  | 1011 (Jack) |  |
| 0100 |  | 1100 (Queen) |  |
| 0101 |  | 1101 (King) |  |
| 0110 |  | 1110 (not used) |  |
| 0111 |  | 1111 (not used) |  |

The circuit operates as follows. The four inputs can be interpreted as representing a number between 0 and 15. Each number between 1 and 13 represents a specific card value in a deck of cards, as follows: Ace=1, number cards are represented as themselves, Jack=11, Queen=12, King=13. An input of 0 represents "no card" and input values of 14 and 15 will not be used (you can assume they never occur); for the purposes of this Phase, you should display a blank if they do occur.

The output is a set of values that will display the value of the card on the seven segment display, as shown in the diagram to the right. Each segment in the display is controlled by one of the output lines, as shown in in the diagram above. Note that the lines driving the seven-segment display are *active-low*. That means that a 0 turns on the segment and a 1 turns the segment off (this is the opposite of what you might expect).

Download the VHDL or Verilog stub on Connect, and add your code to implement this combinational circuit. Create a project in Quartus II, import the pin assignments, compile your design, download it to the board, and test your design. Remember to download the correct pin assignment file for the board you are using.

You do not need to demonstrate this to the TA if the rest of the lab works, but save the project so you can get part marks if the rest of the lab does not work.

## PHASE 3: BACCARAT! [7 marks]

Task 3.1: Learn how to play Baccarat:

The game of Baccarat is simple. There are various versions, but we will focus on the simplest, called Punto Banco, which is played in Las Vegas and Macau. The following text will describe the algorithm in sufficient detail for completing this lab; if you need clarification or more information on any point, there are numerous tutorials on the web (Google is your friend).

The game proceeds as shown in this algorithm:

```
Two cards are dealt to both the player and the dealer face up (first card to the player,
        second card to dealer, third card to the player, fourth card to the dealer).
The score of each hand is computed as described at the bottom of this page.
If the player's or dealer's hand has a score of 8 or 9 {
    the game is over (this is called a "natural") and whoever has the higher score wins (if
                                        the scores are the same, it is a tie).
} else {
    if the player's score from his/her first two cards was 0 to 5 {
        the player gets a third card
        the banker may get a third card depending on the following rule:
                i.   If the banker's score from the first two cards is 7, the banker does not
                     take another card
                ii.  If the banker's score from the first two cards is 6, the banker gets a
                     third card if the face value of the player's third card was a 6 or 7
                iii. If the banker's score from the first two cards is 5, the banker gets a
                     third card if the face value of the player's third card was 4, 5, 6, or 7
                iv.  If the banker's score from the first two cards is 4, the banker gets a
                     third card if the face value of player's third card was 2, 3, 4, 5, 6, or 7
                v.   If the banker's score from the first two cards is 3, the banker gets a
                     third card if the face value of player's third card was anything but an 8
                vi.  If the banker's score from the first two cards is 0, 1, or 2, the banker
                     gets a third card.
    } else {   // the player's score from his/her first two cards was 6 or 7
        the player does not get a third card
        if the banker's score from his/her first two cards was 0 to 5 {
            the banker gets a third card
        } else {
            the banker does not get a third card
        }
    }
    the game is over. Scores are computed as below. Whoever has the higher score
                wins, or if they are the same, it is a tie.
```

In the above algorithm, the score of a hand is computed as follows:

1. The *value of each card* in each hand is determined. Each Ace, 2, 3, 4, 5, 6, 7, 8, and 9 has a value equal the face value (eg. Ace has value of 1, 2 is a value of 2, 3 has a value of 3, etc.). Tens, Jacks, Queens, and Kings have a value of 0.

2. The score for each hand (which can contain up to three cards) is then computed by summing the values of each card in the hand, and the *rightmost digit (in base 10) of the sum is the score of the hand*. In other words, if Value1 to Value 3 are the values of Card 1 to 3, then

   **Score of hand** = (**Value1 + Value2 + Value3**) mod 10

   If the hand has only two cards, then Value3 is 0. You should be able to see that the score of a hand is always in the range [0,9].

It is interesting to note that in this version of the game, all moves are automatic (the player does not have to make any decisions!). The version played in Monte Carlo is slightly different, in that a player can choose whether or not to take a third card. We will not consider that here.

## Task 3.2: Understand the overall functionality your circuit should implement

First, consider the behaviour of the Baccarat circuit from the user's point of view. As shown in the figure below, the circuit is connected to two input keys, a 50MHz clock, and the output of the circuit drives six seven-segment LEDs and ten lights.

The game starts by asserting the reset signal (KEY3) which is active low. The user can then step through each step of the game (deal one card to the player, one to the dealer, etc) by pressing KEY0 (this will be referred as **slow_clock** in this document). The exact sequence of states will be described in Task 3.3. As the cards are dealt, the player's hand is shown on HEX0 to HEX2 (one hex digit per card … remember each hand can contain up to three cards) and the dealer's hand is shown on HEX3 to HEX5. The current score of the player's hand will be shown on lights LEGR3 to LEGR0 (recall that the score of a hand is always in the range [0,9] and can be represented using four bits, and the current score of the dealer's hand will be shown on LEGR7 to LEGR4. We use lights to display the binary version of the score, since the DE1-SoC and DE0-CV only have six hex digits.

There is also a 50MHz clock input; this is used solely for clocking the **dealcard** block which deals a random card. This will be described further in a subsequent task.

At the end of the game, red lights 8 and 9 will indicate the winner: if the player wins, light LEDR(8) goes high. If the dealer wins, light LEDR(9) goes high. If it is a tie, both LEDR(8) and LEDR(9) go high. The system then does nothing until the user presses reset, sending it back to the first state to deal another hand.

Notice that, other than cycling through the states using KEY0 (the slow clock), the user does not need to do anything. This is consistent with the description of the game above.

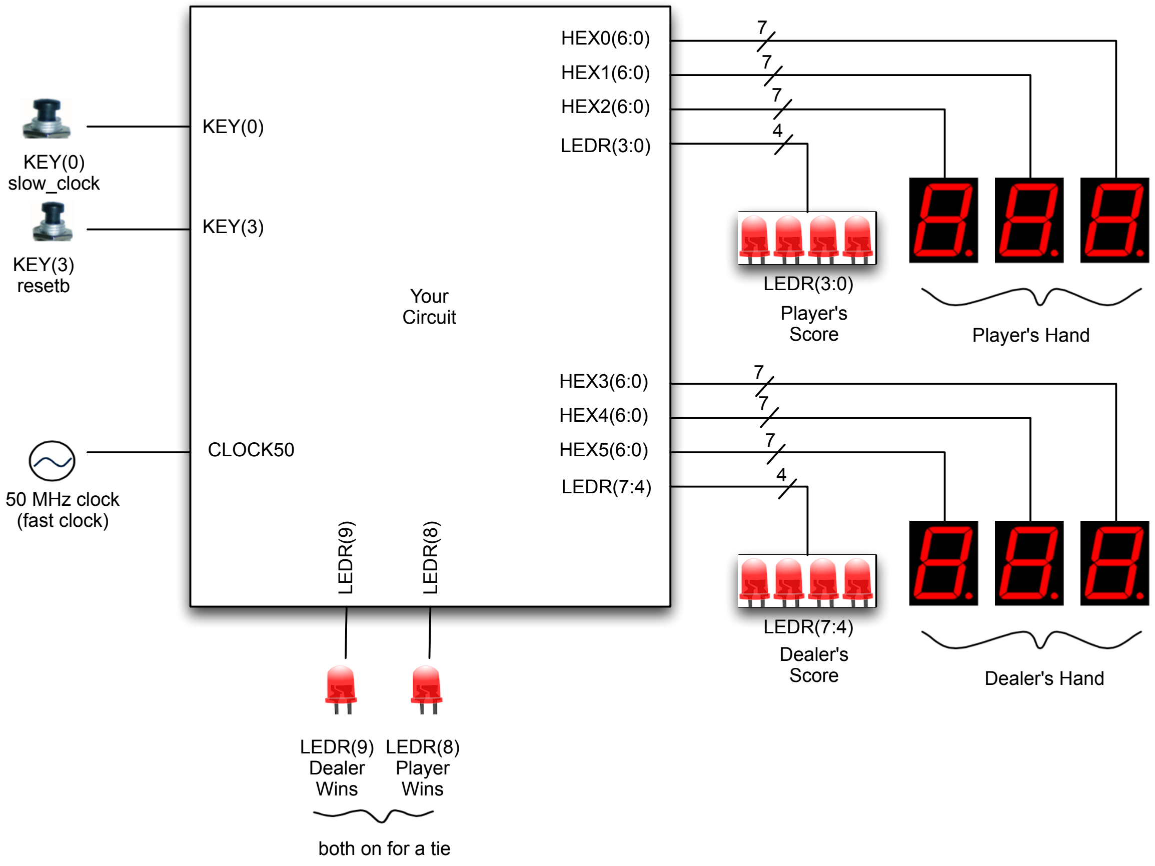


Figure 1: Overall diagram of your circuit from the user's point of view

Task 3.3: Understand the Components of our circuit:

The circuit consists of two parts: a state machine and a datapath. The datapath does all the "heavy lifting" (in this case, keeping track of each hand and computing the score for each hand) and the state machine controls the datapath (in this case, telling the datapath when to load a new card into either the player's or dealer's hand). The overall block diagram is shown on the next page.

First consider the datapath, which consists of everything *except* the statemachine block in the block diagram. There are a number of subcircuits here, and each will be described below:

**dealcard:**

To deal random cards, we need a random number generator. Random numbers are difficult to generate in hardware (can you suggest why?). 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 been given out (casinos try to approximate this approach this by using multiple decks). 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, the subcircuit described in **dealcard.vhd** will 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 when the user presses the "next step" key.[1]

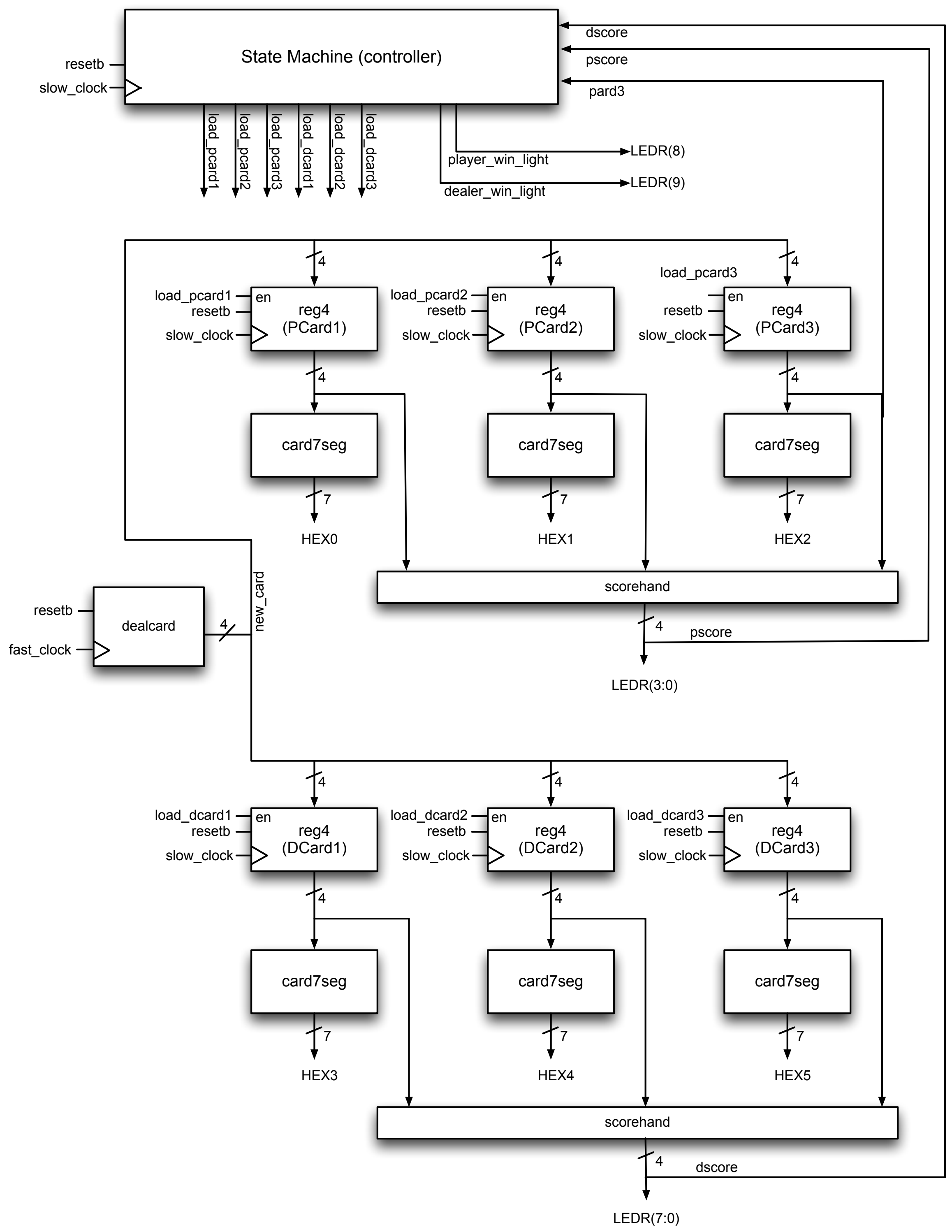
To save you time, we have written **dealcard.vhd** or **dealcard.v** for you. Download **dealcard.vhd** or **dealcard.v** from the Connect site and examine it. This block has two inputs (the fast clock and a reset) and one output (the card being dealt, represented by a number from 1 to 13 as described above). This circuit is essentially a counter. Be sure you understand it before moving on.

**<u>reg4:</u>**

Each card in each hand is stored in a reg4 block, which is a 4-bit wide register (set of 4 D Flip-flops). The upper three reg4 blocks store the player's hand, and the lower three reg4 blocks store the dealer's hand (recall each hand can have up to three cards). Each card is stored as a number from 1 to 13 (Ace=1, number cards are represented as themselves, Jack=11, Queen=12, King=13). We will not store the suit information for each card (the suit of a card does not matter in Baccarat). If a position in the hand does not have a card, we store a 0 to represent "no card". As an example, if the player's hand consists of a 5 and a Jack (and no third card), PCard1 would contain the number 5, PCard2 would contain the number 11, and PCard3 would contain the number 0 (no card). Note that since there are 14 different possible values (including 0), four bits in each register is sufficient.

Each register is clocked using **slow_clock** (which is connected to KEY0 and toggled by the user). On each rising clock edge of **slow_clock**, if the enable signal (for PCard1 the enable signal is called **load_pcard1**) is high, the value from **dealcard** is loaded into the register. The register also contains an active-low reset signal (which is connected to KEY3); when this is low, the value in the register goes to 0.

[1] Note that our circuit has two clock domains: one domain consists of circuitry clocked by the 50MHz clock and one domain consists of circuitry clocked by the slow_clock (which is connected to SW0). In general, most very large digital circuits have multiple clock domains. It is dangerous to connect the output from one domain to the input of another domain (as we do for the output of dealcard). Doing so can lead to something called metastability. This won't be a problem in this lab, because the frequency of the slow_clock is so slow. We will talk about this extensively later in the course. For now, you don't need to worry about it.



Block Diagram of your circuit

**card7seg:**

This is the block you wrote in Phase 2. As you recall, this is a simple combinational circuit with a single 4-bit input (the value of a card encoded as above) and 7 outputs that drive a HEX according to the following pattern:

- The value 0 is "no card" and should be displayed as a blank (all HEX segments off)
- 1 is displayed as "A", 10 is displayed as "0", Jack as "J", Queen as "q", and King as "H"
- 2 through 9 are displayed as themselves, making sure the numeral 9 appears differently than "q"

You can use the block from Phase 2 directly, however, the inputs now come from registers rather than switches.

**scorehand:**

This is a simple combinational circuit that takes the value of three cards and computes the score of that hand. Recall that the score of a hand is computed as follows:

a. The value of each of the three cards in each hand is determined. Each Ace, 2, 3, 4, 5, 6, 7, 8, and 9 has a value equal the face value (eg. Ace has value of 1, 2 is a value of 2, 3 has a value of 3, etc.). Tens, Jacks, Queens, and Kings have a value of 0. If fewer that three cards are in the hand, the missing positions are 0.
b. The values of the cards in each hand are summed, and the *rightmost digit (in base 10) of the sum is the score of the hand*. In other words, if **Value1** is the value of the first card, **Value2** is the value of the second card, and **Value3** is the value of the third card, then

**Score of hand** = (**Value1** + **Value2** + **Value3**) mod 10

You should be able to see that a hand can have a score in the range [0,9], thus 4 bits are sufficient for the output of this block.

**statemachine:**

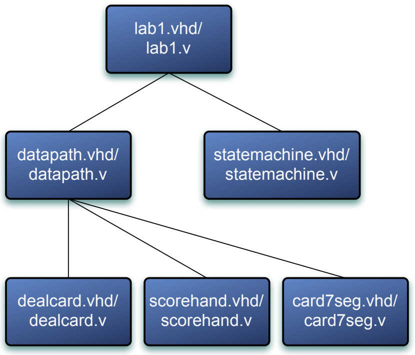
The state machine is the "brain" of our circuit. It has an active-low reset (called **resetb**) and is clocked by **slow_clock** (which is connected to KEY0). On each rising edge of **slow_clock**, the state machine advances one step through the algorithm, and asserts the appropriate control signals at each step. In this circuit, the control signals that the state machine controls are **load_pcard1, load_pcard2, … load_dcard3**. When it is time to deal the first card to the player, the state machine asserts **load_pcard1**, which as was described above, causes the first Reg4 block to load in a card (from the output of the **dealcard** block). During the cycle in which **load_pcard1** is 1, all other **load_pcard** and **load_dcard** signals are 0, so that no other positions in either hand are updated. As the algorithm progresses, the state machine will generate the other control signals to be asserted at the appropriate times.

As should be evident from the earlier discussion, the card drawing pattern depends on the dealer score and the player score (these are used to determine whether a third card is necessary) as well as the player's third card (this is used to determine whether the dealer should receive a third card, as described in the rules in Task 1). Therefore, **pcard3, pscore**, and **dscore** are inputs to the state machine.

## Task 3.4: Write the VHDL or Verilog code:

In this task, you will create the design in either VHDL or Verilog. Your design should follow the hierarchical design approach shown below. If you have forgotten how to create a hierarchical design, review the notes in Slides Set 2. To get you started, stubs for each of the files can be downloaded from the Connect site. Be sure to start with these, so that your interfaces for each module are correct. The reg8 block is not shown in the diagram; you can either create a new module to describe a four-bit register, or write it directly into datapath.vhd/datapath.v (your choice, either will work).

To help you, we are giving you **dealcard.vhd**/**dealcard.v** and **lab1.vhd**/**lab1.v** on the Connect site.



Design Hierarchy

The hardest part of this lab is getting the state machine right. I strongly urge you to draw (on paper) a bubble diagram showing the states, the transitions, and outputs of each transition. You are strongly urged to make sure you have your state machine bubble diagram done by the end of your Working Week lab. If you are unclear how to do this, be sure to discuss with your TA during the lab period. When drawing your diagram for your state machine, make sure that you cover all of the possible input conditions.

Task 3.5: Test your design on a DE2, DE1-SoC, or DE0-CV board:

Test your design by downloading it to the board. Be sure to run through enough scenarios to ensure that each branch in the algorithm is working correctly. Demo the working circuit to the TA. Remember that you should treat your TA as a client, and it is up to you to come up with an appropriate demo that shows that your design works. The TA will ask questions to gauge your understanding of the lab.

Phase 3 is worth, in total, 7 marks. The marking scheme is 4 marks for the datapath and 3 marks for the state machine. If your design works, you do not need to demo the datapath and state machine separately. However, if you are unable to get a working design, you should prepare to demo as many subunits as possible to the TA, either on the board or in simulation so that you can be awarded part marks. Note that demoing a working design is not sufficient; if you can not answer questions from the TA regarding your design, you will *not* get full marks.

**Remember to upload your .vhd or .v files to Connect after you complete your demo**. If you forget this, your mark for the lab will be adjusted to 0.

## Challenge Task: (1 mark)

Challenge tasks are tasks that you should only perform if you have extra time, are keen, and want to show off a little bit. This challenge task is only worth 1 mark, but is far more work than the other tasks in this lab. If you don't demo the challenge task, the maximum score you can get on this lab is 9/10 (which is still an A+). Answers to Challenge Tasks will not be distributed.

As described earlier, in casinos, you can bet money that the player will win, the dealer will win, or the two will tie. Modify your design so that the user can place bets, and win or loose money. The user should be able to specify the amount of the bet, and the type of bet (dealer win, dealer loose, or tie). The circuit should keep track of the money the player has, calculate and add the payoff if the player wins, and charges the player's balance if the player looses (you can look up payoff rates on-line). Since your board has a limited number of lights and switches, you will have to make judicious use of your I/O (you can think of the best way to do this).