EECE 353: Digital Systems Design

Introduction to Lab 3:

BlackJack Game!

Lab 3

In this lab, you will

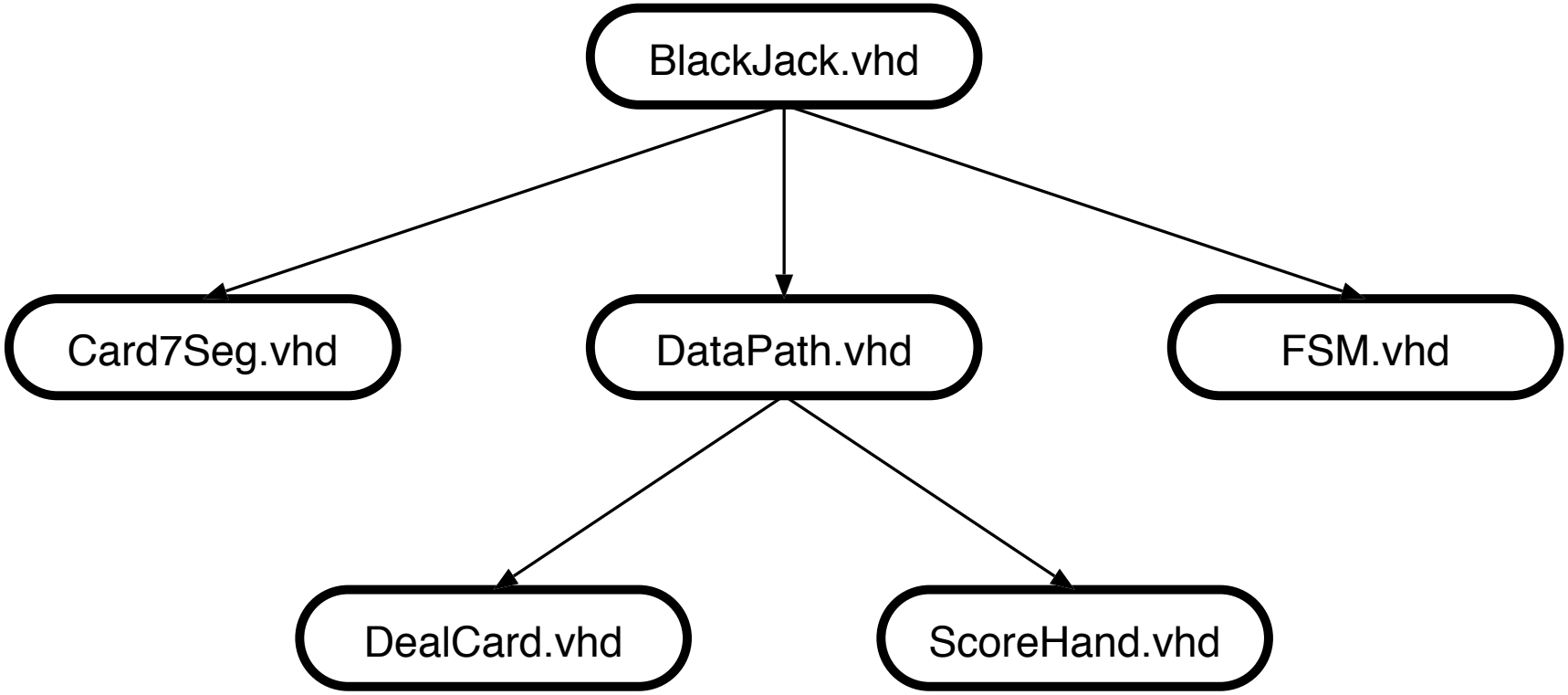
- Design a non-trivial finite state machine in VHDL
- Build complex datapath in VHDL
- Generate random numbers in hardware

Final goal: A circuit that implements the card game BlackJack

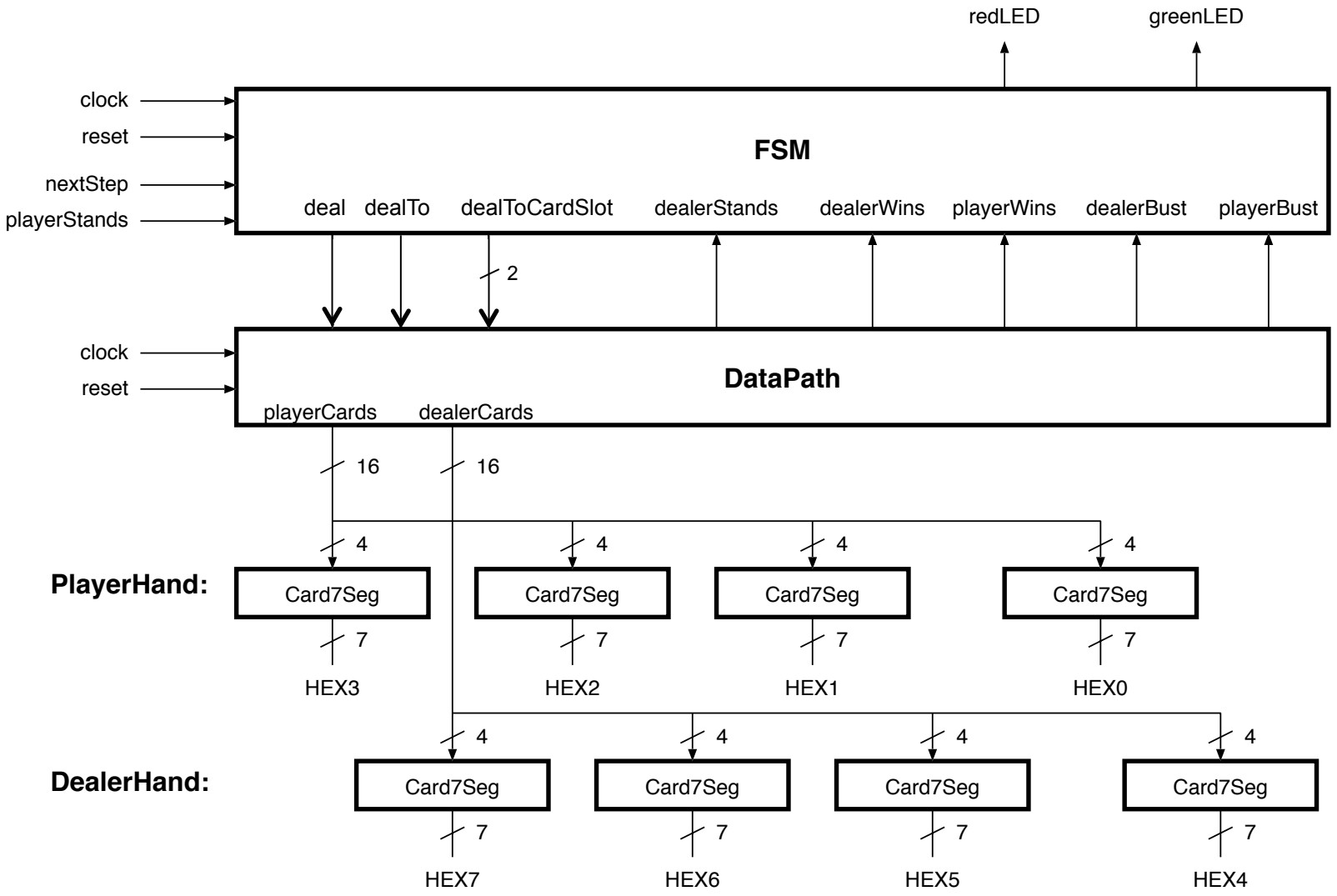
Gameplay

1. **START**: Give player a card
2. Give dealer a card
3. Give player a second card. If player "stands", go to **DEALER'S TURN.**
4. Give player a third card. If player "stands" or goes "bust", go to **DEALER'S TURN**.
5. Give player a fourth card.
6. **DEALER'S TURN:** Give dealer a second card. If dealer stands, go to **WINNER**.
7. Give dealer a third card. If dealer "stands" or goes "bust", go to **WINNER**.
8. Give dealer a fourth card.
9. **WINNER:** Decide winner:
   - If both go "bust", no winner. Go to **ENDGAME**
   - If dealer score >= player score or player busts, go to **DEALER WINS**
   - If dealer score < player score or dealer busts, go to **PLAYER WINS**
10. **DEALER WINS.** Turn on Red LEDs. Go to **ENDGAME.**
11. **PLAYER WINS**. Turn on Green LEDs. Go to **ENDGAME.**
12. **ENDGAME.** Wait forever.

VHDL Files


BlackJack.vhd
Card7Seg.vhd
DataPath.vhd
FSM.vhd
DealCard.vhd
ScoreHand.vhd

Inside blackjack.vhd



Game only progresses when player presses the *nextStep* switch

Card Encoding

- Use 4-bits to represent a single card
- Ignore suits (hearts, diamonds, spades, clubs)

Encoding	Card
0	No card
1	Ace
2 to 10	Cards 2 to 10
11	Jack
12	Queen
13	King
14, 15	Not Used

Displaying Cards

- Dealer and player can each hold a hand of up to four cards.
- Use 16bits (4cards x 4 bits/card) to represent a hand
- Display dealer's and player's hands to 7-segment display:
  - HEX3 – HEX0 for player
  - HEX7 – HEX4 for dealer

Encoding	Card	7-Segment Display
0	No card	BLANK
1	Ace	A
2 to 10	Cards 2 to 10	2 to 9, use 0 to represent 10
11	Jack	J
12	Queen	q
13	King	H

card7seg.vhd

```
COMPONENT Card7Seg IS
PORT(
    card : IN  STD_LOGIC_VECTOR(3 DOWNTO 0); -- value of card
    seg7 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)  -- 7-seg LED pattern
);
END COMPONENT;
```

Dealer A.I.

We use a very simple A.I. for the dealer:

- Makes decisions independent of player's hand
- If current dealer hand is < 16, always HIT
- If current dealer hand >= 17, always STAND

Datapath - scorehand.vhd

Need Two Instances of scorehand component

- One to score player's hand
- One to score dealer's hand

Input:

- 16-bit hand

Output:

- 5-bit: score for the hand
- 1-bit: "stand" tells FSM that dealer should stand (only applicable in the instance for the dealer)
- 1-bit: "bust" tells FSM that hand is a bust

THIS PART IS HARD! Read Step 7 carefully.

Test this component separately. Use testbenches and simulation!

Datapath - dealcard.vhd

How do we do random card generation?

Assumptions and simplifications:

- Dealing from infinite deck → equally likely go generate any card
- Don't care about what cards have already been drawn
- Random amount of time passes between presses of *nextStep* switch
  - Use this observation as randomizer

Solution:

- Continuously count through cards at a very high rate (50MHz clock)
  - A, 2, 3, …, J, Q, K, A, 2, 3, …
- When we need a card, sample the current count

FSM.vhd

We’re not designing the FSM for you this time, but we have provided suggestions on how to go about it.

Look at the steps needed for gameplay. That should give you a good idea on how many states you need, and what should happen in each state.

Hint: My solution has about 10 states

Game Progression

The **nextStep** signal advances the state of the game. Up until now, the FSMs that we talked about advanced state on each clock cycle.

In a real system, each state typically requires **multiple** clock cycles to perform the computations required by the state.

One way to do this, is to describe a **second** simple FSM to generate an **ENABLE** signal from the **nextStep** signal that controls your main FSM.

The VHDL for this part is quite simple, but it does require a bit of thinking. Try to solve this part early so that you can ask for help from your TAs if necessary.

Sample Design and Testing Process

We have included a detailed step-by-step guide on how to finish this assignment.

If there is even a remote chance that you think you will not be able to complete the assignment, you should follow these steps exactly because it will help you get part marks.

Otherwise, it's just a suggestion.

Marking

- No Challenge Task for this Lab

- If you complete the entire assignment and can demo the correct gameplay behaviour, you will receive full marks. 10/10

- If you cannot complete the entire assignment, you can demo specific portions for part marks. If you do this, it is up to you to organize how you will demo the various individual components to the TA.
  - The steps in the “Sample Design and Testing Process” have marks associated with them.

A few points:

1. We have helped you specify how to break down the design into components. In future labs, we will leave the partitioning of components up to you.

2. Use INTEGER, SIGNED or UNSIGNED type when you need to do arithmetic (for example in scorehand.vhd, dealcard.vhd)

3. Don’t leave all your testing until the end. Test each component separately. Testbenches are highly encouraged for scorehand.vhd

4. Synchronous or Asynchronous Reset? Read the requirements description.