UF Herbert Wertheim College of Engineering
Department of Electrical & Computer Engineering
UNIVERSITY of FLORIDA

Dr. Christophe Bobda
Vignesh Saravanan
EEL3701C - Fall 2025

Lab 5: State Machines

Overview

- Use an ASM diagram to model a complex state machine
- Use VHDL as an aid for combinational logic design
- Understand the difference between Moore (synchronous) and Mealy (asynchronous) outputs in a state machine.

Introduction

So far, our sequential logic design has been limited to simple counters (which we discussed in lab 3). Most state machines in the real world, however, are more complex than simple counters. They have additional outputs that are functions of the current state and user-provided inputs. Some of these outputs **(Moore Outputs)** are determined simply by the current state, and other outputs **(Mealy Outputs)** are determined both by the current state and user-provided inputs.

We will design one such state machine in this lab using both VHDL and BDF files. VHDL will be used in order to create a combinational logic block that will go on the top-level entity (the BDF file), which will contain the system inputs, system outputs, and flip-flops necessary for the state machine. While VHDL can actually be used for both combinational and sequential logic design, **sequential logic** in this course will be done **entirely** using flip-flop components in BDFs (no process blocks allowed).

Pre-Lab Procedure

In this lab, you will design a state machine to model the behavior of a simple airport bag scanner that loads a bag, detects the presence of suspicious items, redirects the bag if necessary, and then unloads the bag accordingly.

Bag Scanner Inputs:

- bag_present (H)
- suspicious_item (H)
- **Asynchronous:** Reset (L): should set the state machine to the idle state

Bag Scanner Outputs

- load_next_bag (H)
- xray_enable (H)

- redirect_bag (H)
- unload_bag (H)

Detailed Specifications

1. The bag scanner should begin in an idle state, in which none of the outputs are true. If there is already a **bag present,** the bag scanner should remain in the idle state. If there is no bag present, go to step #2.
2. The bag scanner will now assert **load_next_bag** for three consecutive clock cycles. After the third clock cycle, move on to step #3.
3. Assert **xray_enable** for two consecutive clock cycles. During each clock cycle, check for the presence of any **suspicious item** in the loaded bag. If there is any **suspicious item**, **redirect** the bag immediately and proceed to step 4. Otherwise, proceed to step 4 after the second clock cycle without redirecting the bag.
4. Assert **unload_bag** for three consecutive clock cycles. During the third clock cycle, check whether the **bag is still present.** If it is, remain in the same state. If not, return to the idle state once again.

A few notes:

- If an **input** is not mentioned in the specifications for a given state, then that means that it has no impact on that state (or set of states). If an **output** is not mentioned in the specifications for a given state, it is assumed to be **false** for the entirety of that state.
- Remember that in an ASM diagram, **inputs** that don't matter in a certain state are not included. Similarly, **outputs** that are false in a given state are not included.
- It is possible that you will use, for instance, **four** flip flops in order to represent **less** than 2 * 2 * 2 * 2 = **16** states. In this scenario, keep in mind that some configurations of state bits will be unused by the controller. In your next-state truth table, the next-state bits and the outputs of these unused states should all be represented as **don't cares** (X).

Instructions:

1. Create an ASM chart for the bag scanner described above. This can be done on paper or digitally (using draw.io, for instance) as long as your states and logic are clearly visible. **Include this ASM as a labeled figure in the pre-lab report.** If you would like for a PI to verify that your ASM is correct, **come to office hours.** Do **not** share components of your lab (NSTT, ASM, VHDL code, etc.) on the public help channel in Slack.
2. Create a next-state truth table showing the current state, system inputs, next state, flip-flop inputs (**make sure to use only D flip-flops in your design**), and system outputs. **Include this NSTT as a labeled table in the pre-lab report.**
3. Write down next-state equations for each state bit and the system output equations. **Include these equations as well in the pre-lab report.** There is **no need** to simplify your equations using K-maps in this lab.

4. Create a Quartus project named Lab5_StateMachine and make a BDF for your design. Set it as the top-level entity (Ctrl + Shift + j accomplishes this on Windows).
5. Design the state machine on the BDF using a combinational logic block (created using VHDL) and D-flip-flops. Note that the state bits will be present as outputs in the overall design but will be inputs within the combinational logic block. Similarly, the next-state signals will be outputs within the combinational logic block but intermediate signals within the overall design. **Include screenshots of the BDF and the VHDL code in the pre-lab report.**
6. Perform a functional compilation of your design, simulate it in a waveform.vwf file, and annotate a screenshot of the simulation using Microsoft Paint or a similar tool. The system inputs, system outputs, clock signal, active-low reset, and state bits should all be present in the simulation. When annotating the simulation, make sure to identify the following:
    a. Rising edges of clock signal
    b. Impact of asynchronous, active-low reset signal
    c. State or group of states (Idle, Loading, Xray, Unloading)
    d. Impact of relevant input(s) in each state
    e. Demonstrate that Mealy outputs (signals in the ovals in your ASM diagram) change **immediately.** They should **not** wait for the next rising-edge clock signal in order to change.

    Include your annotated simulation in the pre-lab report.
7. Now, go to Assignments → Pin Planner on Quartus and assign the system output signals to the active-high LEDs on the DE10-Lite. Assign the system input signals and the asynchronous active-low reset signal to slide switches on the DE10-Lite. Your clock signal should be on either a button or a switch depending on what works best for your DE10-Lite. **Include a table in the pre-lab report that identifies the pin on the DE10-Lite corresponding to each signal in the state machine.**
8. Upload your design onto the DE10-Lite and test it, verifying that it matches your ASM and annotated simulation.
9. When you are done with the lab, archive the Quartus project and submit the qar file, along with the pre-lab report, on GitHub Classroom. The qar file submitted on Github Classroom is your final submission, and it cannot be changed after the deadline. This file must contain a working design that is ready to demo. **“Ready to demo” implies that you have already run your design on your DE10-Lite and have assigned all the pins. You will be given no extra time in-lab to assign pins or otherwise change the project file.**

In-Lab Procedure

1. Demonstrate the correct functioning of the bag scanner designed in the pre-lab.
2. Complete the lab quiz.

UF Herbert Wertheim College of Engineering
Department of Electrical & Computer Engineering
UNIVERSITY of FLORIDA

Dr. Christophe Bobda
Vignesh Saravanan
EEL3701C - Fall 2025

Rubric

Note	Points
Pre-Lab	
Correct ASM Chart	4
All rows of NSTT are correct	4
Correct next-state equations	4
Correct annotated simulation that shows all features outlined in the procedure	8
Lab report shows work and/or screenshots for all pre-lab parts.	20
In-Lab	
Lab 5 demo	20
Quiz	40
Total	**100**

Resources for ASM Diagrams

Keep in mind that the ASM diagram required in this pre-lab and in your lab quiz is **not** the same thing as a flowchart. If you are unsure how exactly to make an ASM, it is recommended that you watch the following videos from the YouTube channel *Neso Academy*, which is an excellent resource for many topics within digital electronics.

1. ASM Chart
2. ASM Chart for Moore State Machine

Supplemental VHDL Tutorial

If you would like to review basic VHDL syntax (required in this lab), the following tutorial covers the entire process of creating a combinational logic block in VHDL.

Note: You need to sign in with your UF email in order to access the tutorial.
Supplemental VHDL Resource for Lab 5