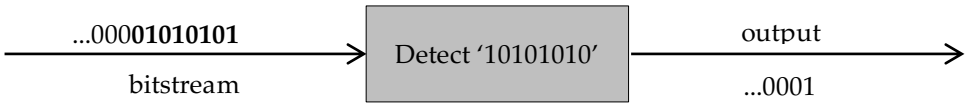
Lab 11: Detecting a Simple Sequence
CS/CoE 0447 Spring 2017

Each person must turn in their own copy of the lab. If you choose to work with a neighbor/partner, put your partner's name on your submitted copy of the lab. Submission timestamps will be checked. Late submissions will not be accepted. Follow instructions.

For this lab, we will use Logisim, which is available at http://www.cburch.com/logisim/.

A **bitstream** is an unstructured sequence of bits commonly used for transmitting or storing digital data. In some applications or devices, it is useful to detect a particular sequence of bits in the stream, for example to recognize the start of a data packet or file (e.g., a particular pattern of bits represents the start of an important piece of data). The following figure shows an example:



In the example above, a 1 is first sent to the bitstream analyzer (the bitstream "flows" from right to left into the analyzer). In the next clock cycle, a 0 is sent, and so on. Once the bitstream analyzer has detected that it has seen 10101010, it outputs a 1.[1] An application or device might need to know when the sequence 10101010 has been seen in the input.

One possible way to detect such an input pattern is to use a large *n*-bit register, where *n* is the size of the bit sequence that must be detected. This sequence might be very large, requiring a large amount of memory. For example, in Ethernet frames, the preamble is eight bytes (256 bits) long.

Another solution, one which you will use in this lab, is to use an advanced sequential logic called **finite state machine (FSM).** The finite state machine, depending on the sequence being detected, may require less memory than simply storing all the previously seen bits. FSMs also allow for very powerful pattern matching beyond just detecting whether a sequence of bits matches a particular pattern.[2]

A FSM consists of a few components, which you will first (a) design and then (b) implement in Logisim. Your design will preferably be done electronically (e.g., in PowerPoint or, Word), though you may draw it with pen-and-paper and submit a scanned copy. Your circuit must be submitted as a Logisim file.

---

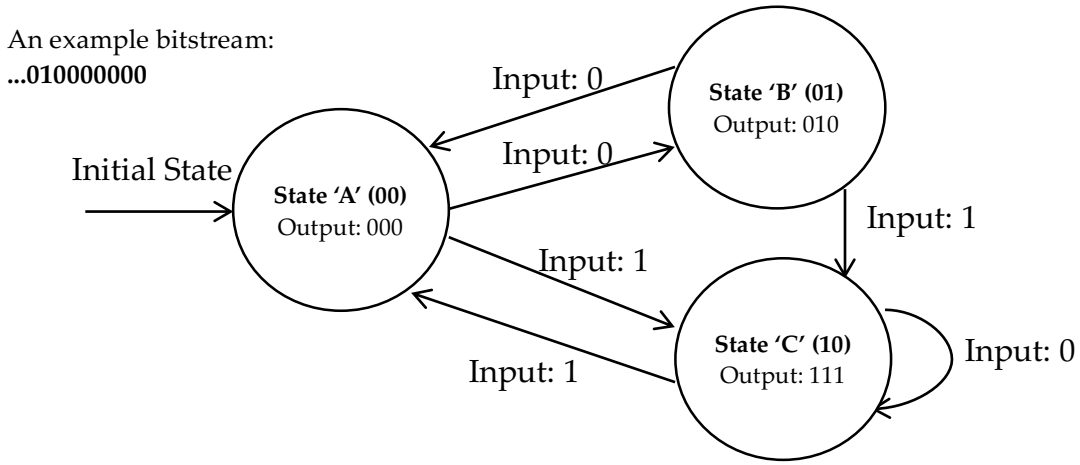
[1] "10101010" is actually an 8-bit pattern used in Ethernet packet preambles to signal the start of an Ethernet data packet.

[2] For example, all regular expressions can be implemented as FSMs. One such example regular expression is: `^4[0-9]{12}(?:[0-9]{3})?$`. That regular expression checks whether a string of numbers is a valid Visa credit card number.

Example FSM Diagram and Design

FSM design first requires the drawing of a finite state machine diagram. The diagram is a visual representation of the entire FSM: its initial state, how it changes states depending on the input, and its output at each state. As humans, we might prefer to look at the diagram, because it is a visual representation. Building a FSM in Logisim will require a more formal representation which will soon be discussed.

Here is an example FSM diagram:



The finite state machine diagram completely describes how the FSM works. When an input arrives, the arrows show what new state the FSM will be in. A FSM can only be in one state at any time. Depending on the state of the FSM, the FSM will output different values. The way that the FSM moves between states and outputs its values is dependent on the purpose of the FSM: No two FSMs will be alike.

The example FSM shown above outputs 111 when the FSM has been given as input an odd number of 0s, followed by a single 1 (optionally followed by an infinite number of 0s). If the FSM has been given as input a run of an odd number of 0s, it will output 010. The FSM also outputs 111 for other bit sequences (e.g. 1, 111).

Because the example FSM has three states in total, we can use two bits to describe which state the FSM is in. Herein, state "A" will be state 00, "B" will be 01, and "C" will be 10. Other encodings can exist (e.g., we could refer to state "C" as 11 but in this example we shall not).

The two bits of state could be, for example, implemented using two D flip-flops. The diagram shows that on each clock cycle the input to the FSM is a single bit. In this lab, your FSMs will only process a single bit of input at a time. Each FSM state has three bits of output. The number of bits of output is not related to how many bits of state there are, or how large the input is. Instead, remember that the number of bits of output depends on what you need the FSM to do.

Now that each state has a number assigned to it (e.g., "A" is 00), **we can use truth tables** to describe (a) how the FSM moves between states given an input (the next state function), and (b) what a particular state's output should be (the output function). Presented next are the example FSM's next-state and output functions. The outputs (generated values) are in bold.

Example FSM Next State Function
Maps a two-bit state and a one-bit input to a new two-bit state.

State_1	State_0	Input	NewState_1	NewState_0
0	0	0	**0**	**1**
0	0	1	**1**	**0**
0	1	0	**0**	**0**
0	1	1	**1**	**0**
1	0	0	**1**	**0**
1	0	1	**0**	**0**

Example FSM Output Function
Maps a two-bit state to a three-bit output value.

State_1	State_0	Output_2	Output_1	Output_0
0	0	**0**	**0**	**0**
0	1	**0**	**1**	**0**
1	0	**1**	**1**	**1**

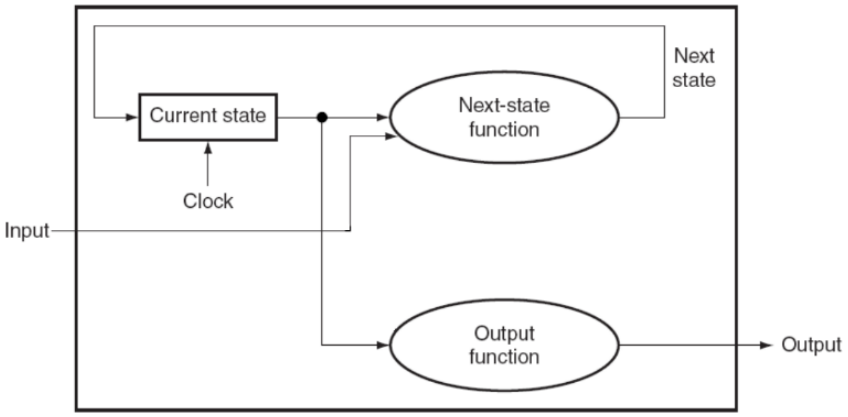
Detecting a simple sequence

You will go through several steps to construct a small circuit that **detects when a 0 arrives immediately after two adjacent ones arrived** in a bitstream (e.g., …00010**011**). The circuit outputs a '0' unless it detects the sequence. If the circuit detects the sequence, it outputs a '1' until the next bit arrives.

a. **Draw** the state diagram for a finite state machine (FSM) that detects the sequence specified above. Your FSM will process one bit at a time. Remember to **encode** the states of the FSM by numbering each state, starting with zero. (For example, State "A" is 00.) Include this encoding on your diagram, as shown in the example. **After the sequence is detected, on the next input, you should send your state machine back to an appropriate state such that it could detect the sequence again if it reappears.** Submit this drawing electronically as a PowerPoint or Word document, or an image file (e.g., a digital scan of a pen-and-paper drawing).

b. **Write** the truth tables for the next state and output functions. Each of your truth tables should have one row for every combination of possible input values and state. Recall that the output of the output function is a single bit (i.e., whether the pattern was seen or not), and that it depends only on the current state of the FSM. Submit these tables electronically in a single Word, Excel, plain-text, or an image file (e.g., a scanned document).

c. **Implement** the circuit in Logisim. You may use combinational analysis to simplify your circuit. Be sure that you use a clock, a button to let you test different inputs, and a register of appropriate width to store the state. Name this circuit **lab11partC-<your Pitt Mail ID>.circ".** For example, "**lab11partC--xyz11.circ**". The diagram at right shows a high-level example of a state machine.



Tip: Under the Simulate menu item, you can specify a slow clock tick frequency (e.g., 0.5 Hz) and enable ticks. This will allow you do easily test your circuit. As the clock ticks, poke your circuit's button to make sure that it correctly detects the proper bit pattern.