Sequential System Design Using ASM Charts

Introduction

Control unit design complexity may range from simple to highly complex. There are number of methods to design and realize control units. Simple control units can be designed using state graphs and state table methods. Complex control units may be designed using algorithmic charts just like flowcharts are used in software development. In this lab, you are introduced to Algorithmic State Machine (ASM) chart technique. *Please refer to the PlanAhead tutorial on how to use the PlanAhead tool for creating projects and verifying digital circuits*.

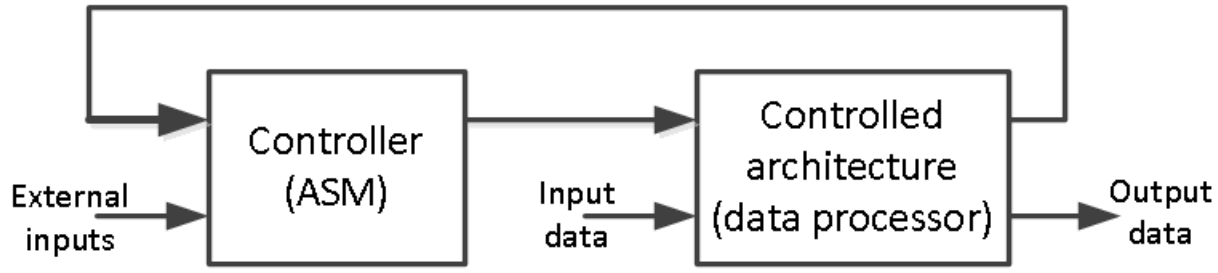
Objectives

After completing this lab, you will be able to:

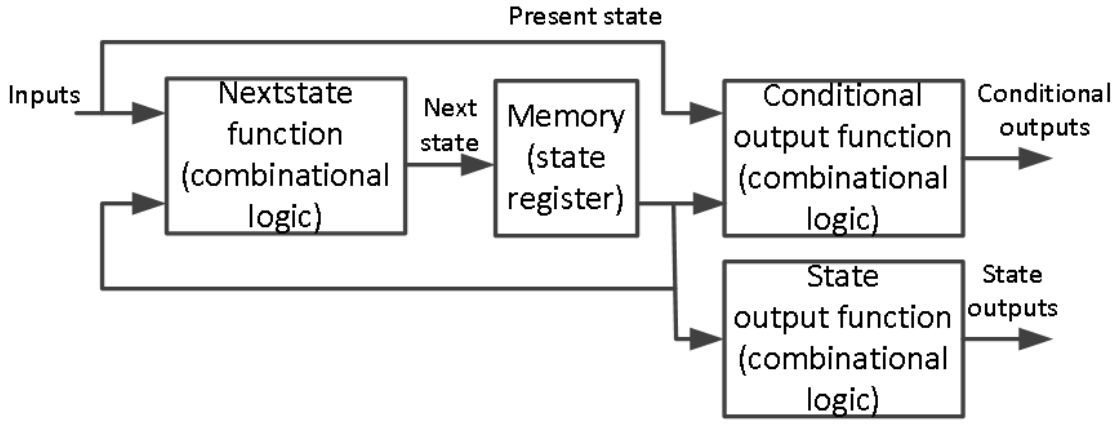
- Use the ASM charts to design sequential systems

ASM Charts Part 1

Just as flowcharts are useful in software design, special kind of flowcharts, called Algorithmic State Machines (ASM) charts, are useful in the hardware design of digital systems. Digital systems typically consist of datapath processing and control path. The control path is implemented using state machines which can be realized using state graphs. As the control path (behavior) of the system becomes complex, it becomes increasingly difficult to design the control path using the state graph technique. The ASM charts technique becomes useful and handy in designing complex and algorithmic circuits. The following diagram shows a complex digital system, partitioned into a controller (to generate the control signals) and the controlled architecture (data processor).

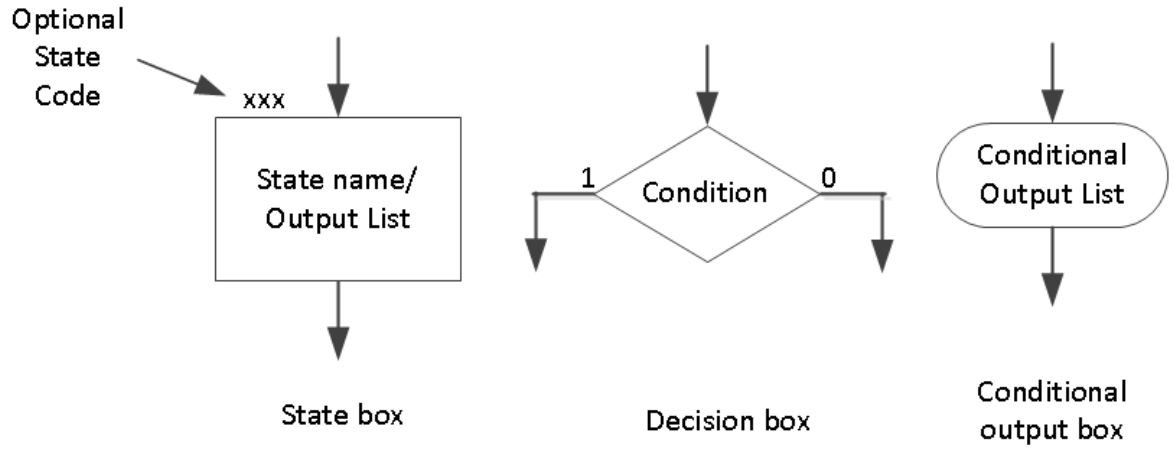


The model of the ASM can be viewed as the combination of Mealy and Moore machines.



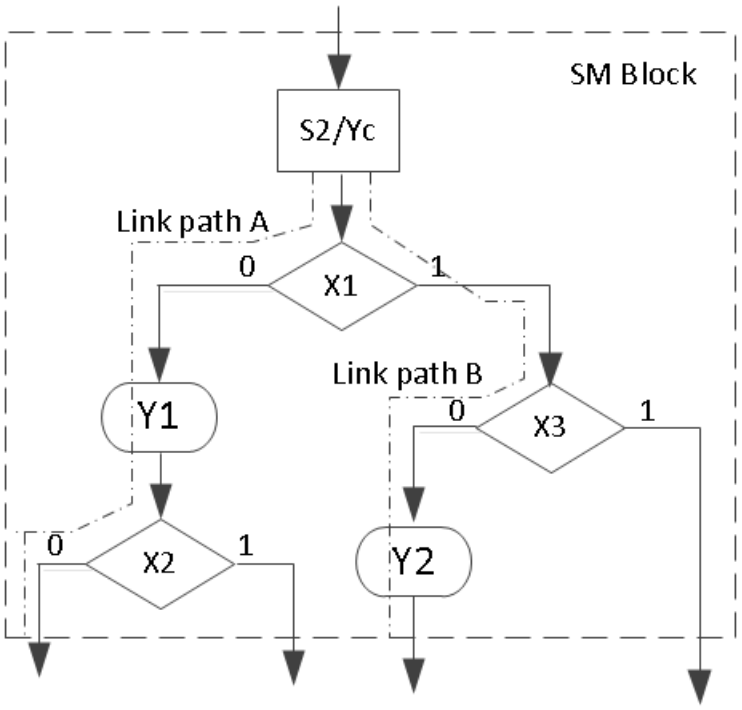
The ASM chart differs from an ordinary flowchart in that certain specific rules must be followed in construing the chart. When these rules are followed, the ASM chart is equivalent to a state graph, and it

leads directly to a hardware realization. The following diagram shows the three main components of an ASM chart.



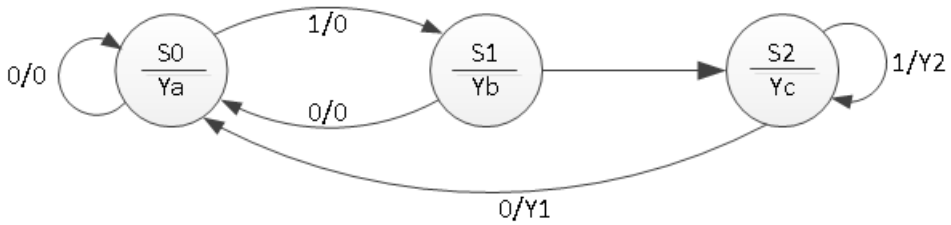
The state of the system is represented by a *state box*. The *state box* contains a state name, and it may contain an *output list* (just like in state in a state graph of the Moore machine). A *state code* may be placed outside the box at the top (if you want to assign a state code). A *decision box* always has true and false branches. The condition placed in the *decision box* must be a Boolean expression that is evaluated to determine which branch to take. The conditional output box contains a *conditional output list*. The conditional outputs depend on both the state of the system and the inputs (just like in the Mealy machine).

ASM chart is constructed from SM blocks. Each SM block contains exactly one state box together with decision boxes and conditional output boxes associated with that state as shown below. An SM block has exactly one entrance path and one or more exit paths. Each SM block describes the machine operation during the time that the machine is in that state. A path through an SM block from entrance to exit is referred to as a link path.

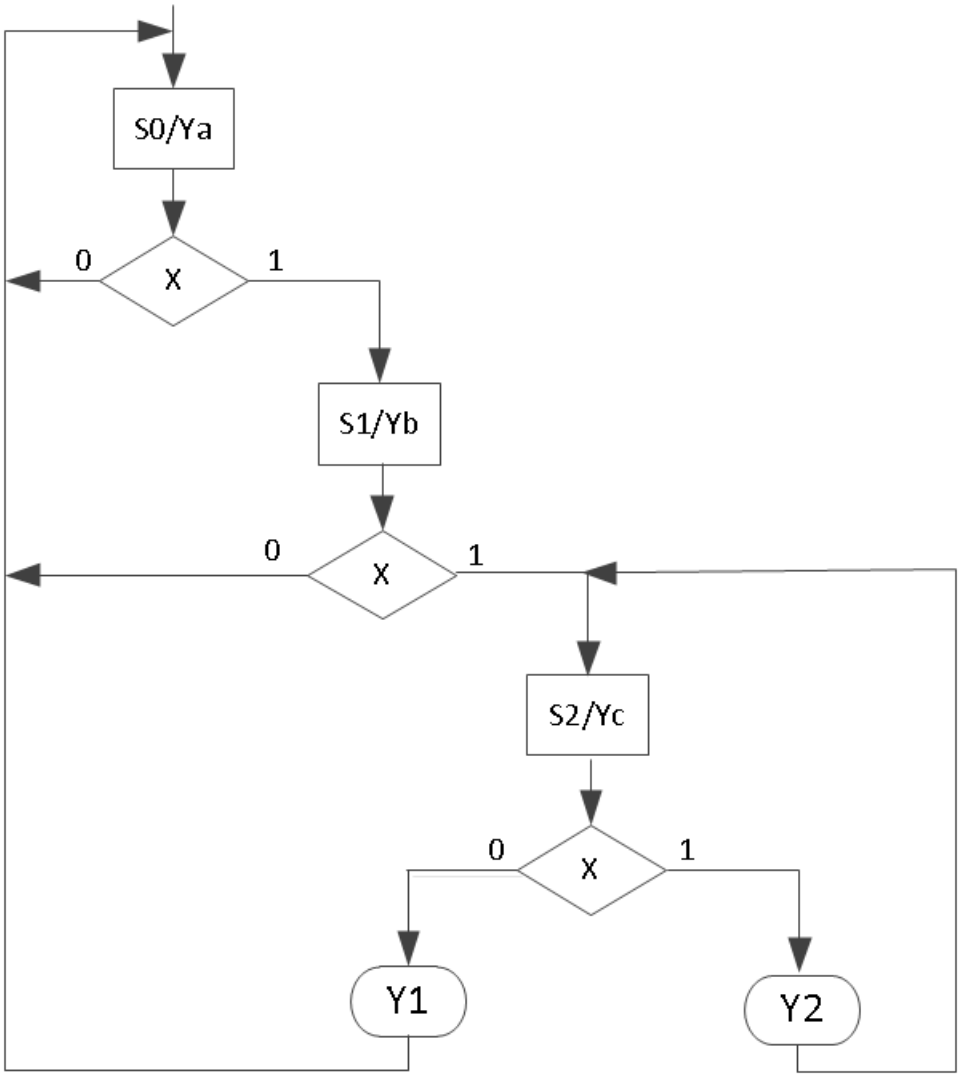


Let us consider an example of a state graph of a sequential network shown below. This state graph has both Mealy and Moore outputs. The output Y1 and Y2 are the Mealy outputs and so should be conditional

outputs. The Ya, Yb, and Yc are the Moore outputs so they should be part of state box. Input X can either be “0” or “1” and hence it should be part of the decision box.



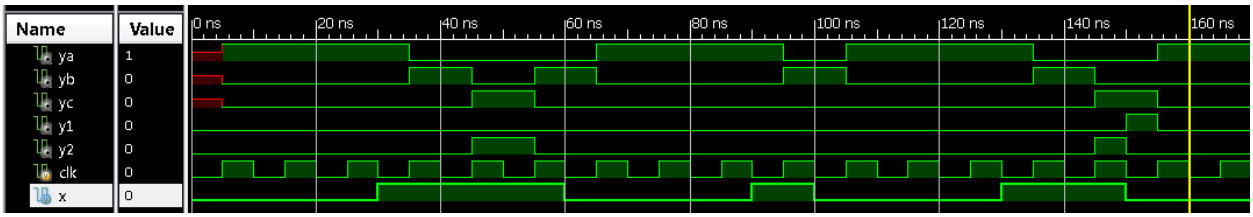
The ASM chart of the above state graph is as shown below.



Once the ASM chart is determined, the conversion to HDL is straight forward. A `case` statement can be used to specify what happens in each state. Each condition box corresponds directly to an *if* statement (or an `else if`). The following code represents the functionality of the above ASM chart.

```
type state_type is (S0, S1, S2);
signal state, next_state : state_type;

SYNC_PROC : process (clk)
begin
      if rising_edge(clk) then
            state <= next_state;
      end if;
end process;

NEXT_STATE_DECODE : process (state, x)
begin
      y1 <= '0';
      y2 <= '0';

      case (state) is
            when S0 =>
                  if (x = '1') then
                        next_state <= S1;
                  else
                        next_state <= S0;
                  end if;
            when S1 =>
                  if (x = '1') then
                        next_state <= S2;
                  else
                        next_state <= S0;
                  end if;
            when S2 =>
                  if (x = '1') then
                        y2 <= '1';
                        next_state <= S1;
                  else
                        y1 <= '1';
                        next_state <= S0;
                  end if;
            when others =>
                  next_state <= S0;
            end case;
end process;

OUTPUT_DECODE : process (state)
begin
      ya <= '0';
      yb <= '0';
      yc <= '0';

      case (state) is
            when S0 => ya <= '1';
            when S1 => yb <= '1';
            when S2 => yc <= '1';
            when others =>
                  ya <= '0';
                  yb <= '0';
                  yc <= '0';
      end case;
end process;
```

The following behavioral simulation result shows the above model functionality.



1-1. Design a 3-Bit x 3-Bit binary multiplier. The multiplier will output 6-Bit product. The data processor unit will consist of a 3-Bit accumulator, a 3-Bit multiplier register, a 3-Bit adder, a counter, and a 3-bit shifter. The control unit will consist of a least-significant-bit (*lsb*) of the multiplier, a *start* signal, a *cnt_done* signal, and *clk* as an input. It will generate *start*, *shift*, a*dd*, and *done* signals. Develop an ASM chart for the control unit. Develop models for the data processor and the control unit. Develop a testbench to validate the design using the behavioral simulation.

1-2. Implement the design of 1-1 using SW7:SW5 as a *multiplier* input, SW4:SW2 as a *multiplicand* input, SW0 as a *clk* input, BTNU as a *start* input. Use LED7:LED2 as the *product* output, and LED0 as a *done* signal output. Go through the design flow, generate the bitstream, and download it into the Nexys3 board. Verify the functionality. Hint: You will have keep the BTNU pushed while switching (generating clock pulse) SW0 for the first time.

Sequential System Design Using ASM Chart Part 2

We saw how the ASM chart technique can be used in designing control units of sequential machine. Now we will use the ASM chart technique along with the sequential design principles to design complex systems.

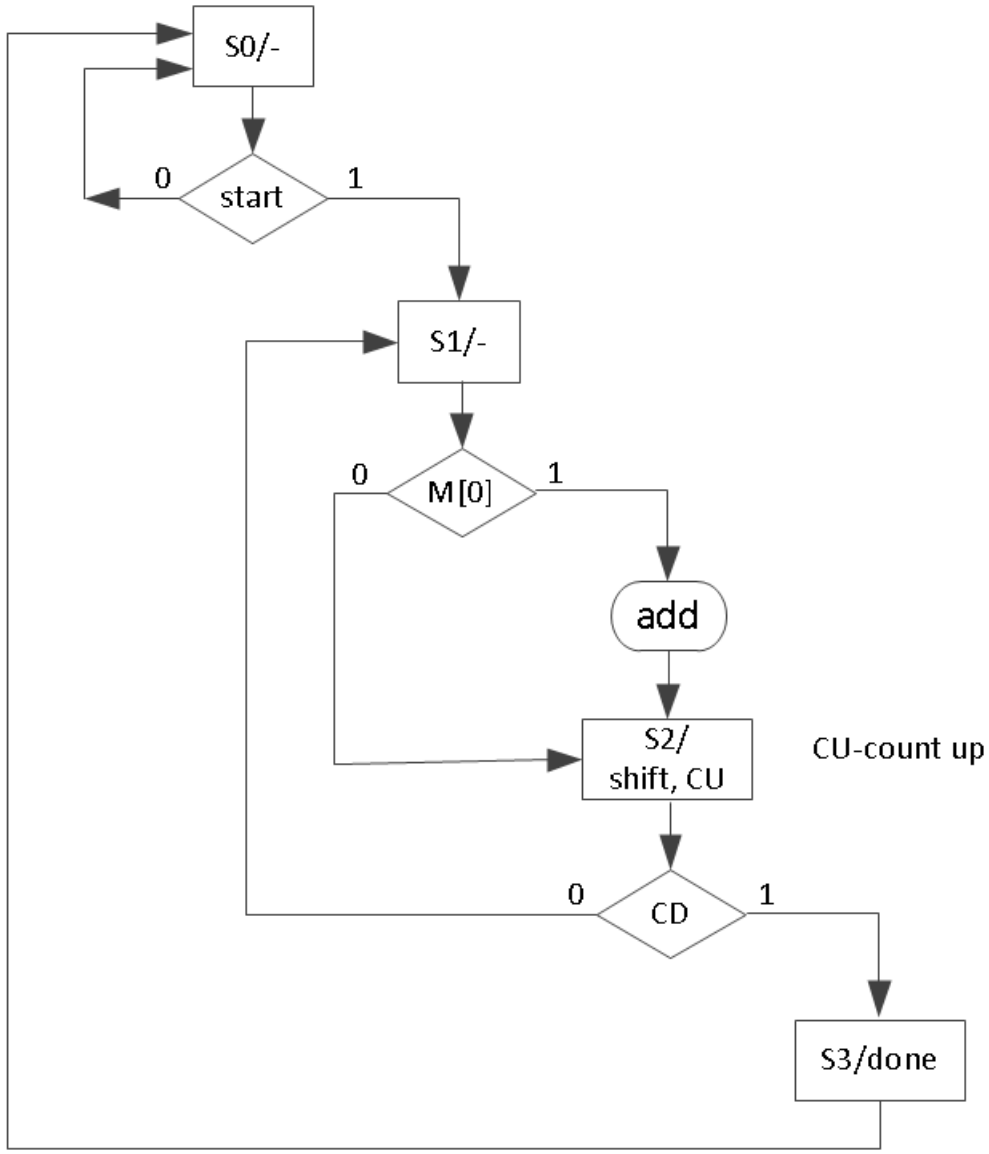
2-1. Modify the design of 1-2 to perform 4-Bit x 4-Bit unsigned multiplication. You will store the 4-Bit multiplicand and multipliers in 32x4 ROM (first 16 locations holding multiplicands and the other 16 locations holding multipliers). You will input multiplicand and multiplier operand addresses using switches. Use the clocking wizard to generate 5 MHz clock. Use the on-board clock source of 100 MHz, the BTNU button to start the calculation, SW7:SW4 to input the multiplicand address, SW3:SW0 to input the multiplier address, LED0 to output the done signal, and right most three 7-segment displays to show the result. Go through the design flow, generate the bitstream, and download it into the Nexys3 board. Verify the functionality.

Conclusion

In this lab, you learned how ASM charts can be used to design complex control units. You also designed digital system to perform binary multiplication using the ASM chart technique to develop the control unit which interfaced to the datapath processing unit..

Solution

ASM chart for 1-2



Block Diagram of 2-1