LAB: Complex Linked State Machines in VHDL

# Bill of Materials (optional task only):

DE-0 Nano FPGA Board

Female-to-female ribbon connector (JP1)

STM32F429FZ Nucleo Board

Prototyping Wires (5 off)

# Software Required

Quartus Prime v16.1

Keil uVision 5.2x (Advanced task only)

PuTTY (Advanced task only)

# Document Revision

V1.0 – 08/02/2018 – Initial version for ELEC241

V1.01 – 08/02/2018 – Clarified task in 01-01 to address misunderstandings in the lab

# Task 01-01 Build and Test a Timer

This task is **related** to the example given in the lecture, but not exactly the same.

Problem Statement

A user presses and holds down a button x

* If the user holds down the button for 256 clock cycles (0.25s), the output Y is set to 1
* If the user releases the button before 256 clock cycles are done, the process has to start again

This task is broken down into two state machines, which are linked together (see Figure 3). The first is a reusable timer component. The second is a state machine to receive input from a switch.

In this section, you should address the first task, which is to build the timer sub-component in VHDL. The ASM chart for the timer is shown in Figure 1.

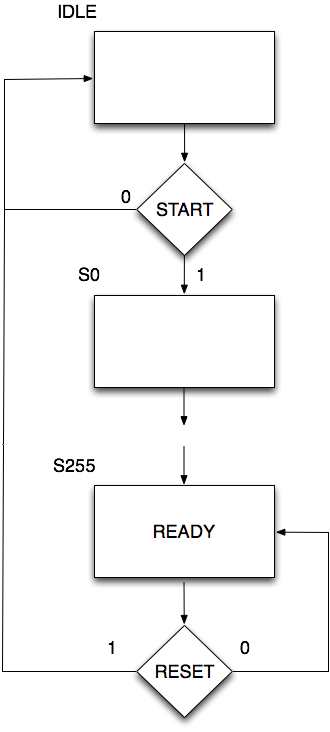


Figure 1. ASM Chart for the Timer

To get you started, a template project has been provided (LinkedFSM.zip). The VHDL entity and architecture have been provided. The entity is also shown below.

**entity** counter\_233 **is**

**port**(

clk : **in** std\_logic;

start : **in** std\_logic;

reset : **in** std\_logic;

ready : **out** std\_logic

);

**end entity**;

However, the architecture is not yet complete. Below is the beginning of the architecture that is yet to fully conform to the design in Figure 1.

## TASKS

1. Complete the Timer architecture below. Note the comments indicating where you need to write more code. Follow the ASM chart carefully.
2. Create a symbol for this component and place it on the top-level schematic. Add some inputs and outputs for test purposes.
3. Build the code.
4. Create vector waveform file to test this device. Think carefully about what test inputs will verify the design in the ASM chart (assessed).
5. Show your solution and test results in your log book.

**Tips:**

Use if-elsif-else instead of a case statement.

States S0 to S254 are identical. They only need a single if statement.

**Question**: If we modify the number of states to count from S0 to S254, how many gates / registers do we save? Comment on the reason for the difference.

**architecture** rtl **of** counter\_233 **is**

**signal** state : integer **range** 0 **to** 256 := 0;

**begin**

-- Logic to advance to the next state

**process** (clk, reset)

**begin**

**if** (rising\_edge(clk)) **then**

**if** (state = 0) **then**

**if** (start = '1') **then**

state <= 1;

**end if**;

**-- \*\*\*\*\* TO BE DONE \*\*\*\***

**end if;**

**end if**;

**end process**;

-- Output depends solely on the current state

**process** (state)

**begin**

**-- \*\*\*\* TO BE DONE \*\*\*\***

**end process**;

**end** rtl;

# Task 01-02 Linking the Switch Controller to the Timer with the Schematic Editor

Now you need to build a second component. This is left hand ASM chart in Figure 2. The component on the right is the timer component from the previous task. The dashed lines represent the connections between the two components.

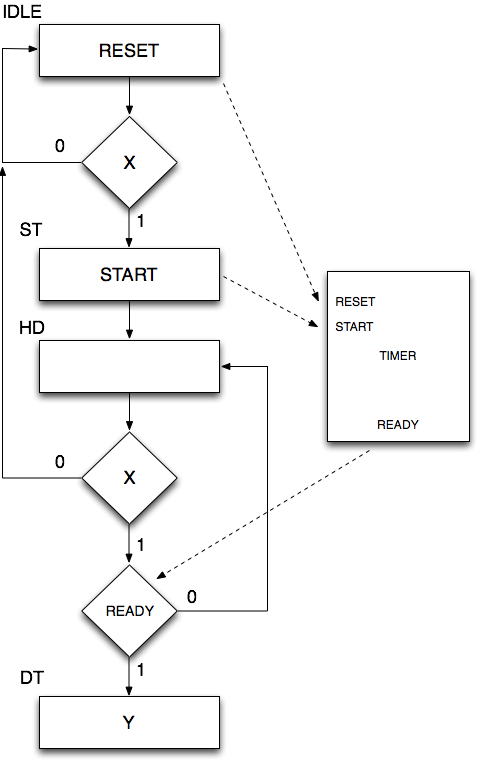


Figure 2. ASM Chart for a switch controller. This system has three outputs (RESET, START and Y) and two inputs (X and READY)

Once again, you are provided with the entity, and only part of the architecture.

**entity** control\_button **is**

**port**(

clk : **in** std\_logic;

X : **in** std\_logic; --Switch input

timer\_ready : **in** std\_logic; --Timer is ready input

--Outputs

reset\_timer : **out** std\_logic;

start\_timer : **out** std\_logic;

Y : **out** std\_logic

);

**end entity**;

For the architecture shown below, the more standard case-statement based state machine is used. However, it is not complete.

## TASKS

1. Complete the architecture below. Note the comments indicating where you need to write more code.
2. Create a symbol for this component and place it on the top-level schematic. Add some inputs and outputs for test purposes.
3. Connect to the timer component
4. Build the code.
5. Create vector waveform file to test this device. Think carefully about what test inputs will verify the design in the ASM chart (assessed) and the design brief.
6. Show your solution and test results in your log book.

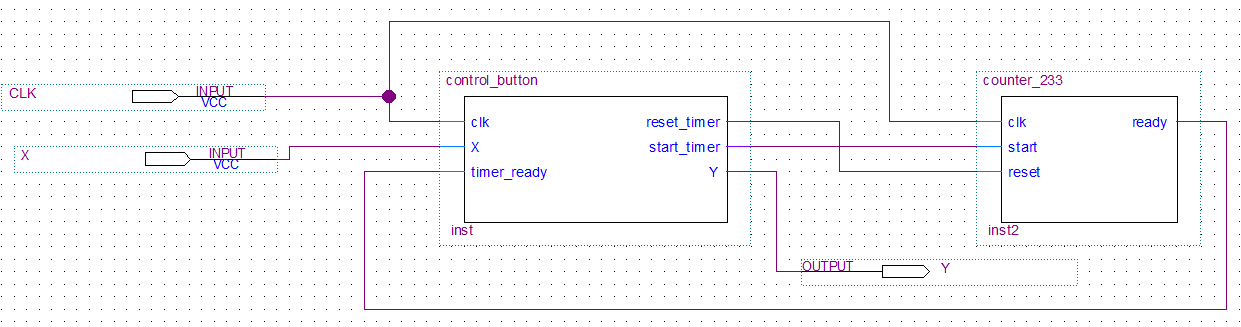


Figure . Showing the coupling of two state machines using the schematic editor

**Question**: Comment on why we have split the design into two state machines. What are the benefits?

**architecture** rtl **of** control\_button is

-- Build an enumerated type for the state machine

**type** state\_type **is** (IDLE, ST, HD, DT);

**signal** state : state\_type := IDLE;

**begin**

**process** (clk) -- Logic to advance to the next state

**begin**

**if** (rising\_edge(clk)) **then**

**case** state **is**

**when** IDLE=>

if X = '1' then

state <= ST;

else

state <= IDLE;

end if;

**when** ST=>

**-- \*\*\*\* To be done \*\*\*\***

**when** HD=>

**-- \*\*\*\* To be done \*\*\*\***

**when** DT =>

**-- \*\*\*\* To be done \*\*\*\***

**end case**;

**end if**;

**end process**;

**process** (state) -- Output depends solely on the current state

**begin**

**case** state **is**

**when** IDLE =>

Y <= '0';

**-- \*\*\*\* To be done \*\*\*\***

**when** ST =>

**-- \*\*\*\* To be done \*\*\*\***

**when** HD =>

**-- \*\*\*\* To be done \*\*\*\***

**when** DT =>

**-- \*\*\*\* To be done \*\*\*\***

**end case**;

**end process**;

**end** rtl;

# Task 01-03 Linking the Switch Controller to the Timer with Structural VHDL

In the previous task, two state machines were linked by simply wiring them together in a schematic. However, what if we didn’t use a schematic, or maybe wanted to create another VHDL entity that combined them together?

1. Refer to the previous module where we covered **structural VHDL**.
2. Now create a third entity in VHDL called button\_hold as shown below
3. Your task is to create the architecture and use structural VHDL to import and connect both the timer and the switch controller.
4. You should then create a symbol for this component and place it on the top level schematic.
5. You should now test this new component to show it produces the same output as in the previous section.

**entity** button\_hold **is**

**port**(

clk : **in** std\_logic;

X : **in** std\_logic; --Switch input

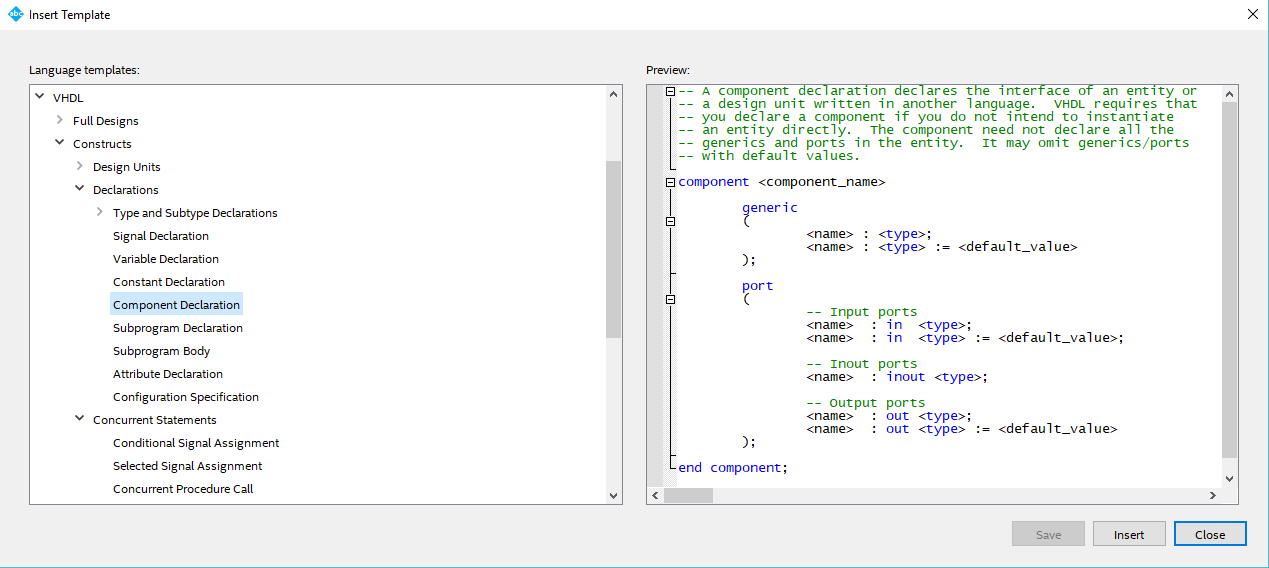
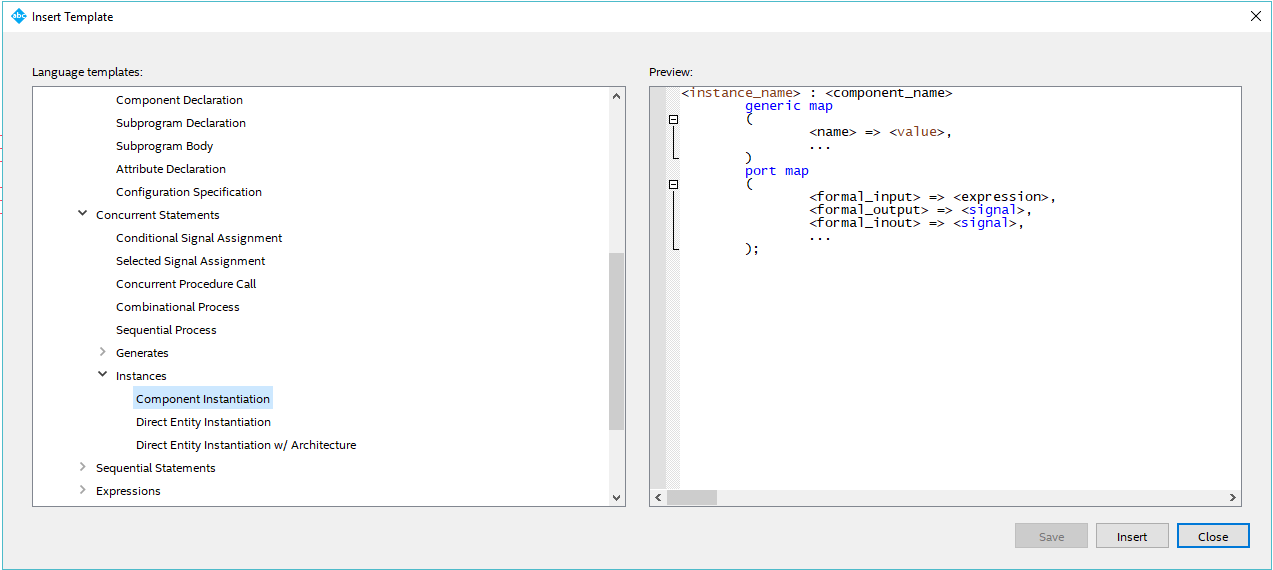
--Outputs

Y : **out** std\_logic

);

**end entity**;

**Really useful TIP**: Use the template generator in Quartus to help you! While editing your VHDL code, right click and select insert template. Choose VHDL->Constructs. Look for “Component Declaration” and “Component Instantiation” (You don’t need the generic sections).

# Task 02-01 Switch Debounce Component

You task is now to adapt what you did above to build a full switch debounce component. This component should have the following Entity:

**entity** switch\_debounce **is**

**port**(

clk : **in** std\_logic;

X : **in** std\_logic; --Switch input

--Outputs

Y : **out** std\_logic

);

**end entity**;

### Requirements

1. The output Y should **toggle** every time the input is goes HIGH then LOW.
2. A timer should be used to add a suitable clock delay to prevent switch bounce. Assume a clock frequency of 1024Hz.

### Tasks

* Draw a new ASM chart for this device. Use the existing timer as in previous tasks (you should not need to modify it)
* Create a VHDL entity switch\_debounce. Use the entity shown above. Remember that each entity should be in a VHDL file of the same name.
* Write the architecture. For full marks, use structural VHDL to link to the timer.
* Create a symbol file for the new component
* Test the component using a vector waveform file. Again, think carefully about the input sequences. Remember to include cases where switch bounce occurs (simulate this).
* Include your ASM chart, VHDL and test results in your log book

**Question**: Each time you test your design, consider how long it takes check the waveform output. Comment on how this approach impacts on your productivity, how it might scale as the design becomes more complex, and how reliable this approach is.

# Optional Task Connecting the FPGA to the Nucleo F429ZI Board

**This task is neither assessed or ready! Please check back for a status update.**

For fun, you could try connecting this device to the Nucleo Board. Set up an interrupt to output the USER\_BUTTON state to a GPIO pin. This pin should then be connected to the DE-0 FGPA board.

You will need to use the phase locked loop to generate a 1024Hz clock.

1. Connect the ribbon cable to the FPGA connector JP1 as shown in Figure 4.

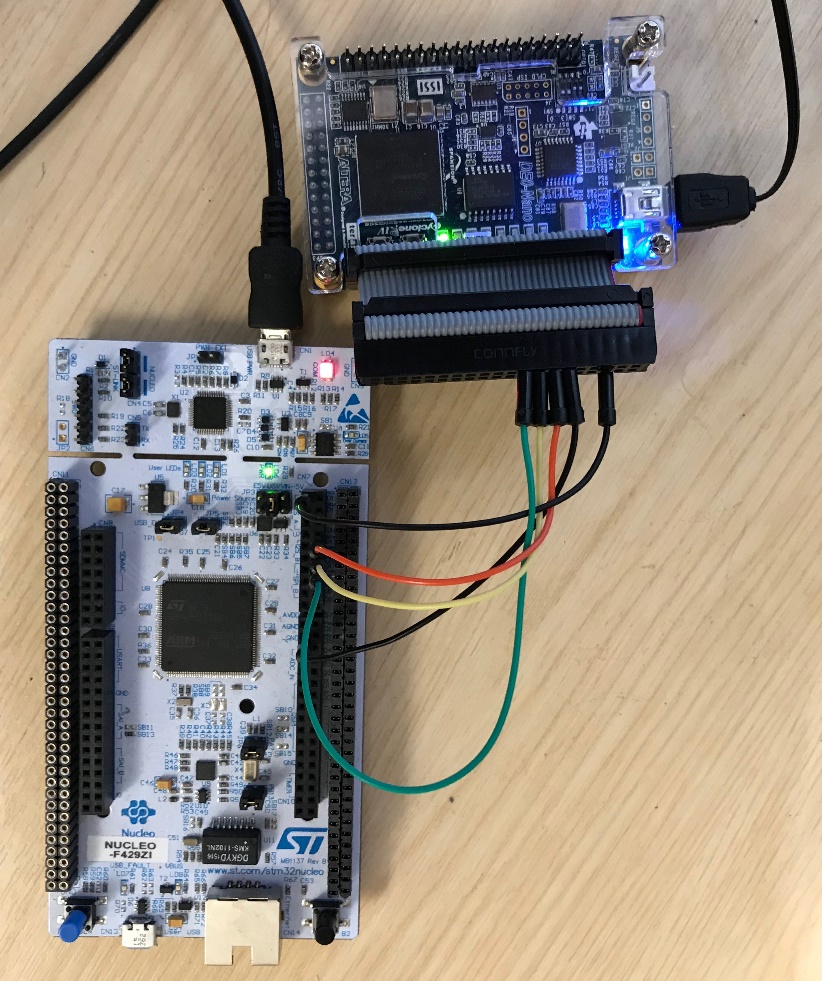


Figure 4. Showing the connections between the FPGA and the Nucleo Board

1. Using the connections in Table 1, connect the FPGA to the Nucleo Board using prototyping wires. Use Figure 5, Figure 6 and Figure 7 to locate the correct pins.

Table 1. Nucleo to DE0 Nano Connections for the SPI interface

|  |  |  |
| --- | --- | --- |
| MCU Signal | MCU Pin | FPGA Pin |
| SWITCH\_OUT | PC\_6 (D16) | A2 (JP1-Pin 5) GPIO\_2 |
| SWITCH\_IN | PA\_7 (D11) | D6 (JP1-Pin18) GPIO\_13 |

# Appendix A – Nucleo and DE0-Nano Connections

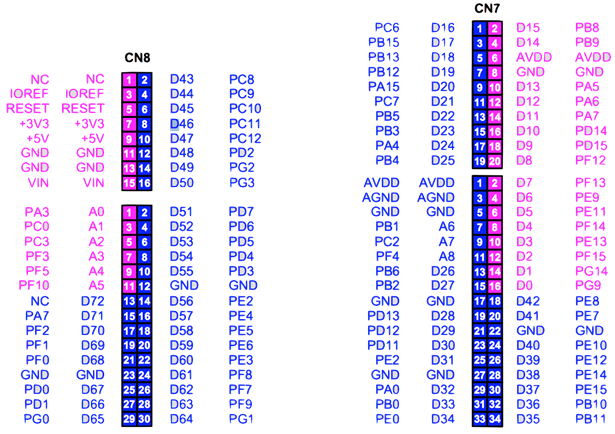


Figure 5 Nucleo Pinouts

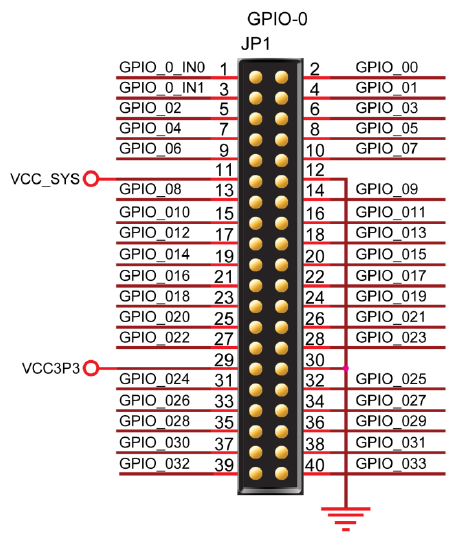


Figure 6. DE0-Nano Pinouts for JP1

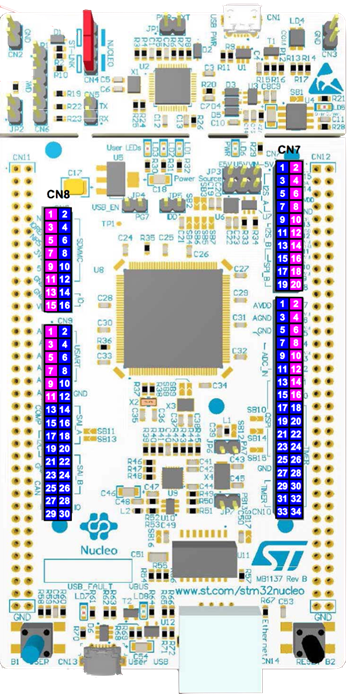


Figure 7. Nucleo FZ429ZI

A circuit board

Description generated with very high confidence

Pin 1

Figure 8. Showing the pin 1 on the FPGA DE-0 Nano Board

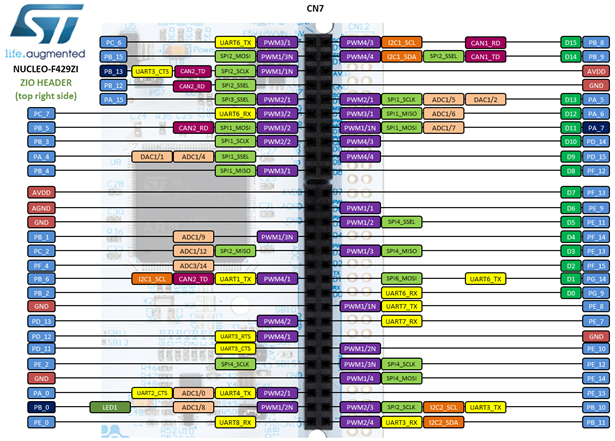
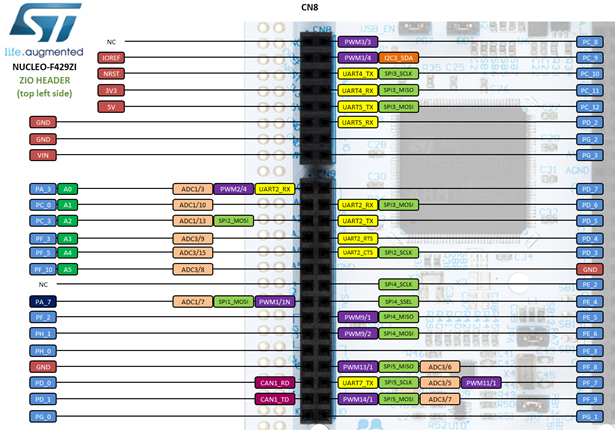


Figure 9. Showing the alternative functions of the pins on the Nucleo F429ZI CN7 connector



# Appendix B – ASM Charts

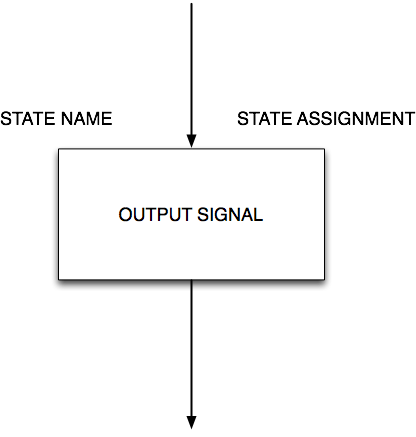


Figure 10 State with Moore Output Signal. The state name and assignment (optional) area all unique

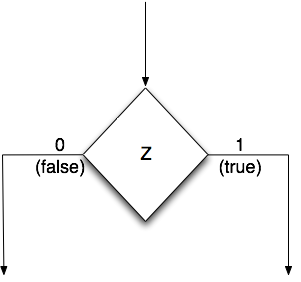


Figure 11 State Transition Decision. The variable Z is being tested. This must ultimately lead to a state box, possibly via another decision and/or Mealy Output(s)

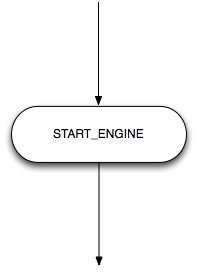


Figure 12. Mealy Output. The example here is the START\_ENGINE signal which is asserted high. It only makes sense to connect a decision symbol to the input of the Mealy output symbol.