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.

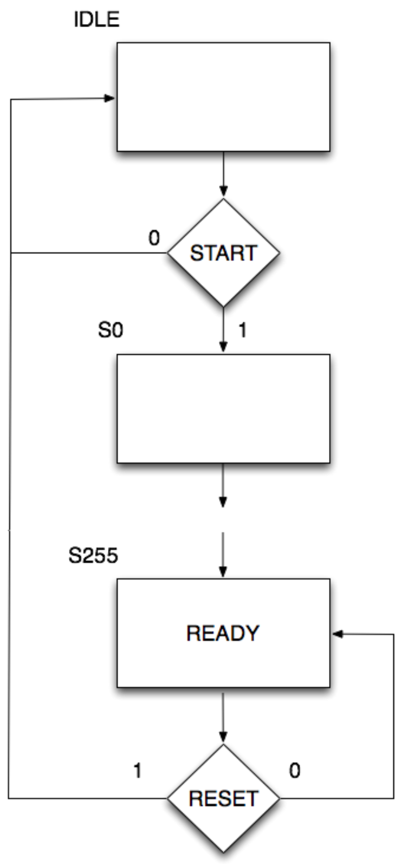


Figure 1. ASM Chart for the Timer

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.

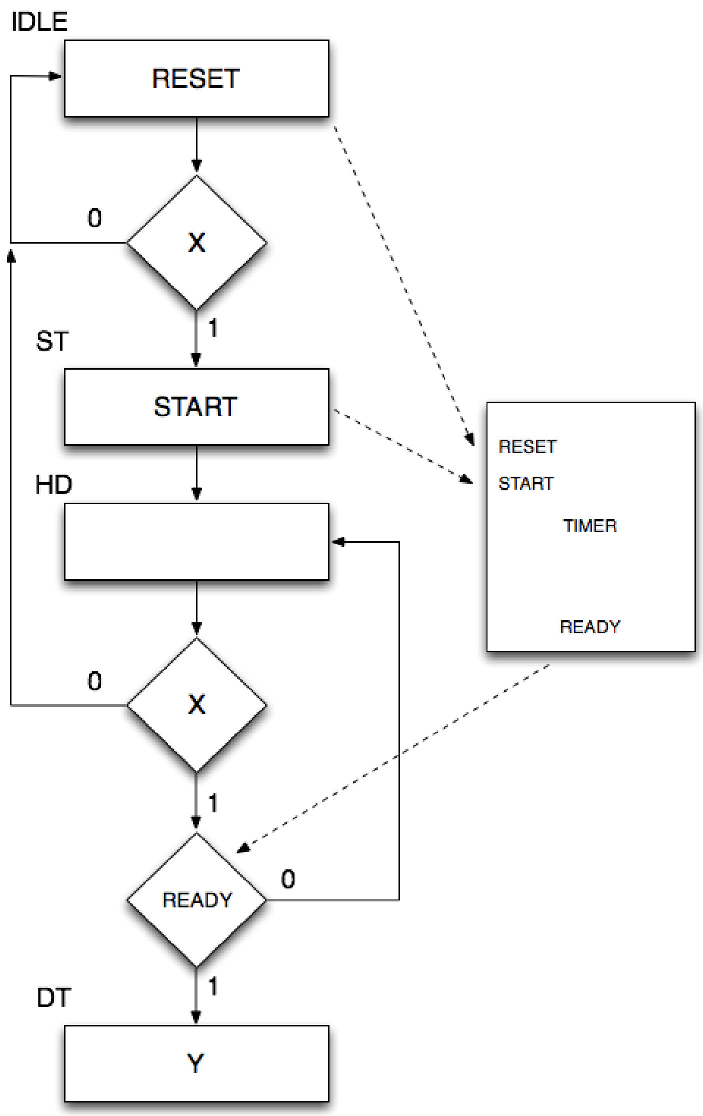


Figure 2. ASM Chart for a switch controller. This system has three outputs (RESET, START and Y) and two inputs (X and READY)

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.

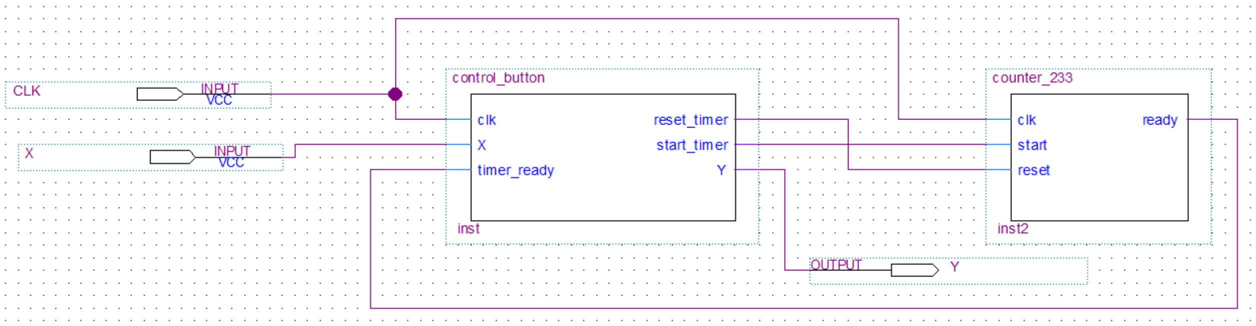


Figure 3. Showing the coupling of two state machines using the schematic editor

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;
```

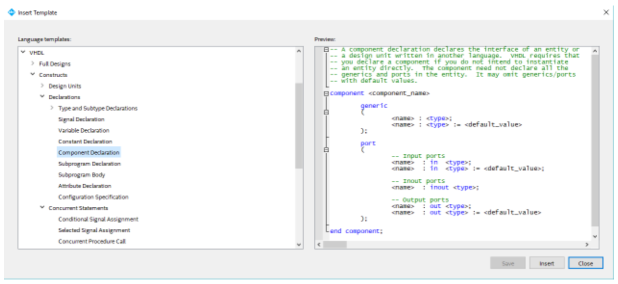
Task 01-03 Linking the Switch Controller to the Timer with Structural VHDL

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?

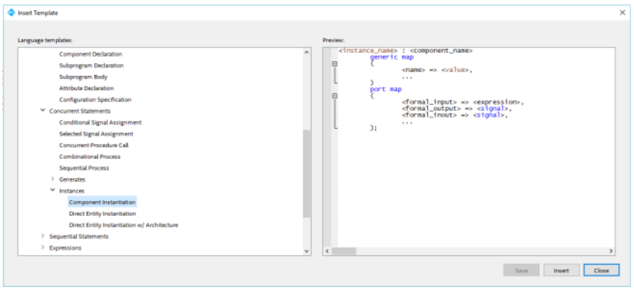
1. Refer to the previous module where we covered **structural VHDL**.
2. Now create a third entity in VHDL called button_hold as shown below
3. Your task is to create the architecture and use structural VHDL to import and connect both the timer and the switch controller.
4. You should then create a symbol 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.

```vhdl
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).



```vhdl
-- A component declaration declares the interface of an entity or
-- a design unit written in another language.  VHDL requires that
-- you declare a component if you do not intend to instantiate
-- an entity directly.  The component need not declare all the
-- generics and ports in the entity.  It may omit generics/ports
-- with default values.

component <component_name>

        generic
        (
                <name> : <type>;
                <name> : <type> := <default_value>
        );

        port
        (
                -- Input ports
                <name>  : in  <type>;
                <name>  : in  <type> := <default_value>;

                -- Inout ports
                <name>  : inout <type>;

                -- Output ports
                <name>  : out <type>;
                <name>  : out <type> := <default_value>
        );

end component;
```



```vhdl
<instance_name> : <component_name>
        generic map
        (
                <name> => <value>,
                ...
        )
        port map
        (
                <formal_input> => <expression>,
                <formal_output> => <signal>,
                <formal_inout> => <signal>,
                ...
        );
```

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.

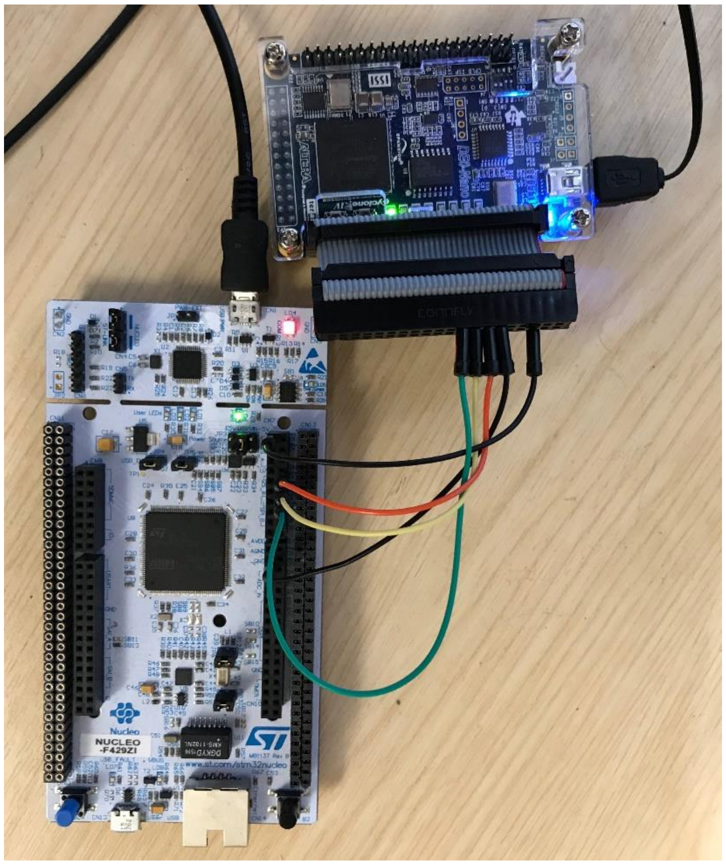


Figure 4. Showing the connections between the FPGA and the Nucleo Board

2. 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

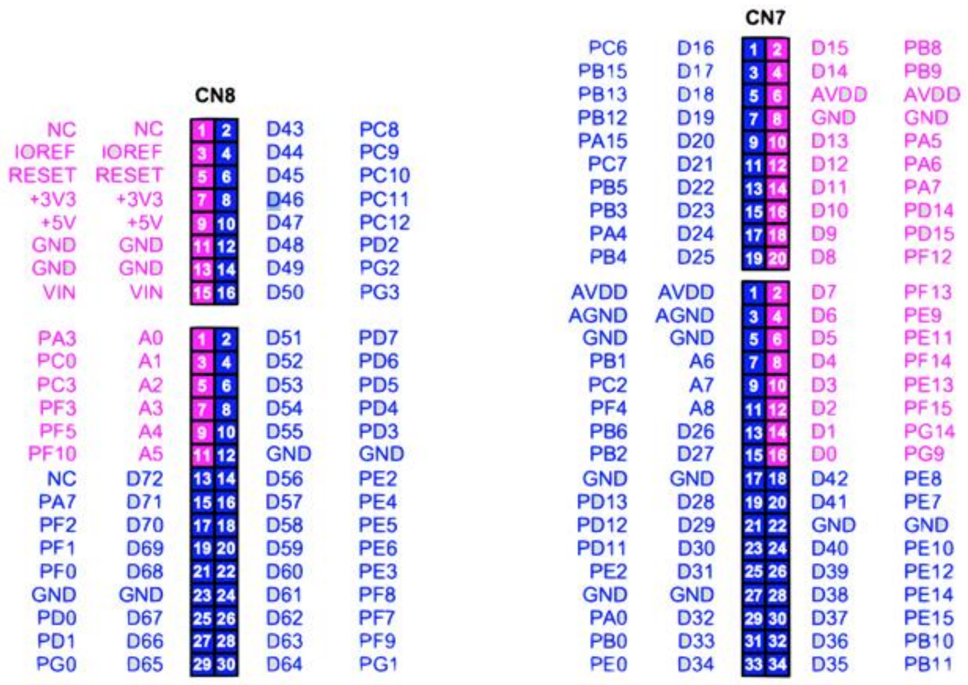


Figure 5 Nucleo Pinouts

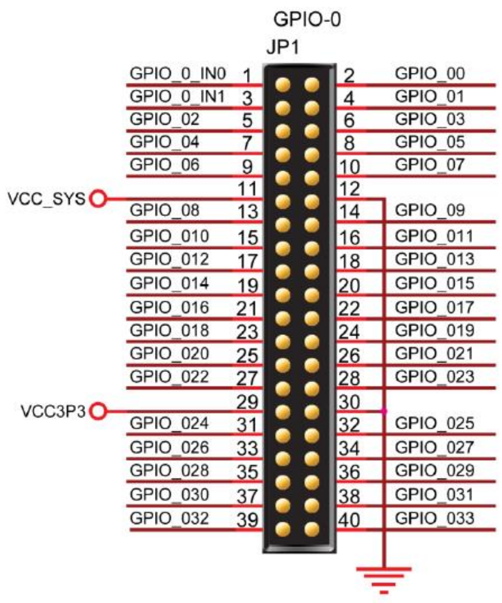


Figure 6. DE0-Nano Pinouts for JP1

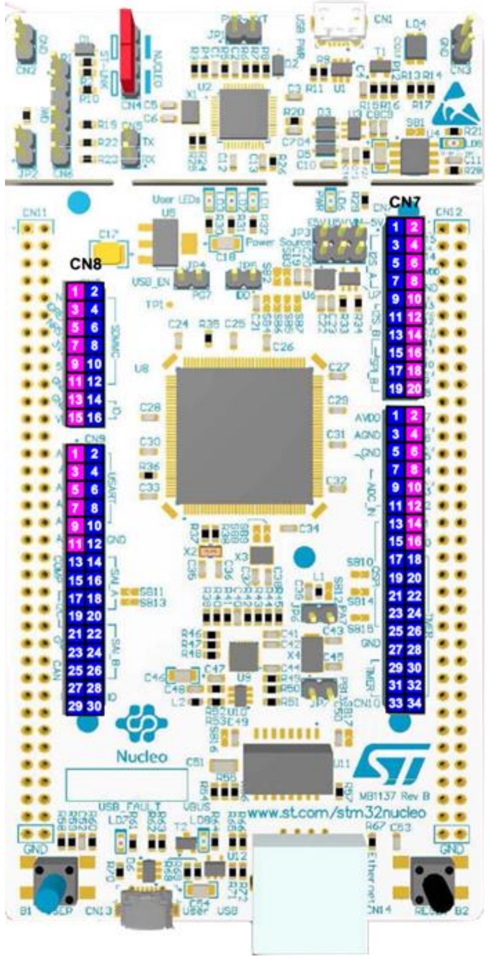


Figure 7. Nucleo FZ429ZI

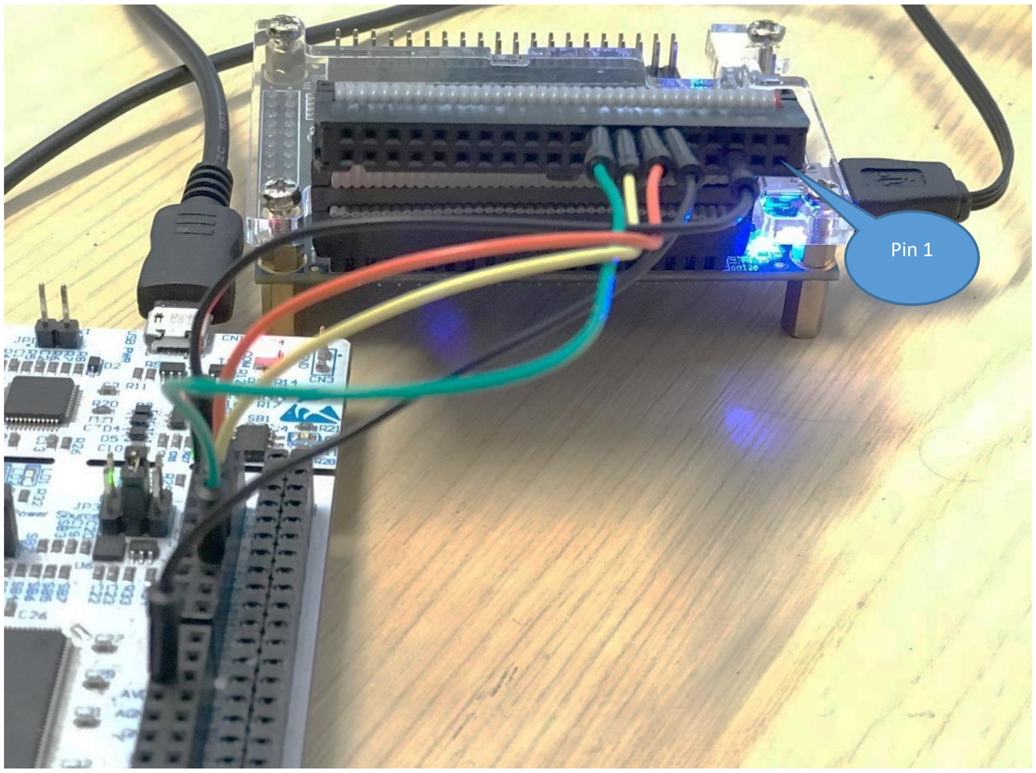


Figure 8. Showing the pin 1 on the FPGA DE-0 Nano Board

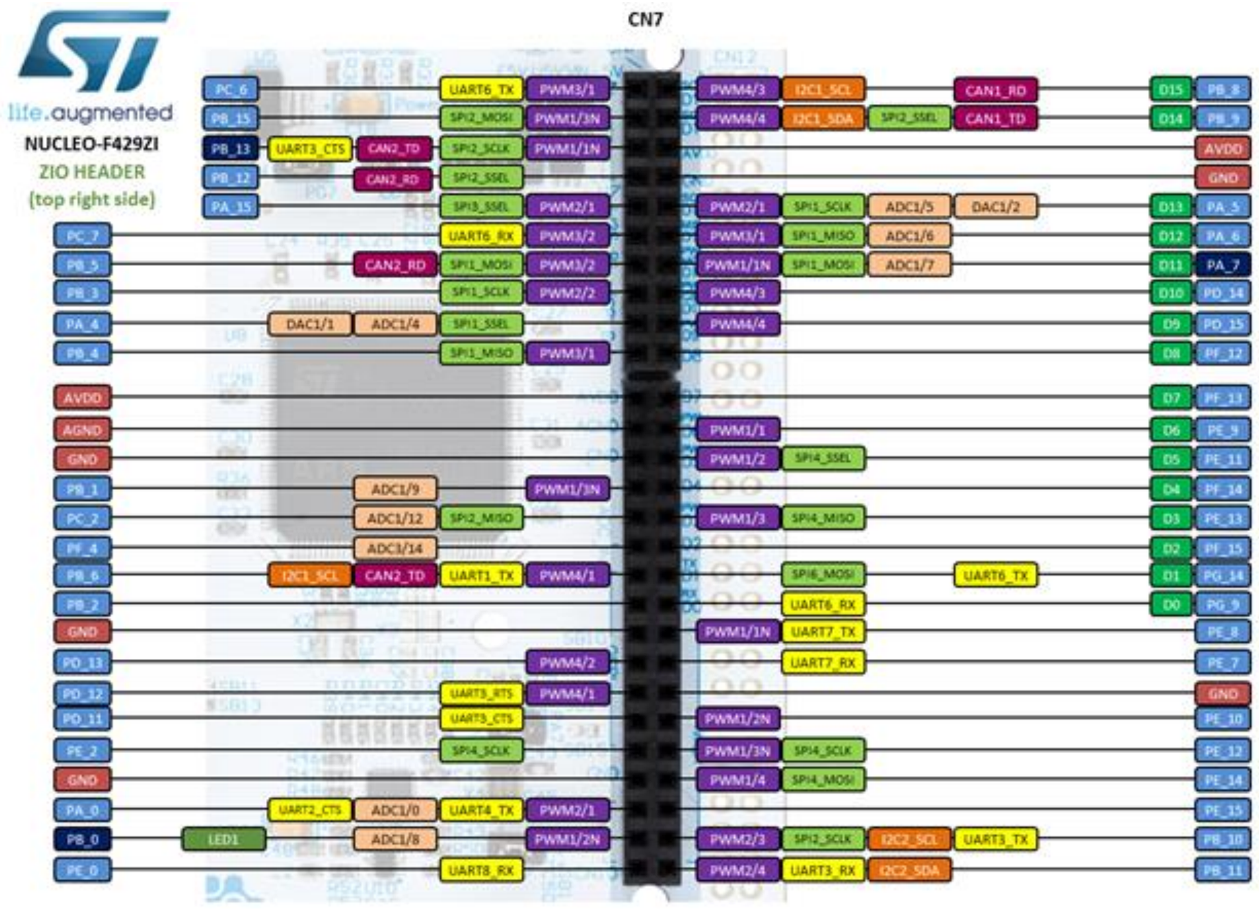
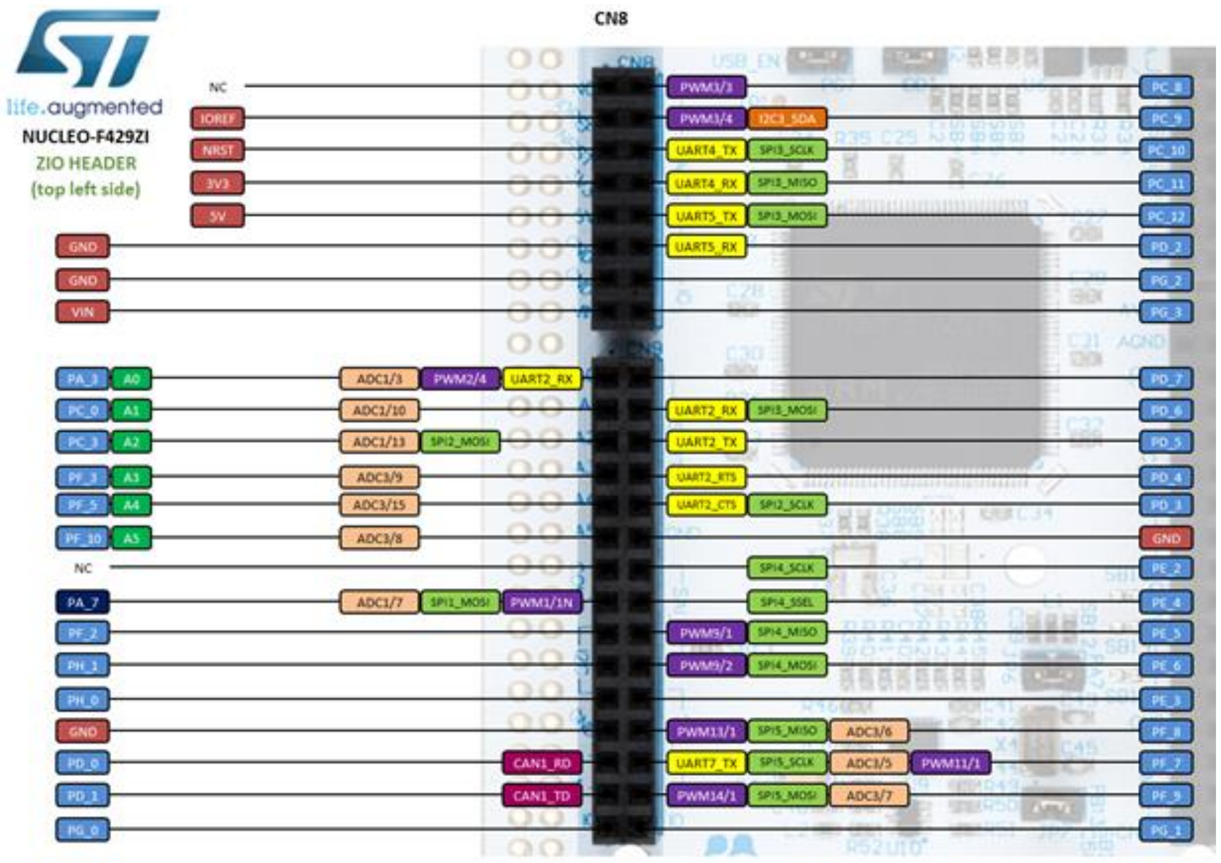

CN7
life.augmented
NUCLEO-F429ZI
ZIO HEADER
(top right side)
PC_6 UART6_TX PWM3/1 PWM4/3 I2C1_SCL CAN1_RD D15 PB_8
PB_15 SPI2_MOSI PWM1/3N PWM4/4 I2C1_SDA SPI2_SSEL CAN1_TD D14 PB_9
PB_13 UART3_CTS CAN2_TD SPI2_SCLK PWM1/1N AVDD
PB_12 CAN2_RD SPI2_SSEL GND
PA_15 SPI3_SSEL PWM2/1 PWM2/1 SPI1_SCLK ADC1/5 DAC1/2 D13 PA_5
PC_7 UART6_RX PWM3/2 PWM3/1 SPI1_MISO ADC1/6 D12 PA_6
PB_5 CAN2_RD SPI1_MOSI PWM3/2 PWM1/1N SPI1_MOSI ADC1/7 D11 PA_7
PB_3 SPI1_SCLK PWM2/2 PWM4/3 D10 PD_14
PA_4 DAC1/1 ADC1/4 SPI1_SSEL PWM4/4 D9 PD_15
PB_4 SPI1_MISO PWM3/1 D8 PF_12
AVDD D7 PF_13
AGND PWM1/1 D6 PE_9
GND PWM1/2 SPI4_SSEL D5 PE_11
PB_1 ADC1/9 PWM1/3N D4 PF_14
PC_2 ADC1/12 SPI2_MISO PWM1/3 SPI4_MISO D3 PE_13
PF_4 ADC1/14 D2 PF_15
PB_6 I2C1_SCL CAN2_TD UART1_TX PWM4/1 SPI6_MOSI UART6_TX D1 PG_14
PB_2 UART6_RX D0 PG_9
GND PWM1/1N UART7_TX PE_8
PD_13 PWM4/2 UART7_RX PE_7
PD_12 UART3_RTS PWM4/1 GND
PD_11 UART3_CTS PWM1/2N PE_10
PE_2 SPI4_SCLK PWM1/3N SPI4_SCLK PE_12
GND PWM1/4 SPI4_MOSI PE_14
PA_0 UART2_CTS ADC1/0 UART4_TX PWM2/1 PE_15
PB_0 LED1 ADC1/8 PWM1/2N PWM2/3 SPI2_SCLK I2C2_SCL UART3_TX PB_10
PE_0 UART8_RX PWM2/4 UART3_RX I2C2_SDA PB_11

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


CN8
life.augmented
NUCLEO-F429ZI
ZIO HEADER
(top left side)
NC
IOREF
NRST
3V3
5V
GND
GND
VIN
PA_3 A0 ADC1/3 PWM2/4 UART2_RX
PC_0 A1 ADC1/10
PC_3 A2 ADC1/13 SPI2_MOSI
PF_3 A3 ADC3/9
PF_5 A4 ADC3/15
PF_10 A5 ADC3/8
NC
PA_7 ADC1/7 SPI1_MOSI PWM1/1N
PF_2
PH_1
PH_0
GND
PD_0 CAN1_RD
PD_1 CAN1_TD
PG_0
PWM3/3 PC_8
PWM3/4 I2C3_SDA PC_9
UART4_TX SPI3_SCLK PC_10
UART4_RX SPI3_MISO PC_11
UART5_TX SPI3_MOSI PC_12
UART5_RX PD_2
PG_2
PG_3
PD_7
UART2_RX SPI3_MOSI PD_6
UART2_TX PD_5
UART2_RTS PD_4
UART2_CTS SPI2_SCLK PD_3
GND
SPI4_SCLK PE_2
SPI4_SSEL PE_4
PWM9/1 SPI4_MISO PE_5
PWM9/2 SPI4_MOSI PE_6
PE_3
PWM13/1 SPI5_MISO ADC3/6 PF_8
UART7_TX SPI5_SCLK ADC3/5 PWM11/1 PF_7
PWM14/1 SPI5_MOSI ADC3/7 PF_9
PG_1

Appendix B – ASM Charts

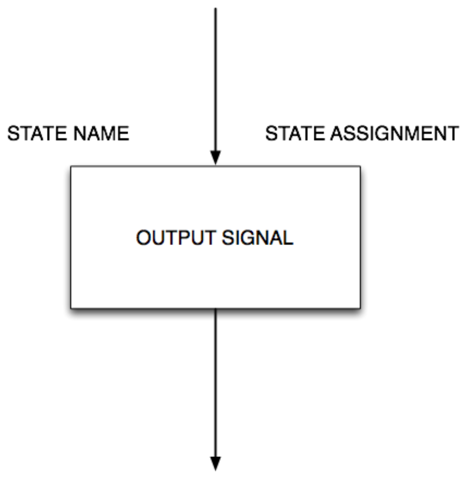


Figure 10 State with Moore Output Signal. The state name and assignment (optional) area all unique

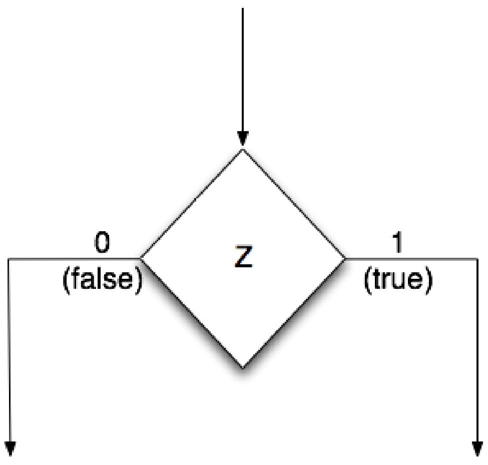


Figure 11 State Transition Decision. The variable Z is being tested. This must ultimately lead to a state box, possibly via another decision and/or Mealy Output(s)

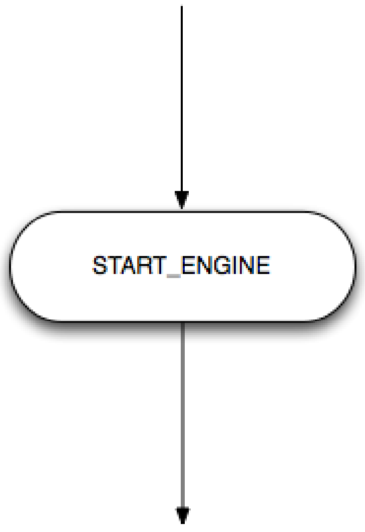


Figure 12. Mealy Output. The example here is the START_ENGINE signal which is asserted high. It only makes sense to connect a decision symbol to the input of the Mealy output symbol.