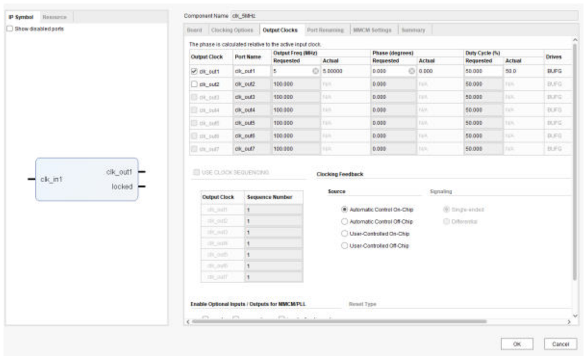
1. Introduction

Generating complex FPGA circuits requires developers to use pre-existing circuit pieces (which can often be acquired through a 3rd party) to increase productivity and reduce production latency. Some complex architectural resources such as clocks must be configured and instantiated instead of inferred. Furthermore, there are also commonly used complex circuits such as the Reed-Solomon decoder which can be quickly synthesised by leveraging the intellectual property (IP) Catalogue tools available through the IP Catalogue. The purpose of this lab is to use the Architectural Wizard and IP catalogue to generate a 1-second pulse generator and a 7-segment display (SSD) counter.

2. 1-second pulse generator

On the Nexys4 DDR board, a 100 Mhz clock source is used. This clock source can be used to generate several clocks at different frequencies and phase shifts using architectural resources called Mixed-Mode Clock Manager (MMCM) and Phased Locked Loop (PPL). The clocking source generator can be invoked by accessing the Clocking Wizard entry under the Clocking sub-folder of the FPGA Feature and Design folder of the IP Catalogue.



Component Name clk_5MHz

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz) Requested	Output Freq (MHz) Actual	Phase (degrees) Requested	Phase (degrees) Actual	Duty Cycle (%) Requested	Duty Cycle (%) Actual	Drives
✓ clk_out1	clk_out1	5	5.00000	0.000	0.000	50.000	50.0	BUFG
clk_out2	clk_out2	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
clk_out3	clk_out3	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	BUFG

USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source: [✓ Automatic Control On-Chip] Automatic Control Off-Chip User-Controlled On-Chip User-Controlled Off-Chip

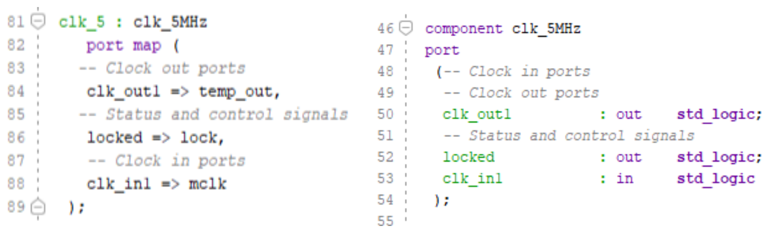
Signaling: [✓ Single-ended] Differential

Enable Optional Inputs / Outputs for MMCM/PLL

Reset Type

Figure 2.1: Clocking wizard from the IP catalogue which allows developers to leverage 3rd party components.

As shown in Figure 2.1, the clocking wizard from the IP catalogue can be used to generate a clock when imputed a frequency, however, the lowest frequency that the custom clock can output at the moment is 5 Mhz. In addition, the MMCM settings for this 5 Mhz clock will be left untouched as the current jitter settings, clock divisor values, and phase shift values will suffice.

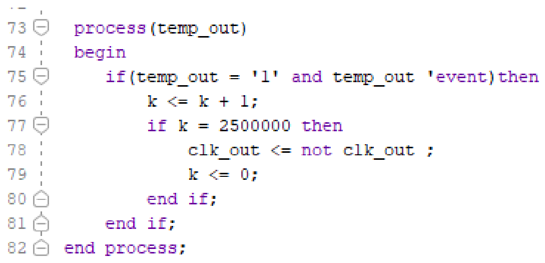


```
clk_5 : clk_5MHz
   port map (
  -- Clock out ports
   clk_out1 => temp_out,
  -- Status and control signals
   locked => lock,
   -- Clock in ports
   clk_in1 => mclk
 );
```

```
component clk_5MHz
port
 (-- Clock in ports
  -- Clock out ports
  clk_out1          : out    std_logic;
  -- Status and control signals
  locked            : out    std_logic;
  clk_in1           : in     std_logic
 );
```

Figure 2.2: Insatianted a 5 Mhz clock from the IP catalogue.

Invoking this 5 Mhz clock as a component of a circuit through the clocking wizard; as shown in Figure 2.2, reduces the time spent on the circuit development process, as developers do not need to metaphorically "reinvent the wheel". This also ensures that the IP of the 5 Mhz developer is protected and can not be easily replicated for profit, similarly to software developers protecting their IP. Although the 5 Mhz clock can be added as part of the circuit design, the contents of the 5 Mhz clock component can not be observed in Vivado.



```
 process(temp_out)
 begin
    if(temp_out = '1' and temp_out 'event)then
        k <= k + 1;
        if k = 2500000 then
            clk_out <= not clk_out ;
            k <= 0;
        end if;
    end if;
end process;
```

Figure 2.3: temp_out from the 5 Mhz clock used in a counter to toggle the output of the 1 Hz clock

Finally, a 1 Hz clock can be achieved by conditionally toggling the 5 Mhz clock output signal. After counting up to half the clock period of the desired frequency, the output will toggle, generating a clock. This can be illustrated in the code snippet of Figure 2.3. The count-up value can be calculated by using Equation 1, which determines the toggle count-up value for the desired frequency.

$$\frac{5 * 10^6}{2 *} = \quad ,$$

Equation 1: desired output frequency with 50% duty cycle

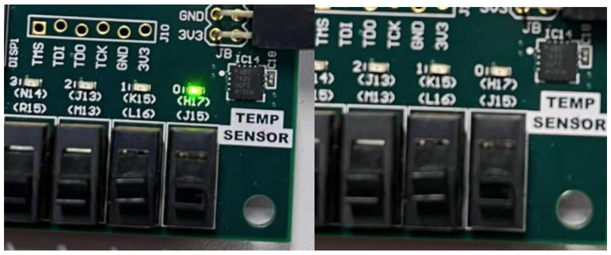


Figure 2.3: LED toggles between 1 and 0 every second

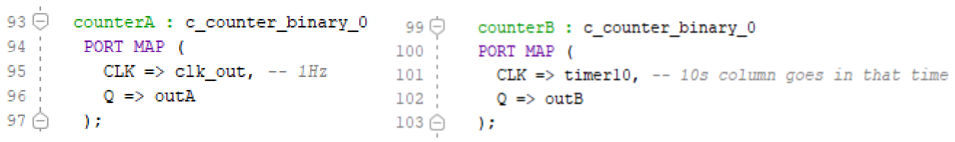
As shown in Figure 2.3 the LED switches on and off every second. This sequential method approach to reducing the 5 Mhz clock output to a 1 Hz clock output was picked over a bit divider approach. This is because of bit divisibility error, as there are 22.23 bits for the 5 Mhz clock to divide down by, it is impractical to implement as the clock output will result in inaccuracy as shown in the bit divider equation below.

$$\frac{5*10^6}{2} = 1 \quad , \quad = 22.23 \text{ bits}$$

Equation 2: bit divider equation

3. A 4-bit counter on a double-digit 7-segment display.

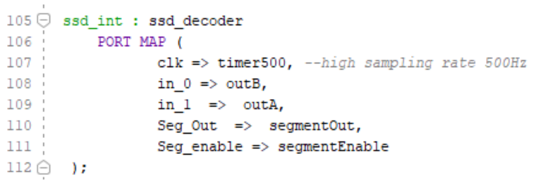
To further demonstrate the development capabilities of employing 3rd party components through the IP catalogue, a double-digit 7-segment display counter can be created by invoking 4-bit binary counters and a double-digit 7-segment display, similar to the 5 MHz clocking wizard. As shown in Figure 3.1, the IP catalogue binary counter was instantiated after being added. Consequently, the 2-digit SSD_decoder was instantiated after being added as a component from the IP catalogue as shown in Figure 3.2



```
counterA : c_counter_binary_0
 PORT MAP (
   CLK => clk_out, -- 1Hz
   Q => outA
 );
```

```
counterB : c_counter_binary_0
PORT MAP (
  CLK => timer10, -- 10s column goes in that time
  Q => outB
);
```

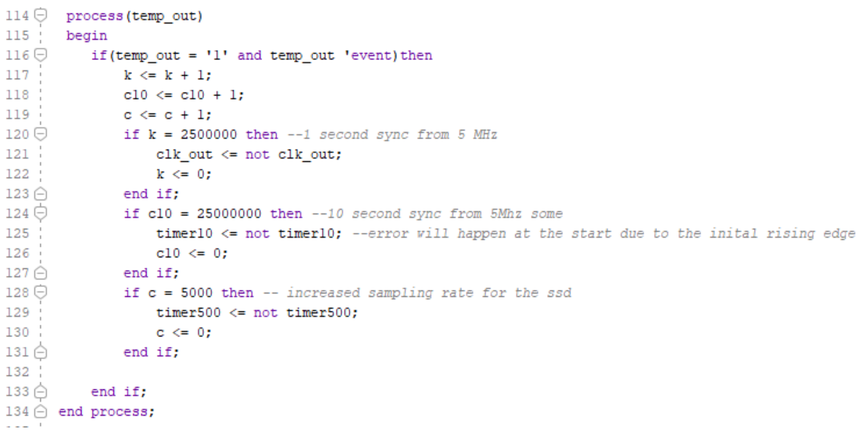
Figure 3.1: instantiated 4-bit binary counters from the IP catalogue.



```
ssd_int : ssd_decoder
    PORT MAP (
            clk => timer500, --high sampling rate 500Hz
            in_0 => outB,
            in_1  =>  outA,
            Seg_Out  =>  segmentOut,
            Seg_enable => segmentEnable
 );
```

Figure 3.2: instantiated double-digit SSD decoder from the IP catalogue.

This binary counter component will be instantiated twice so that one column would be ticking every second and the other column would be ticking every 10-seconds. As one of the columns will be ticking every 10-seconds, the count-up toggle value can be calculated using equation 1; resulting in the count-up value being “250000”. The behavioural structure code can be seen in Figure 3.3 which encompasses the count-up values of the refresh rate, 10-second count-up, and the 1-second count-up. Furthermore, the refresh rate of the SSD is clocked at 500 Hz (which when inputted to Equation 1, produces a count-up value of 5000) as recommended by the lab script to prevent flickering.



```
 process(temp_out)
 begin
    if(temp_out = '1' and temp_out 'event)then
        k <= k + 1;
        c10 <= c10 + 1;
        c <= c + 1;
        if k = 2500000 then --1 second sync from 5 MHz
            clk_out <= not clk_out;
            k <= 0;
        end if;
        if c10 = 25000000 then --10 second sync from 5Mhz some
            timer10 <= not timer10; --error will happen at the start due to the inital rising edge
            c10 <= 0;
        end if;
        if c = 5000 then -- increased sampling rate for the ssd
            timer500 <= not timer500;
            c <= 0;
        end if;

    end if;
end process;
```

Figure 3.3: behavioural structure clock breakdown 5 Mhz to 500 Hz, 1 Hz, and 0.1 Hz.

Finally, the output of these 2 binary counters would be propagated towards the 2-digit SSD as shown in Figure 3.2. The refresh rate of the SSD must be greater than the binary counter-changing speed. This is because a low refresh rate will result in the flickering of the SSD. As shown in Figure 3.4, the 2-digit SSD display driven 2 binary counters ticks accordingly.

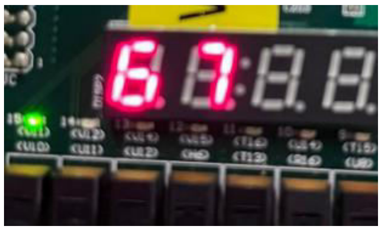


Figure 3.4: double-digit SSD display created using behavioural structural and IP catalogue

Conclusion

In conclusion, development time for FPGA circuit design can be reduced drastically through the use of 3rd party components accessed from the IP catalogue. This includes the Architecture Wizard, which allows users to create a custom clock such as the 5 Mhz clock used in this lab. Furthermore, 3rd party components such as the 4-bit binary counter and 2-digit SSD decoder were created from the IP catalogue which accelerated the time it took for this system to be developed.