DAT093
Introduction to Electronic System Design
Hints on clocking

In many cases, there is a need to synchronize the code using a clock frequency that is lower than the frequency of the system clock, the lower frequency could for example be a sampling clock.

The only real timing element we have in our designs is the system clock, unless we introduce another external clock. To avoid this external clock the normal way to generate the lower frequency clock is to use the system clock and count a number of clock cycles from the system clock, toggle the lower frequency clock signal, count again, toggle again and so on. We can only use one flank of the system clock, positive or negative, to trigger the counter and the result is that the lower frequency clock can only have a frequency that is the system clock frequency divided by an integer number and if we use the same count for the high and the low half period this integer must be an even number. If the number is odd then we will have to generate a slightly asymmetrical clock.

It is good practice to if possible only use one clock signal in a design. This means that we should not trigger our clocked design on the generated low speed signal but instead trigger on the system clock and then use *clock enable* signals for the lower frequencies. The reason for this is that the clock signals will be distributed through separate clock nets within the FPGA and the FPGA contains only a limited number of these nets and these nets cannot be freely routed. By using clock enables there are still just a few clock signals, often only one, while the enable signals are distributed like all other signals and can be connected to flip-flops using their enable inputs.

This means that for the internal synchronization we should **not** use signals like the ones in *Figure 1*

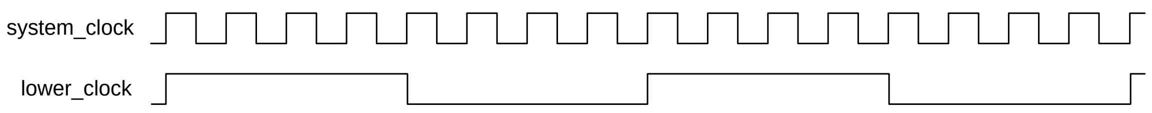

Figure 1 System clock and divided clock

and write code like

```
IF rising_edge(system_clock) THEN
   .. .. ..
   .. .. ..
END IF;
```

```
IF rising_edge(lower_clock) THEN
   .. .. ..
   .. .. ..
END IF;
```

Instead we should use an enable signal for the lower frequency as shown in *Figure 2*

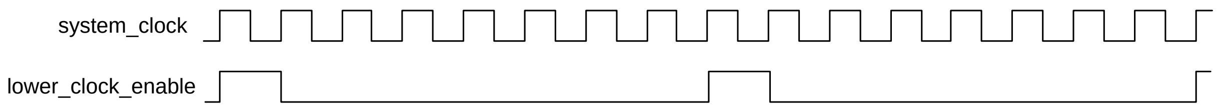

Figure 2 System clock and clock enable for the divided clock

and write code like

```
IF rising_edge(system_clock) THEN
   IF (lower_clock_enable='1') THEN
      .. .. ..
      .. .. ..
   END IF;
END IF;
```

Experience says that the design will have a better chance of functioning as expected if the two IF clauses are kept separate and not merged into on IF clause so don´t write

```
IF (rising_edge(system_clock) AND
                       (lower_clock_enable='1')) THEN
      .. .. ..
      .. .. ..
   END IF;
END IF;
```

Doing it this way we will only route the system clock signal through the clock nets while the enable signal will be treated like a normal signal and it will be routed through the common, more frequent, nets and connect to the flip-flops through enable inputs.

The clock enable signal is generated in a similar way to the generation of the lower frequency clock described earlier. We count system clock cycles by using a modulus counter where the count value is the number of system clock pulses between each enable signal and we activate the lower clock enable signal during one and only one system clock period of each counting cycle. This will not take more hardware than creating a clock signal with a lower frequency since we need a counter in both cases.

Observe that to generate the same lower clock frequency we will need different counter ranges in the two cases. In the first case (*Figure 1*) we need to do two counts for one period, one count for the low part of the period and one count for the high part of the period. In the second case (*Figure 2*) we need one count cycle per period of the lower frequency but we need to count up to a higher, double value.

Clock enable applied to lab assignment 5

Let´s see how this applies to *Laboratory assignment 5*.

In this assignment you will design interfaces to an ADC and a DAC both connected through SPI busses. The transfer on the busses is controlled by the serial clock, the *SPI clock*. To follow the rules above you should introduce a *SPI clock enable signal* to synchronize the transfers. In this case you can´t leave out a symmetrical SPI clock though.

The external SPI devices need the SPI clock signal to have a more symmetric period to function as intended, that is you need to connect a symmetrical SPI clock to the external SCLK pins connected to the ADC and the DAC but it is still best to use clock enable signals within your design.

When you set up the SPI clock signal and the SPI clock enable signal it is natural to start each period with a clock enable signal but then you have two options for the SPI clock

- You could start with a high SPI clock signal, *Figure 3*
- You could start with a low SPI clock signal, *Figure 4*

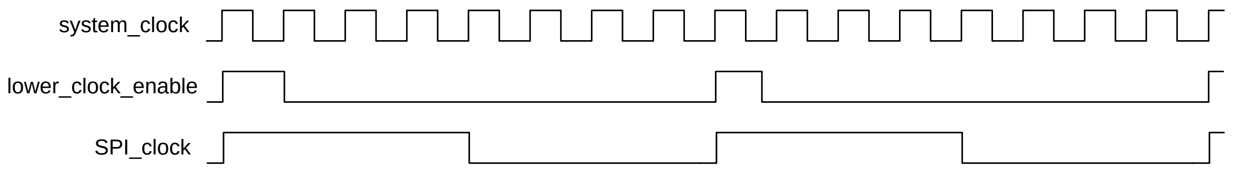

Figure 3 Start with a high SPI clock

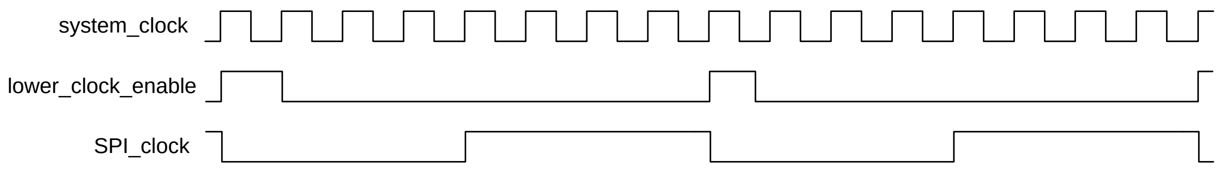

Figure 4 Start with a low SPI clock

Does the choice matter? Let´s look at the timing for the two interfaces, *Figure 5* and *Figure 6*.

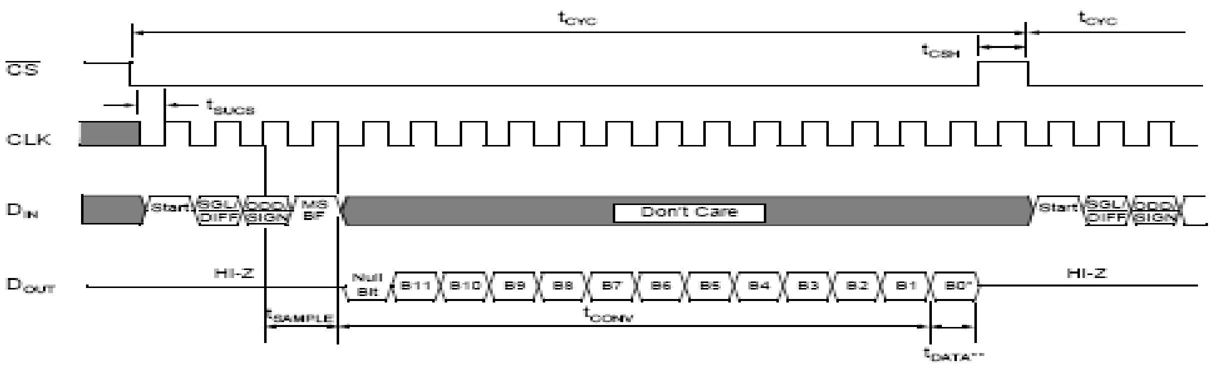

Figure 5 Timing for the ADC interface to the MCP3202

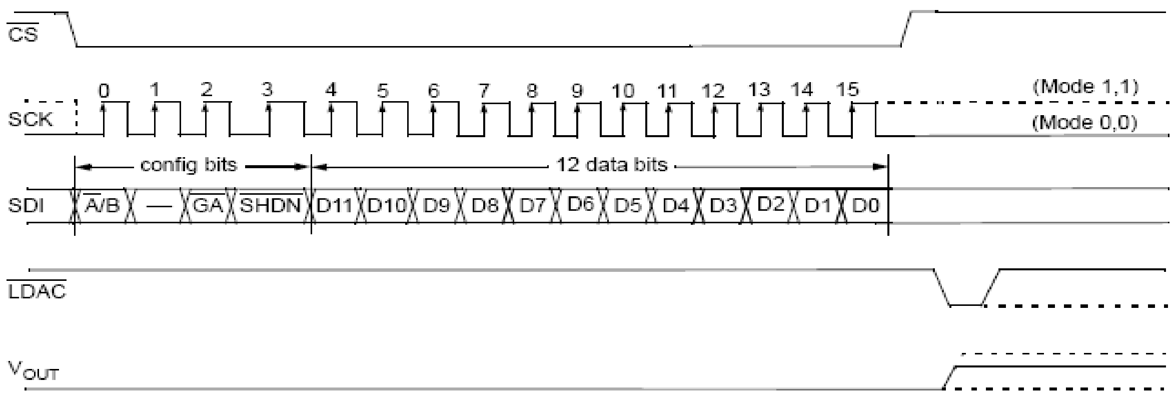

Figure 6 Timing for the DAC interface to the MCP4822

From the timing diagrams, we can see that we must first place data or configuration bits on the relevant line and then the data will be read into the external device on the rising edge of the SPI clock.

One way to do this is to use a SPI clock that starts high, *Figure 3*, and place the data on the line at one rising edge and then trigger it on the next rising edge. This is possible since the placing of the data is also triggered by the rising edge of the SPI clock which means that when we place the data on the line the current rising edge has already passed and it will not be triggered until the next rising edge.

It might seem simpler to use a SPI clock that starts low though, *Figure 4*. In this case we get a natural flow where we place data on the line at the SPI clock enable signal and then the data will be read by the external device when the rising edge of the SPI clock comes half a period of the SPI clock signal later.