Rotary Encoder Interface
for
Spartan-3E Starter Kit

Ken Chapman
Xilinx Ltd
20th February 2006

With thanks to Peter Alfke (Xilinx Inc.)

Rev.2

Limitations

Limited Warranty and Disclaimer. These designs are provided to you “as is”. Xilinx and its licensors make and you receive no warranties or conditions, express, implied, statutory or otherwise, and Xilinx specifically disclaims any implied warranties of merchantability, non-infringement, or fitness for a particular purpose. Xilinx does not warrant that the functions contained in these designs will meet your requirements, or that the operation of these designs will be uninterrupted or error free, or that defects in the Designs will be corrected. Furthermore, Xilinx does not warrant or make any representations regarding use or the results of the use of the designs in terms of correctness, accuracy, reliability, or otherwise.

Limitation of Liability. In no event will Xilinx or its licensors be liable for any loss of data, lost profits, cost or procurement of substitute goods or services, or for any special, incidental, consequential, or indirect damages arising from the use or operation of the designs or accompanying documentation, however caused and on any theory of liability. This limitation will apply even if Xilinx has been advised of the possibility of such damage. This limitation shall apply not-withstanding the failure of the essential purpose of any limited remedies herein.

This design module is **not** supported by general Xilinx Technical support as an official Xilinx Product.
Please refer any issues initially to the provider of the module.

Any problems or items felt of value in the continued improvement this reference design would be gratefully received by the author.

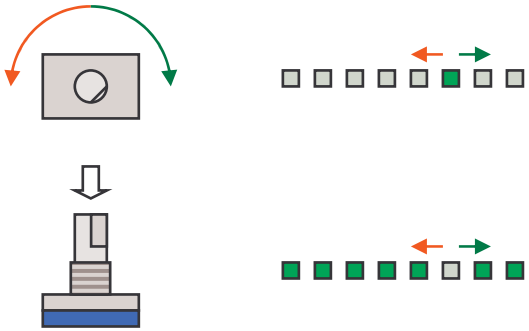
Ken Chapman
Senior Staff Engineer – Spartan Applications Specialist
email: chapman@xilinx.com

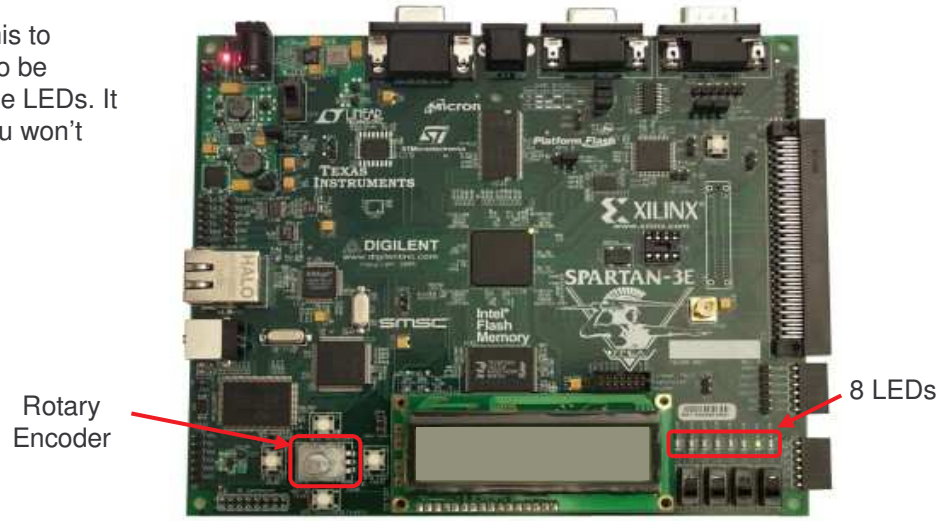
The author would also be pleased to hear from anyone using PicoBlaze (KCPSM3) or the UART macros with information about your application and how these macros have been useful.

Design Overview

This main purpose of this reference design is to provide an interface to the rotary encoder on the Spartan-3E Starter Kit. However, it is hoped that this document combined with the VHDL code and user constraints file (UCF) may provide useful reference for those less familiar with Spartan-3E devices and looking for advice on how to design using them. Some additional exercises are suggested for self education.

The design provided detects left and right rotation of the encoder and uses this to controls which of the 8 simple LEDs is illuminated. The rotary control can also be depressed (with or without simultaneous rotation) and this is used to invert the LEDs. It is only very simple behaviour, but my daughter said it was “cool” and I bet you won’t be able to resist playing with it too!





Things to learn from this design

Interfacing to a rotary encoder.
Constraining pin locations using UCF constraints.
Simple VHLD illustrating synchronous design techniques.
Using the built in pull-up and pull-down resistors in the input/output blocks.
Direct drive of LEDs.
Good design practice for higher performance
  Dedicated flip-flops in the input/output blocks.
  PERIOD timing constraint.
Configuration using iMPACT batch file.

Try it now – it only takes 30 seconds!

As well as the source design files, a compiled configuration bit file is provided which you can immediately download into the Spartan XC3S500E device on your board. It is recommended that you try this to become familiar with what the design does before continuing to read.

To make this task really easy the first time, unzip all the files provided into a directory and then **double click on ‘install_left_right_leds.bat’**. Assuming you have the Xilinx software installed, your board connected with the USB cable and the board powered (don’t forget the switch), then this should open a DOS window and run iMPACT in batch mode to configure the Spartan-3E with the design.

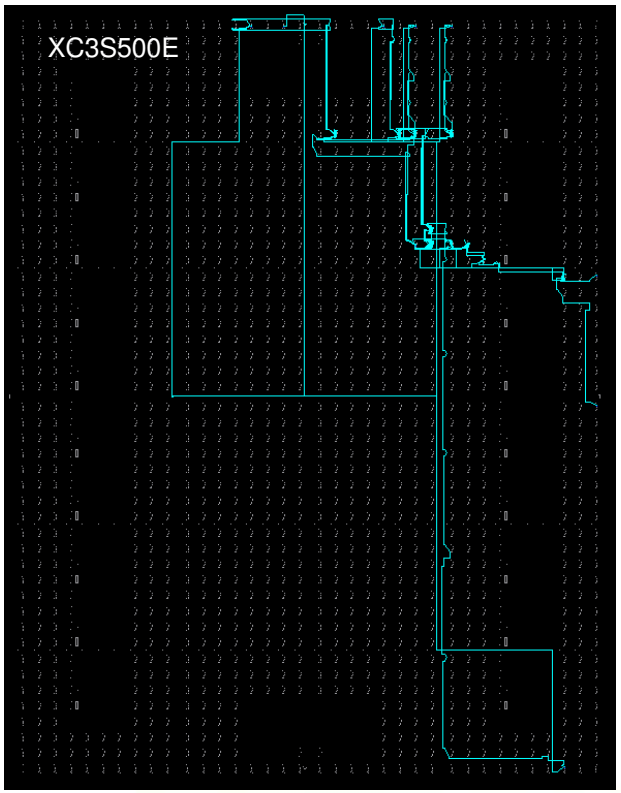
Design Size

This design hardly touches the XCSS500E device actually using just 0.5% of the logic resources in total. Clearly room left for you to add more ☺

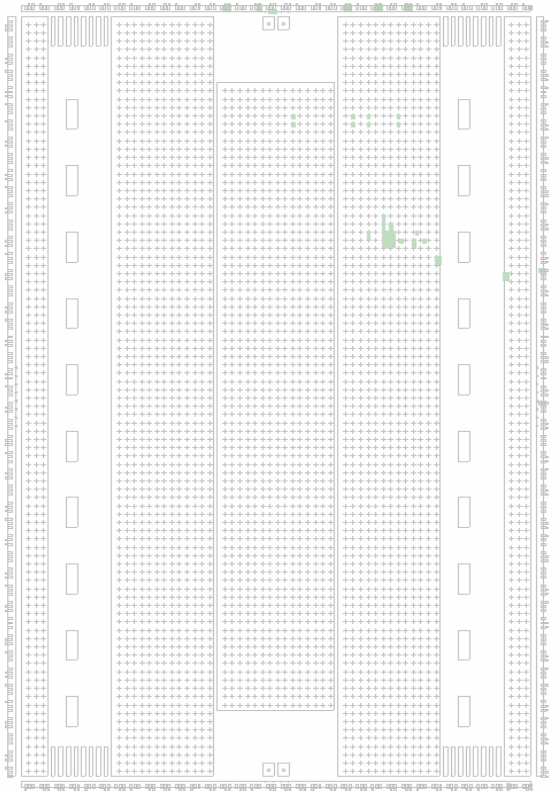
MAP report

```
Number of occupied Slices:     22 out of   4,656    1%

Total equivalent gate count for design:  955
```

FPGA Editor view



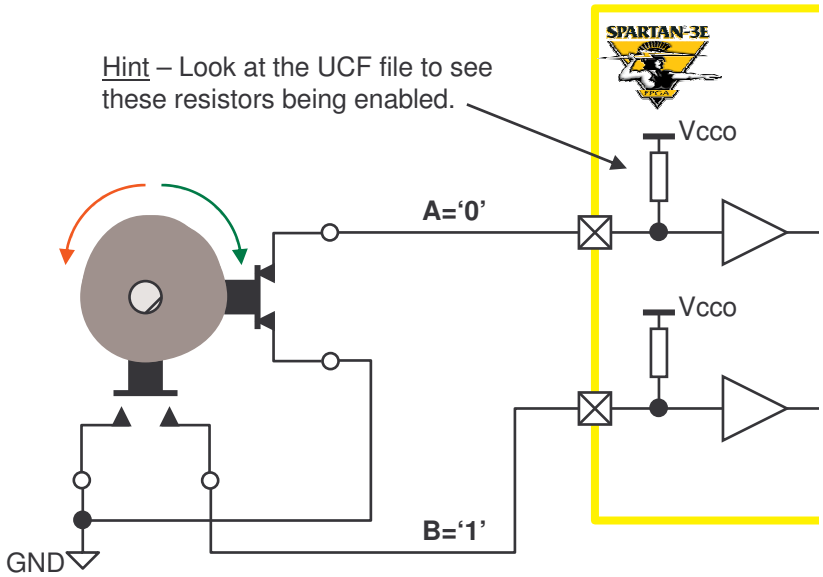
Floorplanner view

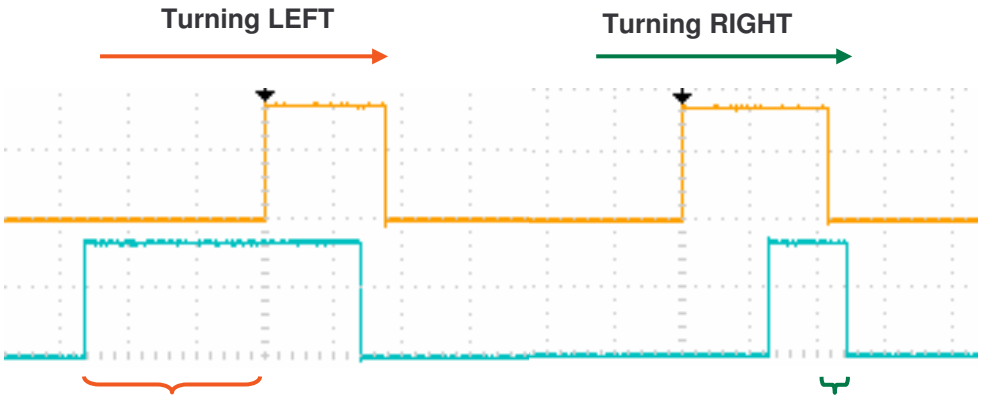


Rotary Encoder & Signals

The Encoder - The basic principle of the rotary encoder is that of a cam connected to a shaft which is used to operate two switches. When in the stationary 'detent' position both switches are closed. Then depending on which way the shaft is rotated, one switch will open before other. Likewise, as the rotation continues, one switch will be closed before the other. This diagram only depicts that one sequence of the switches will occur for every 360˚ revolution. The encoder on the board actually repeats the sequence every 18˚ (20 clicks per revolution).

The Signals - To provide logic signals that the Spartan-3E device can work with, one side of each switch is connected to ground. So when the switch contacts are closed, the signal to the Spartan-3E is definitely Low or '0'. When the switch contacts are opened during rotation, a PULL-UP resistor is required to raise the signal to High or '1'. A nice feature of Spartan input/output blocks (IOB) is that they have built in optional PULL-UP and PULL down resistors which save adding external components.





These oscilloscope traces were captured on the Starter kit and show the signals being detected by the Spartan-3E. The irregularity of the 'pulses' is to be expected given the human input! The red and green brackets indicate the logic levels consistent with the switch positions depicted in the diagram. Note that just looking at the 'A' and 'B' signals as a snapshot is not enough information to evaluate direction. The sequence of signals has to be evaluated.

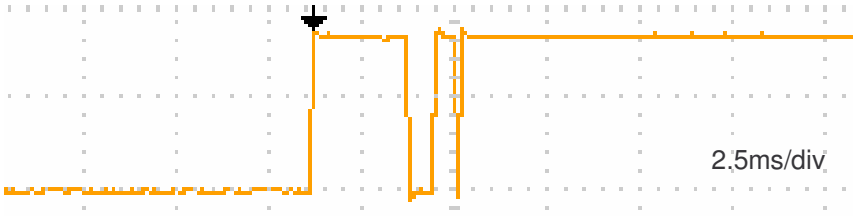
The encoder on the Starter Kit board has a third simple switch which is closed when you press the shaft towards the base. One side of this switch is connected to 3.3v on the board, so in this case a '1' is generated when pressed and an IOB PULL-DOWN resistor is used to provide a '0' at all other times.

Contact Bounce

In the design provided, you should find that each ‘click’ of the shaft moves the illuminated LED one position only. Note that it is one position every time and not suddenly several positions even if you turn it really slowly or click it one position in an ‘instant’ (try it and see). If this control was for tuning your car radio, you would also expect the same reliable operation to home in on the radio station of your choice.

Unfortunately, anything mechanical is prone to bounce and this is very true of switch contacts which are said to ‘chatter’. Although switch manufactures go to considerable lengths to minimise this effect, sometimes we have to design around the issue and that is the case with the rotary encoder interface.

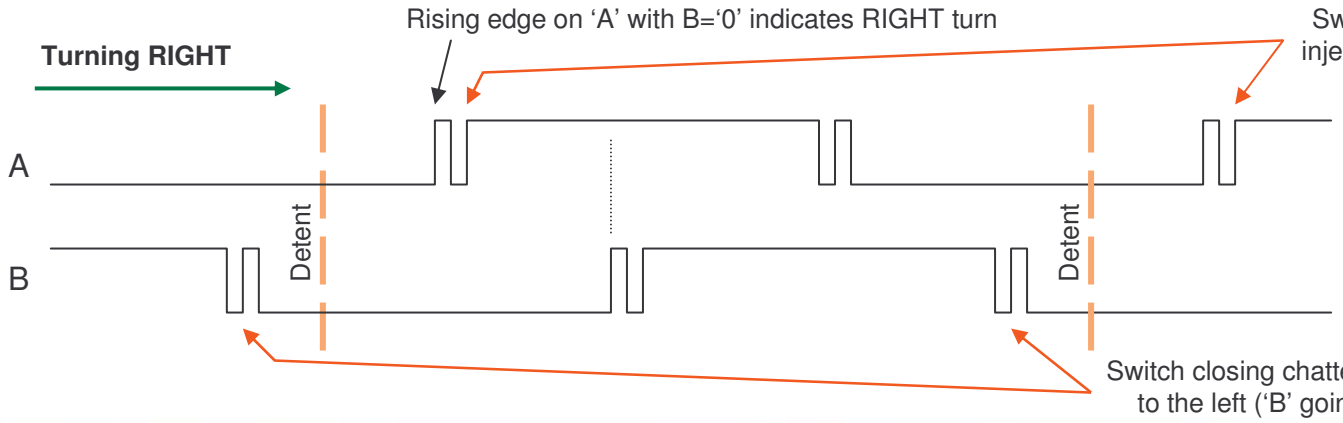
This scope trace shows the chatter observed on the ‘A’ switch on one particular occasion. The switch opened and the signal went High for approximately 2.5ms. It then closed for ~1ms, opened for ~1ms, closed again momentarily, before finally settling on truly being open and providing a steady High signal. Even if other cases are not this bad, the interface must deal with them appropriately when they do occur.



Note – Spartan devices are relatively fast. There are 50,000 clock cycles at 50MHz in the duration of one of these 1ms chatter events.

The initial obvious way to determine the direction of rotation is as follows:-
If ‘A’ goes High whilst ‘B’ is Low then rotation is to the to the Right.
If ‘B’ goes High whilst ‘A’ is Low then rotation is to the to the Left.

The diagram below indicates how switch chatter could be interpreted as additional rotation ‘clicks’ in either direction even when the intention is only to take one step to the right.



Exercise – Implement this ‘obvious’ rotation detector and see how unreliable it is.

Rotary Contact Filter

Clearly the objective of the filter is to eliminate the switch chatter completely. This is a achieved by detecting only the first change of the signal and ignoring all subsequent activity on the same signal until the other switch also changes state. Flip-flops provide the 'memory' for this function.

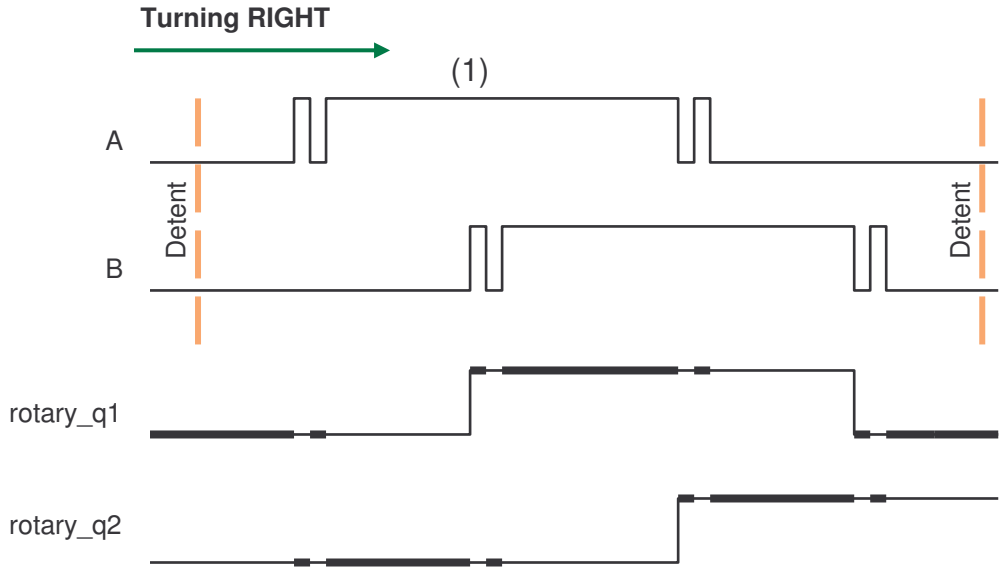
rotary_q1

Set ('1') when A is High and B is High
Reset ('0') when A is Low and B is Low. } i.e. XNOR
Remember current state in all other cases.

rotary_q2

Set ('1') when A is Low and B is High
Reset ('0') when A is High and B is Low. } i.e. XOR
Remember current state in all other cases.

```
rotary_filter: process(clk)
 begin
   if clk'event and clk='1' then

     rotary_in <= rotary_b_in & rotary_a_in;

     case rotary_in is

       when "00" => rotary_q1 <= '0';
                    rotary_q2 <= rotary_q2;

       when "01" => rotary_q1 <= rotary_q1;
                    rotary_q2 <= '0';

       when "10" => rotary_q1 <= rotary_q1;
                    rotary_q2 <= '1';

       when "11" => rotary_q1 <= '1';
                    rotary_q2 <= rotary_q2;

       when others => rotary_q1 <= rotary_q1;
                      rotary_q2 <= rotary_q2;
     end case;

   end if;
 end process rotary_filter;
```

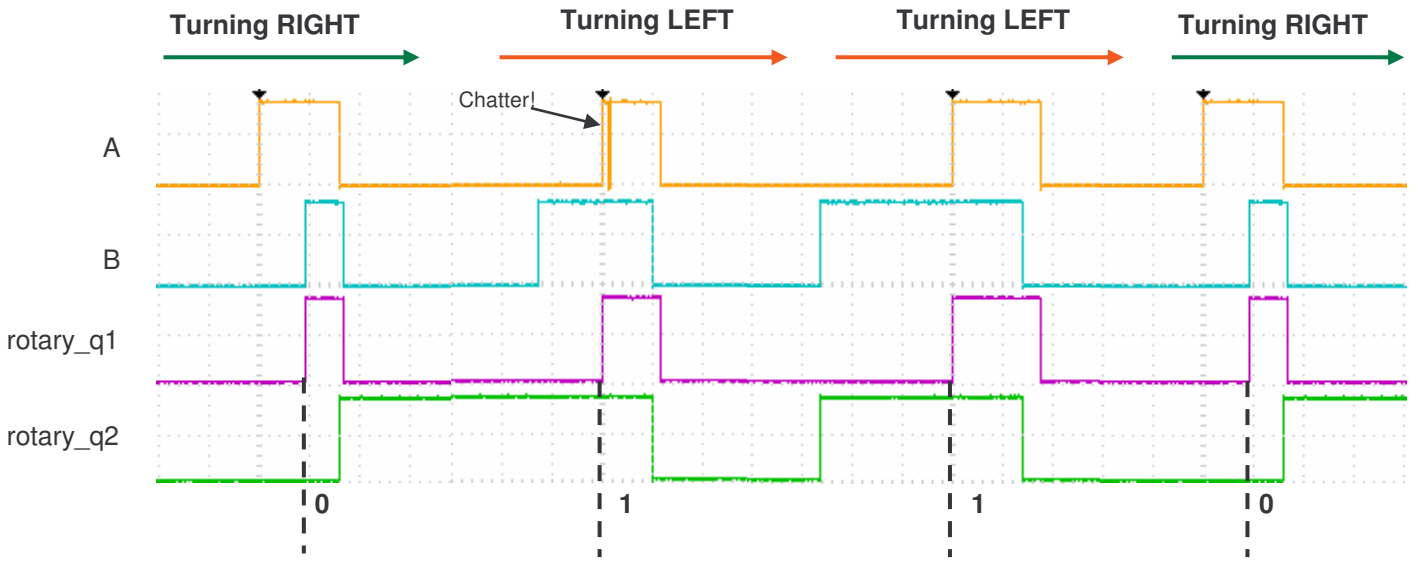


The bold lines indicate where each signal is being forced and the normal lines indicate where the flip-flop memory is retaining the current state. Although we now have a slightly different signal behaviour, the signals are clean and direction can still be determined.

Note that this can all be achieved with a synchronously clocked circuit because the 50MHz clock rate is much faster than any events taking place at the switch. Synchronous design techniques avoid introducing different forms of unreliable behavior as well as leading to maximum efficiency of the Spartan-3E.

Direction and Rotation Events

The following scope traces show the output of the rotary filter compared with the original inputs. It is then possible to determine the direction.



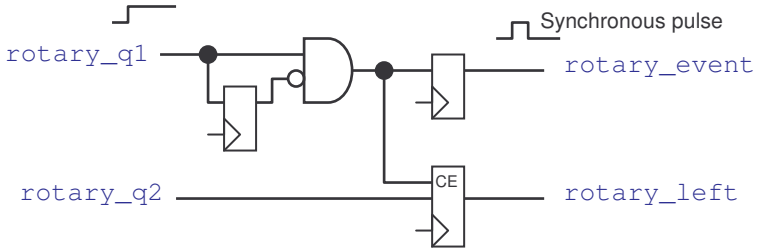
This is not a simulation!

Hint – Never forget to exploit Spartan programmable logic during test and debugging. It was possible to observe the 'rotary_q1' and 'rotary_q2' signals on a real oscilloscope by adding temporary connections to pins assigned to the board J4 connector.

It can be seen that when 'rotary_q1' changes from Low to High, the state of 'rotary_q2' indicates the direction. We can therefore use 'rotary_q1' to determine each event and 'rotary_q2' to indicate direction.

```
direction: process(clk)
 begin
   if clk'event and clk='1' then

     delay_rotary_q1 <= rotary_q1;
     if rotary_q1='1' and delay_rotary_q1='0' then
       rotary_event <= '1';
       rotary_left <= rotary_q2;
      else
       rotary_event <= '0';
       rotary_left <= rotary_left;
     end if;

   end if;
 end process direction;
```

The Low to High transition of 'rotary_q1' is used to form a synchronous pulse and remember the direction of rotation. This maintains synchronous design and is a million times better than using 'rotary_q1' as another clock signal.

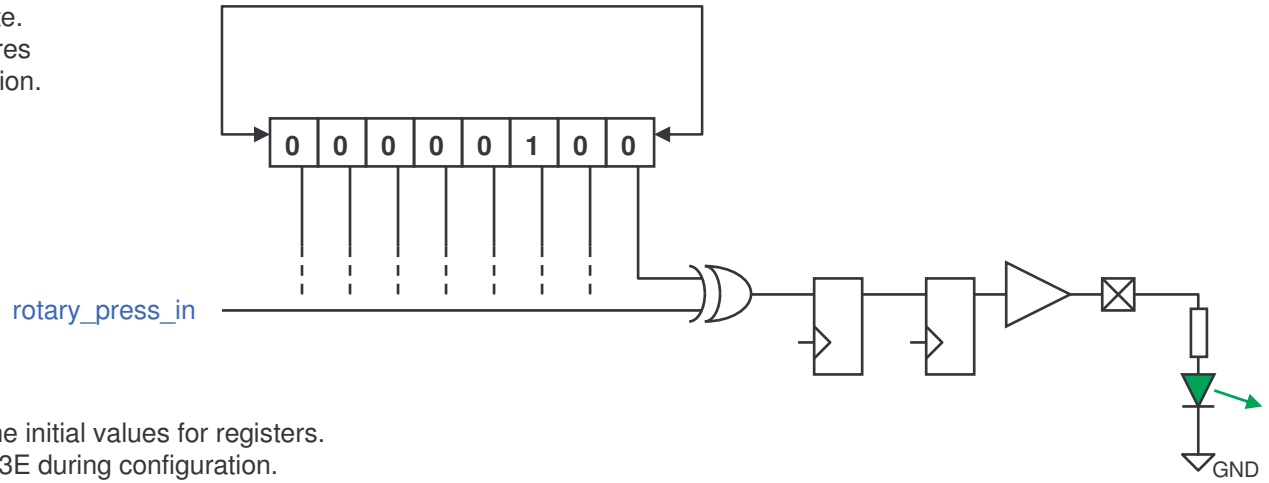


Exercise – Observe 'rotary_event' and 'rotary_left' on an oscilloscope.

Driving LED's

In this design the rotary events are used to control a simple 8-bit sift register. Each shift register bit is then optionally inverted (by pushing the rotary press switch) on the way to the output pin which drives the corresponding LED. A 390Ω resistor on the board is used to limit the current to approximately 3.5mA.

'rotary event' enables the shift register to operate.
Notice how the single clock cycle pulse ensures that the LED pattern only moves by one position.
'rotarty_left' is used to determine the direction.



Hint – Signal declarations can be used to define initial values for registers.
These are then loaded into the Spartan-3E during configuration.
Check that LD4 really is the one to illuminate when you first configure the device.

```
signal led_pattern : std_logic_vector(7 downto 0):= "00010000";
```

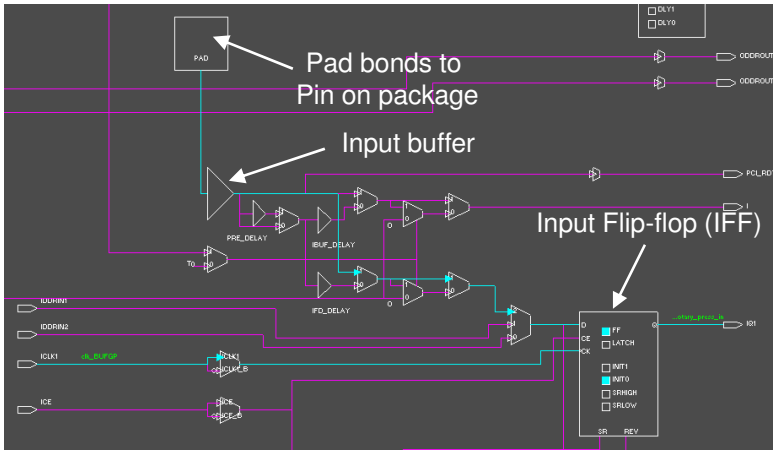
Exercise 1 – Modify the shift register code such that the illuminated LED stops when it reaches the far left (LD7) or far right (LD0) position.

Exercise 2 – The 'rotary_press' input is not protected against switch chatter. Design and implement a suitable filter circuit to clean this signal.
1) Implement a test circuit based on a counter to demonstrate that switch chatter is present.
2) Design and implement your anti-chatter circuit.
3) Use your test design to demonstrate that the operation of the switch is now reliable.

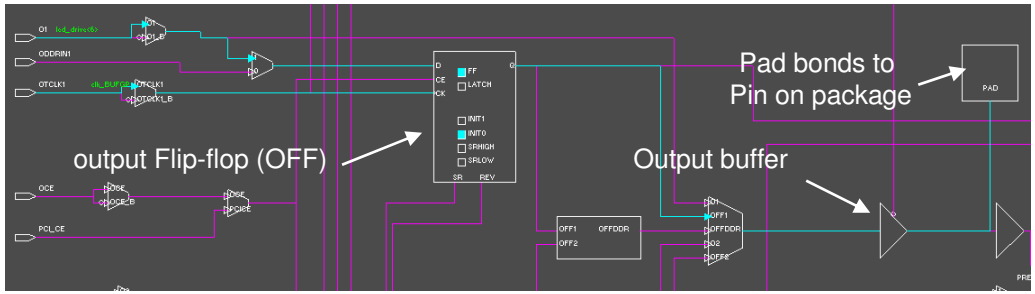
IOB Flip-Flops

Although not vital for this very low performance design, the VHDL does includes additional flip-flops which are mapped automatically into the input/output blocks (IOB) of the Spartan-3E device. If you use the FPGA Editor to look inside the IOB blocks, you can see that these flip-flops have been used.

IOB for 'rotary_a' input.



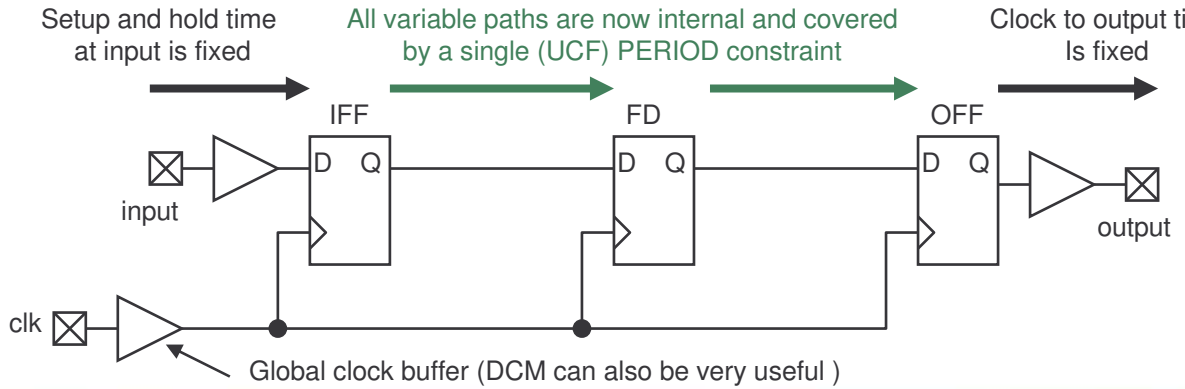
IOB for 'led(6)' output.



How? – All you need to do is include the additional layer of flip-flops connected directly signals defining the pins. The tools will do the rest for you.

```
if clk'event and clk='1' then

   led <= led_drive;

end if;
```

Why bother? Using the IOB flip-flops is good design practice for when designs require higher performance. Using the IOB flip-flops means that the input and output timing is deterministic. In fact, even before you start your design you can know the input and output timing from the Spartan-3E data sheet. Besides, that, why waste them?



Hint – A period constraint describes the duration (or period) of one clock cycle. In this case the 50MHz clock has a period of 20ns. The tools then attempt to keep all connections between flip-flops short enough to allow the propagation of signals through the interconnect and logic to be less than this value.