Quadrature decoder with or without clock input
- or -
How to use a rotary encoder with an FPGA using VHDL

Ahmet Inan
xdsopl@googlemail.com

May 24, 2016

Abstract

It has been a long time since I played with 74HCT devices to build an 16-Bit-ISA card bus interface. But times change and you wouldn't do that today having all these nice tools and devices. I'm talking of course about CPLD (sea of gates) and FPGA (lookup tables) devices and most important of all, their design synthesis tools. So I wanted to know, how rotary encoders work, how I could use them with FPGA's and also learn VHDL. It's gonna be fun, let's go!

1 Contacts of mechanical switches bounce

Before going to the logic operation side of things, let's first study the signals generated by a mechanical switch based rotary encoder. As you can see from the signals captured [1] by a dual channel oscilloscope in figure 1, the contacts of the mechanical switches inside of the rotary encoder will bounce while switching and create erratic pulses until they finally settle down. Fortunately, only one of the two signals created by the rotary encoder will ever have those erratic pulses simultaneously. So could we apply the things we've learned from debouncing a double throw switch with an SR latch[2]? Yes, we can!

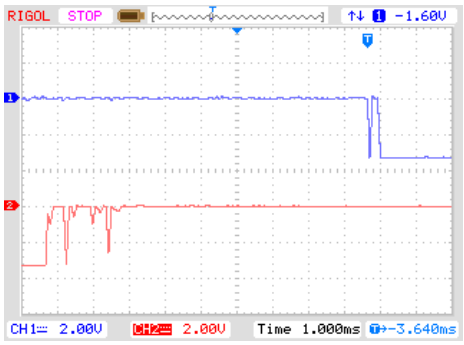

Figure 1: Bounces caused by the mechanical switches of the rotary encoder

[1] Image created by Rigol DS1102E oscilloscope and inverted using ImageMagick *convert*.
[2] Jack G. Ganssle wrote a very nice article about it: A Guide to Debouncing.

2 Quadrature signals

Let's disregard the erratic pulses for a moment so we can focus on the interrelation of the signals shown in figure 2 created by a rotary encoder. On the left you can see waveforms that happen when the wheel of the rotary encoder is turned one step left and one step right is what you see on the right side. These signals are said to be *in quadrature* as their relation is that they are ortogonal to each other or in simpler terms, 90° out of phase. We can also gather, that there are only four signal level constellations we have to consider and creating the signals to be debounced by our trusty SR latches is now easy:

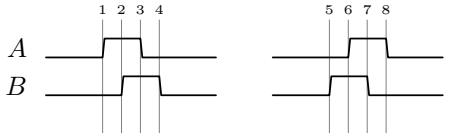

Figure 2: Signals in quadrature

$$1-2, 7-8: \quad A = H \wedge B = L \quad \Longrightarrow E \leftarrow A \wedge \neg B \tag{1}$$
$$2-3, 6-7: \quad A = H \wedge B = H \quad \Longrightarrow C \leftarrow A \wedge B \tag{2}$$
$$3-4, 5-6: \quad A = L \wedge B = H \quad \Longrightarrow F \leftarrow B \wedge \neg A \tag{3}$$
$$else: \quad A = L \wedge B = L \quad \Longrightarrow D \leftarrow \neg(A \vee B) \tag{4}$$

3 Design with logic gates

Putting those new signals $C - F$ to good use and designing a circuit made from logic gates should lead us to something like shown in figure 3. Before immediately jumping to build the circuit in hardware (now that you have the circuit) please be aware that the signals shown in figure 2 are active high signals. With TTL logic you usually use pull-up resistors with mechanical switches and thus have active low signals. The reason being that it needs a much higher current when driving an TTL input low than high. With CMOS logic it doesn't matter: you can exchange pull-up with pull-down resistors for the mechanical switches, therefore inverting the signal with no problems. Using active low signals with this circuit will make it work unreliable and you should redesign the circuit if you intend to go that way (bonus points to those who do). So is it really gonna work if we build it? Let's find out!

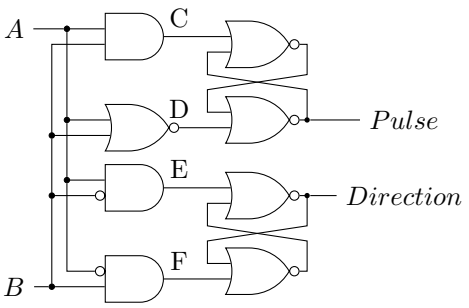

Figure 3: Logic gate circuit

4 Circuit analysis

Carefully analysing the waveforms of the signals $C - F$, *Pulse* and *Direction* created by our circuit when stimulated by the clean signals A and B might look like in figure 4.

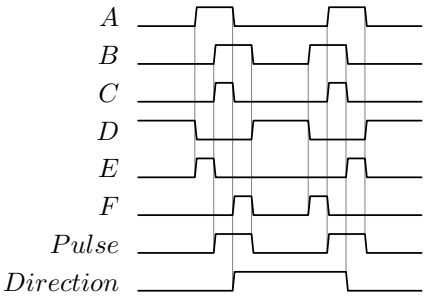

Figure 4: Clean waveforms

```
c <= a and b;
d <= a nor b;
e <= a and not b;
f <= b and not a;
dir <= dir_n nor e;
dir_n <= dir nor f;
pul <= pul_n nor d;
pul_n <= pul nor c;
pulse <= pul;
direction <= dir;
```

Figure 5: VHDL code of asynchronous design

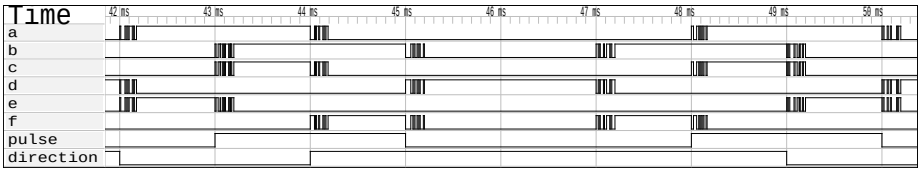

Figure 6: Waveforms of asynchronous design

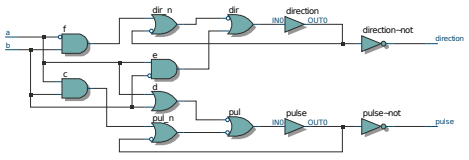

Figure 7: RTL view of asynchronous design

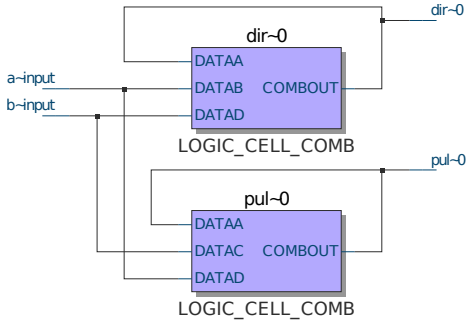

Figure 8: Reduced to two combinational loops

```
Warning: Found combinational loop of 2 nodes
Warning (332126): Node "decoder_inst|pul~0|dataa"
Warning (332126): Node "decoder_inst|pul~0|combout"
Warning: Found combinational loop of 2 nodes
Warning (332126): Node "decoder_inst|dir~0|dataa"
Warning (332126): Node "decoder_inst|dir~0|combout"
```

Figure 9: Combinational loop warning

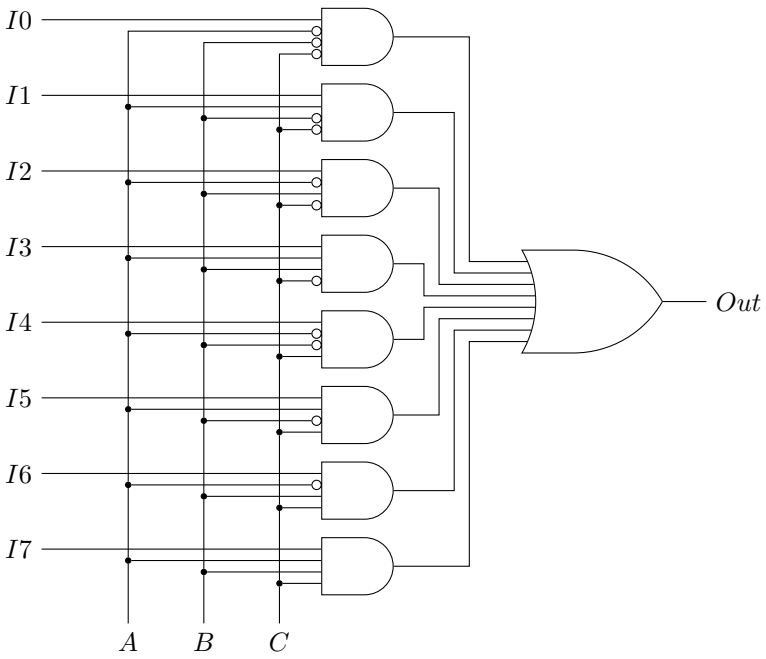

Figure 10: Lookup table logic circuit

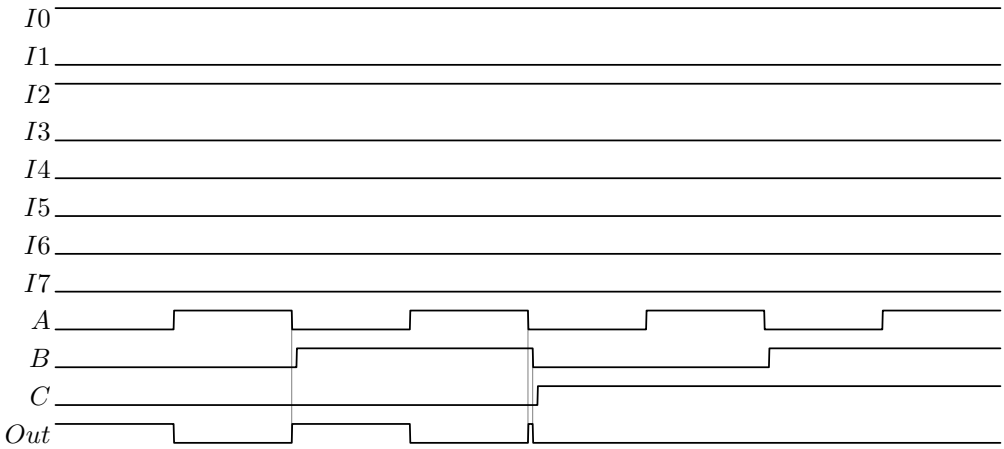

Figure 11: Glitch on output when multiple inputs change at almost the same time

```
c <= a xor b;
dir <= b when rising_edge(clock) and c = '1' else dir;
pul <= a when rising_edge(clock) and c = '0' else pul;
pulse <= pul;
direction <= dir;
```

Figure 12: VHDL code of synchronous design

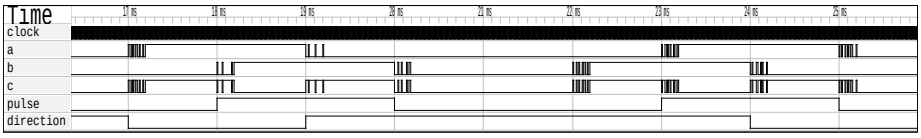

Figure 13: Waveforms of synchronous design

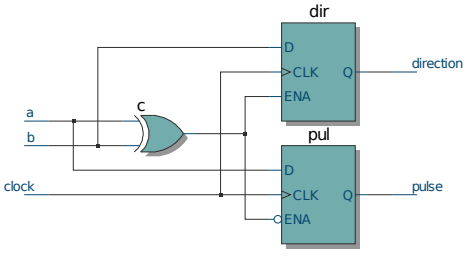

Figure 14: RTL view of synchronous design

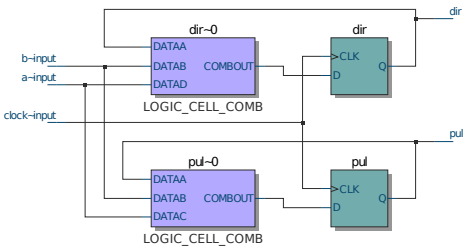

Figure 15: Technology map view of synchronous design