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

Ahmet Inan
xdsopl@googlemail.com

June 5, 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 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 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 at a time. So could we apply the things we've learned from debouncing a double throw switch with an SR latch[2]? Yes, we can!

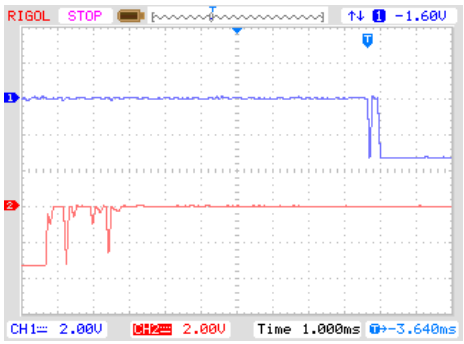

Figure 1: Bounces caused by the mechanical switches of the rotary encoder

[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 created by a rotary encoder, shown in figure 2. On the left you can see waveforms that happen when the wheel of the rotary encoder is turned one step in one direction and one step in the opposite direction is what you see on the right side. These signals are said to be *in quadrature* [3] as their relation is that they are orthogonal 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:

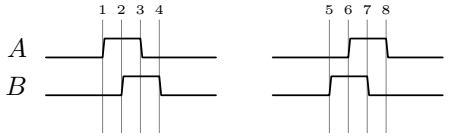

Figure 2: Signals in quadrature

$$1\text{–}2, 7\text{–}8: \quad A = H \wedge B = L \quad \Longrightarrow E \leftarrow A \wedge \neg B \tag{1}$$

$$2\text{–}3, 6\text{–}7: \quad A = H \wedge B = H \quad \Longrightarrow C \leftarrow A \wedge B \tag{2}$$

$$3\text{–}4, 5\text{–}6: \quad A = L \wedge B = H \quad \Longrightarrow F \leftarrow B \wedge \neg A \tag{3}$$

$$else: \quad A = L \wedge B = L \quad \Longrightarrow D \leftarrow \neg(A \vee B) \tag{4}$$

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 [4] 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!

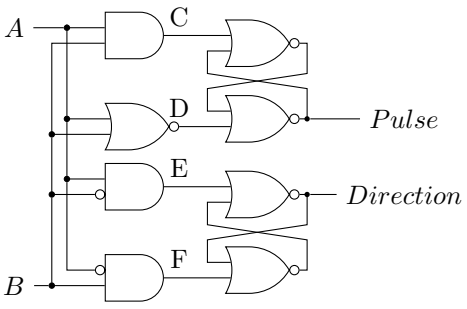

Figure 3: Logic gate circuit

[3] Wikipedia on: In-phase and quadrature components.
[4] Wikipedia on: Pull-up resistor.

4 Circuit analysis

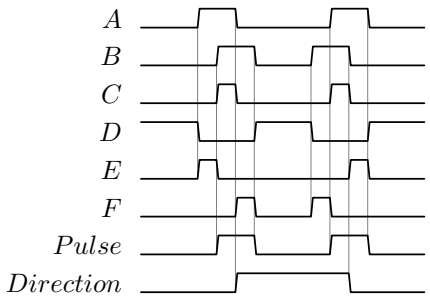

Figure 4: Clean waveforms

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 6. We can see, that the high levels of the signal pairs C, D and E, F do not overlap. This is important for the SR latch to work correctly, as the state when both inputs are high would cause a race condition [5], leading to undefined behavior. Even the signals *Pulse* and *Direction* look good: *Direction* stays steady while *Pulse* is going from low to high. Those are exactly the signals we can use with rising edge-triggered devices. Setup and hold times shouldn't be an issue either, as the electronic devices we are going to use are many orders of magnitude faster than the signals created by our mechanical rotary switches. So let's build it. With VHDL!

5 Design with VHDL

Signals A and B are the inputs we have and need to create the signals for the outputs *Pulse* and *Direction* of our circuit. Keeping in mind that VHDL statements run concurrently, we can easily write our circuit from figure 3 down like in figure 5 But wait a minute, aren't we creating race conditions here already? Nope, and here's why: VHDL uses something called the *delta cycle algorithm*[6] to simulate the circuit. In short: Signal assignments in VHDL get their value updated in the next delta cycle, just like a physical logic gate would deliver its new resulting output value after some time has past when stimulated by new input values. Okay, but why are we using signals A and B directly in our VHDL circuit but *Pulse* and *Direction* not? If you look closer, A and B are used exclusively as inputs whereas the signal pairs dir, dir_n and pul, pul_n are used as inputs and outputs at the same time. With the last two simple signal assignments we can keep the signals *Pulse* and *Direction* strictly as outputs. Can we please see the circuit in action now? Quick, to the testbench!

```
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

[5] Wikibooks on: Digital Circuits/Latches
[6] Jan Decaluwe wrote a very nice article about it: VHDL's crown jewel.

6 Testbench

What I didn't realise at first while working with VHDL was that having a testbench is such an awesome concept. We can actually create a virtual environment where we can test our design in isolation and do all kinds of nasty stimulations we can think of to it, without ever worrying about the magic smoke to come out. Even nicer is the fact, that there are free and open source tools like GHDL [7] for VHDL simulation and GTKWave [8] for waveform visualization. Those two are the only tools you will ever need if you want to learn to code in VHDL. The result of running a testbench simulation on the VHDL code from figure 5 in GHDL and visualized in GTKWave would look like in figure 6. As you can

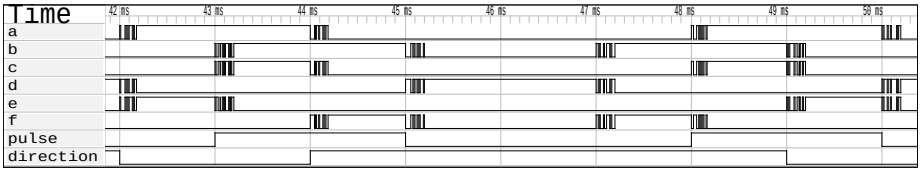

Figure 6: Waveforms of asynchronous design

see, we are even able to simulate the contact bounces in a testbench! Signals *A–F* have the erratic pulses but behold, the *Pulse* and *Direction* signals are properly debounced by the SR latches. If you want to run this testbench on your own, this is the place where you should stop reading and have fun with the VHDL code on my GitHub page: `https://github.com/xdsopl/vhdl`
Had fun? Thought so. Let's continue with the hardware.

7 Which FPGA vendor

For my FPGA experiments I ordered an Altera MAX10 devkit [9] on Digi-Key [10]. That's why I'm going to continue with the results and screenshots [11] made from Altera's free Quartus Prime Lite Edition [12] design software. Don't forget to order an download cable with the above devkit, as it has no onboard programmer. I ended up ordering the cheaper USB Blaster clone from Terasic on Digi-Key [13] which also works with the software from Altera. The only thing I couldn't understand was the lack of galvanic isolation for a simple device of such high price. So be careful when poking around for differential signals with your oscilloscope[14]. With the decision over which hardware to choose cleared out of the way, let's continue and use Quartus to synthesize our VHDL design.

[7] `http://ghdl.free.fr/`
[8] `http://gtkwave.sourceforge.net/`
[9] Altera MAX 10 FPGA Evaluation Kit EK-10M08E144ES/P
[10] Digi-Key Part Number: 544-3042-ND
[11] Used "print to PDF" and edited in Inkscape
[12] `http://dl.altera.com/?edition=lite`
[13] Digi-Key Part Number: P0302-ND
[14] Dave Jones made a very nice video about it: How NOT To Blow Up Your Oscilloscope!

8 Hardware synthesis

The VHDL code from figure 5 compiled in Quartus and viewed in it's RTL Viewer looks like in figure 7. It might look a little odd but it is essentially the same

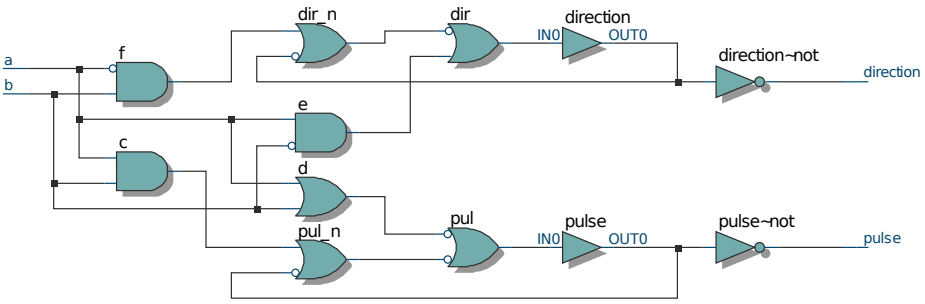

Figure 7: RTL view of our design

circuit as in figure 3 But we are more interested about the configuration that will later land on the FPGA. That's where the Technology Map Viewer comes into play and our design looks like in figure 8 Wow, just two lookup tables! And

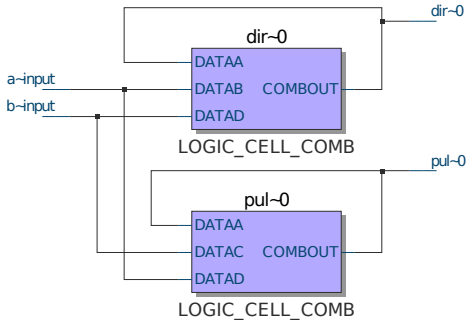

Figure 8: Reduced to two combinational loops

it works, too! It still looks to good to be true and indeed this configuration harbours hidden dangers as we can see some hints from Quartus shown in figure 9. The problem lies within the feedback of $COMBOUT$ to $DATAA$. But it works

```
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

fine, I tried it, so why the fuss? Let's look at how lookup tables work.

9 Lookup tables

There are actually a lot of different implementation designs for lookup tables solving one or another problem depending on their use case and constraints. A possible implementation using logic gates is shown in figure 10. As you can see, changing the value of a single signal like for example A would stimulate almost every single logic gate in parallel and therefore a lot of transistors. Not every transistor is born equal on the same die: some of them will switch faster whereas others will be weaker, etc. So we have the problem, that not every signal level change on the inputs A–C will result in the simultaneous switching of all transistors and thus we have overlapping transitions on the outputs of the logic gates, causing glitches. Some FPGA vendors guarantee [15] that their LUT implementation won't glitch on a value change of a single input or if all value permutations of the signals involved by the change would result in the same output. Even if we would have an implementation that would guarantee a glitch free operation on multiple signals changing at once, we would still experience glitches, as it can't be guaranteed that all input signals will change their value simultaneously. In figure 11 we see the waveforms and a glitch happening to a guaranteed glitch free operating lookup table with the constant value signals $I0$–$I7$ and beeing stimulated by the signals A–C. Looking closer at the

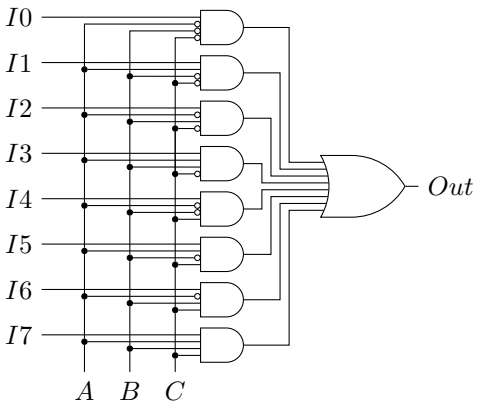

Figure 10: Lookup table logic circuit

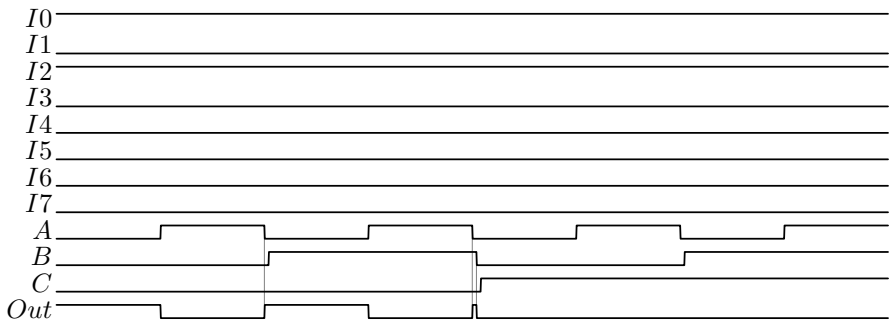

Figure 11: Glitch on Out when multiple inputs change at almost the same time

edges of the signals A–C you can see, that they do not arrive at the same time: A comes before B and B comes before C. That little delay causes a glitch that could easily ruin your day. Depending on the load of the outputs and routing of the signals, such unequal delays are hard to avoid. Fortunately there are practical ways to deal with it: Enter the world of synchronous designs.

[15] You have to wade through forum posts to get something from officials :(

10 Register-transfer level

Fixing the glitch is so easy, that it is almost boring: use an externally clocked register in the feedback loop! Even the VHDL code get's simpler as you can see in figure 12. We sample A or B on the rising edge of our external $Clock$ signal,

```
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

which should be clocked slower than the output stabilization time of the lookup tables inside the FPGA but at least two times faster [16] than the shortest stable [17] pulse seen on our signal C. Putting the synchronous design in a testbench gives us the waveforms shown in figure 13. Properly debounced signals $Pulse$ and

Figure 13: Waveforms of synchronous design

$Direction$ and exactly with the timing needed for edge triggered devices! For completeness sake, let's also look at the quite boring RTL view in figure 14. Just one XOR gate and two D-type flip-flops with clock enable input. Let's

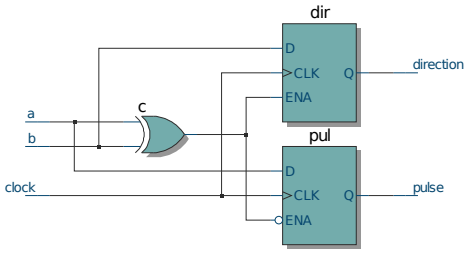

Figure 14: RTL view of synchronous design

finish up here quickly by looking at the technology map view in figure 15.

[16] See Nyquist limit.
[17] Not accounting for the contact bounce pulses.

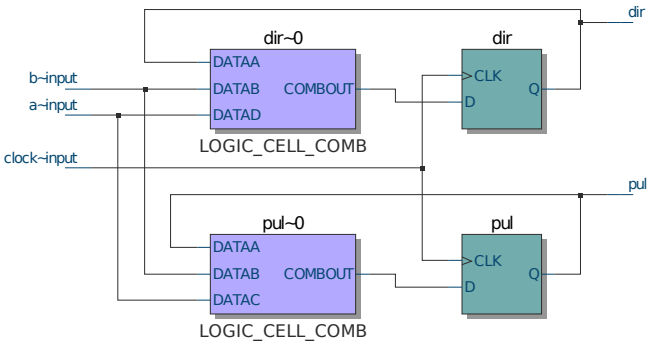

Figure 15: Technology map view of synchronous design