4x4 matrix keypad interface

Embedded systems - Journal 1

Kenneth Rungstrøm Larsen kenla16@student.sdu.dk.

1 Introduction

This project revolves about the design and implementation of an interface for a 4x4 matrix keypad and a 7-segment display, combined via the Pynq-7020 FPGA development board. The interface has to accept inputs from the keypad and show the corresponding value on the display.

2 Design

Throughout the project the method of writing concurrent code is used to implement hardware components, also known as hardware descriptions, that executes concurrently instead of sequentiallty.
Each component is designed to perform one task only, thereby complying with the general rule of partitioning hardware descriptions into reusable sub-components.
The interface is comprised of 5 main components and 5 auxiliary that interfaces between the primary components. For simulation there is an additional component that mimics the actual keypad. Besides this there is a port-map which is a necessary wrapper that routes signals between components.

2.1 Overview

Figure 1 is a graphical view of the port-map with all components needed for hardware implementation. The keypad mimicking components is left out, since it doesn't belong in the hardware implementation, but connects between output [y_0] and input [in1_0] when simulating. The display_0 component in the upper right hand corner has also been left out, since it is functionally equivalent to the look-up table component. In my opinion the look-up table is more important, since it outputs the correct binary value representation of the button number, where the display_0 component just activates a sequence of output pins the display is connected to. This pin configuration will change if just one of the connections to the display is changed.

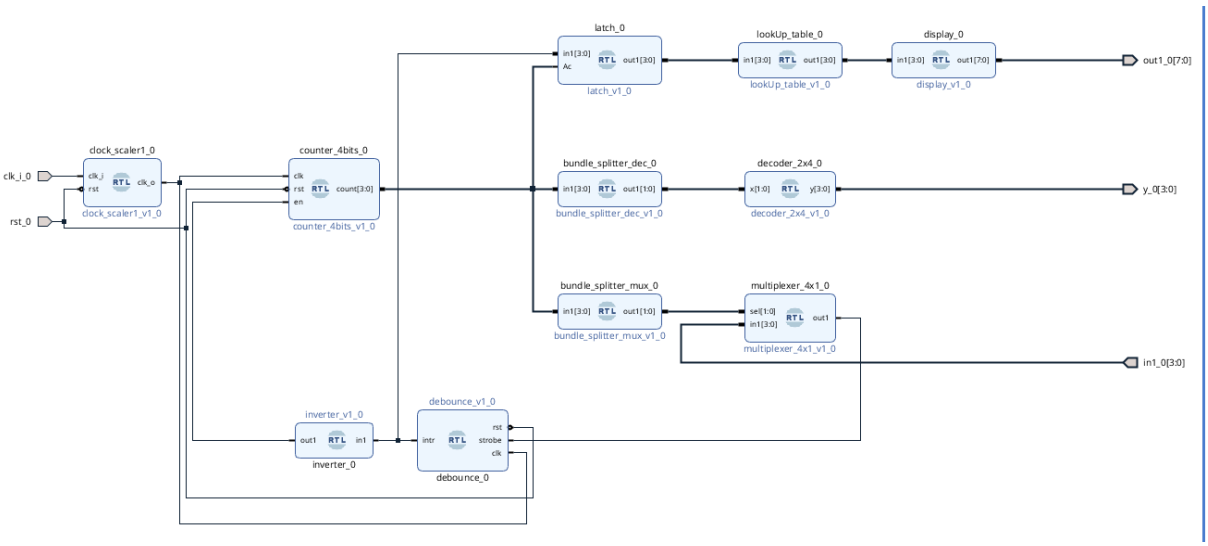

Figure 1: Component overview

2.1.1 Counter

The counter components task is to count from 0 to 15, in binary. Whenever it reaches 15 its internal register overflows and continues from 0. With each clock-pulse the counter increments the value in its register which creates a continiously loop. The counter receives 3 binary inputs signals enable [en], reset [rst] and clock [clk] and produces one bundled output of 4 bits.
When rst=1 the counter stops reacting to the clk signal and resets the internal register to 0. If rst=0 and en=0 the counter still doesn't react to the clk, but as soon as en=1 the counter starts counting.
The output from the counter is routed to a look-up table, a decoder and a multiplexer, through auxiliary components.

2.1.2 Decoder

The decoder component accepts a 2 bit bundled input and outputs a 4 bit bundled output. The rules are that the output can only take on 4 out of the 15 values that a 4 bit number can usually represent. This is because the 4 bits are actually tied to 4 physical outputs pins that needs to be activated one at a time. The decoders task is to decode the 2 bit input to one of the 4 valid outputs, ie. '00' => '0001', '01' => '0010', '10' => '0100' and '11' => '1000'. Since the input is taken from the counters output, the decoder will sequentially shuffle through its output values as the input keeps changing with every clock pulse.
The output from the decoder is tied to the 4 inputs of the keypad.

2.1.3 Multiplexer

The multiplexer component accepts a 4 bit bundled input, a 2 bit bundled 'select [sel]' input and has a 1 bit output. The input is connected to 4 physical pins, each connected to the output pins of the keypad. A value on the sel input chooses which input to connect to the multiplexers output. If sel='00' the output = input 1, if sel='01' the output = input 2, etc. The sel input is taken from the counters output which means that the multiplexer will sequentially shift, with each clock pulse, between what input is connected to its output. If at some point an input has a value of '1', the output will be set to '1' too.
The output is connected to the debouncer component

2.1.4 Bundle splitters

This section represents 2 of the auxiliray components. These components are called bundle splitters and their job is to split the 4 bit output from the counter into 2 bit. The first bundle splitter is for the decoders input. It extract the 2 most important bits [MSBs] of the counters output, ie. 'xxyy' => bundle splitter dec => 'xx'. The second bundle splitter is for the multiplexers sel input. It extracts the 2 least important bits [LSBs] of the counters output, ie. 'xxyy' => bundle splitter mux => 'yy'. The reason for doing this is that the 2 LSBs will change value 4 times faster than the 2 MSBs. This in turn means that when the decoder sets an output high, a row in the matrix keypad will be activated and if a key is pressed, the corresponding colum will have value of '1'. The multiplexer will read all 4 colums per activated row, determining whether a key has been pressed.

2.1.5 Inverter

The 3rd auxiliary component is the inverter. It takes the output from the debouncer, inverts the value, and feeds that to the en input of the counter. This means that when the debouncers output is '1' the counters en input is '0' and is therefore disabled.

2.1.6 look-up table

Since we now have a way of stopping the counter, without resetting it, we can determine which key was pressed by looking at the counters output. The value will represent which of the 16 keys is pressed, but it will not necessarily correspond to the value written on the button. To write the correct value to the display a translation between the counters output value and the correct value is needed. This translation is done with a look-up table. The look-up table accepts a 4 bit bundled input and translates to a 4 bit bundled output. The translated values are listed in the table below.

In	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011
Out	0001	0010	0011	1010	0100	0101	0110	1011	0111	1000	1001	1100
In	1100	1101	1110	1111								
Out	1110	0000	1111	1101								

2.1.7 clock-scaler

The 4th auxiliary component is a clock scaler. This component takes in the external 125MHz clock, present on the FPGA's H16 pin and scales it by 2^{17}, yielding about 950Hz. This is calculated by formula $n = log_2(f_{clk}/f)$, where n is the power and f is the desired frequency. The component is basically a 17 bit counter that increments its value with each external clock pulse until its value matches with a fixed "compare" value. When the compare value is reached the counter is reset and the counting starts over again. By doing it like this the output from the clock-scaler is symmetrical, ie 50% duty-cycle, and can therefore be used as the clock source for the counter introduced earlier.

2.1.8 Debouncer

Since a physical button bounces, switches on and off rapidly, within the first couple of μS a debouncer component is needed. This component takes in a noisy signal and waits until the signal i stable before it produces an output. The component is placed between the multiplexers output and the inverters input. The component is made as a 3 state, finite state machine with an idle state [S0], a waiting state [S1] and reaction state [S2]. In S0 the FSM wait for the first pulse from the multiplexer. When the transition condition is met the FSM changes state to S1 where an 8 bit counter is activated. The counter acts as a time delay and when MSB changes to '1', the FSM changes state to S2. In S2 the FSM creates a single pulse by briefly setting its output to '1' and then back to '0' again.
This briefly stops the counter and simultaniously signals a latch.

2.1.9 Latch

The 5th auxiliary component is the latch. This component has a 4 bit bundled input, a 4 bit bundled output and a 1 bit activation signal. The activation signal is tied to the debouncers output and when it changes value to '1', the latch latches the value in the counter to the look-up table. By using a latch the look-up table will get a consistent value while the counter continues to count, resulting in an interface that is always ready to accept new button presses.

3 Simulation

Generally when creating hardware designs with VHDL and FPGAs it is recomended to simulate the design before attempting to program the actual FPGA, since the procedure for programming can be relatively slow. When simulating we verify that for a known input the system generates the expected output. A successful simulation is a good indication that the system will behave appropriately when implemented in hardware. For larger designs or tightly timeing constrained designs it is also advisable to make a simulation of the system timings, where a generalized propagation time, for each used logic gate, is factored in. This kind of simulation verifies that the systems timings are obayed given real world signal delay.
To verify that my design is working properly I included the keypad mimicking component, instructed it to output each of the 16 values, one at a time, and recorded the output from the look-up table. When simulating button 0, the one with label 1, I expected the value '0001' in the lookup table. Likewise when simulation button 1, the one with label 2, I expected the value '0010' in the look-up table, etc.
Image 2 shows my simulation attempt with the button 0 pressed.

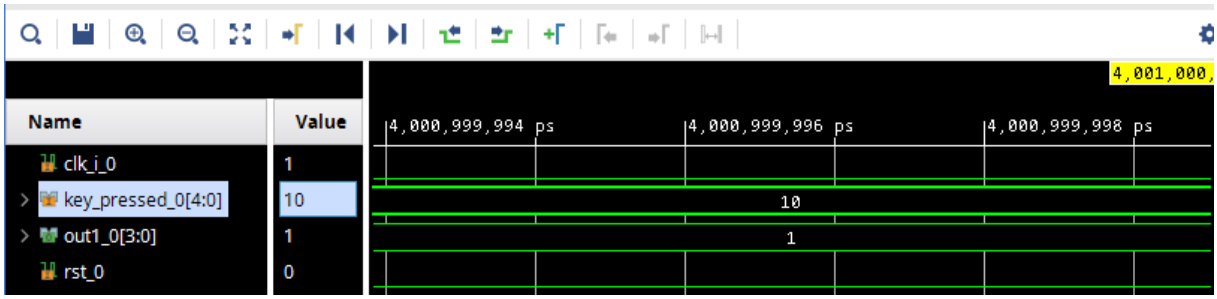

Figure 2: Simulation, output is 1 when input is 0 - note keypad-sim needs '10000' to generate output '0000'

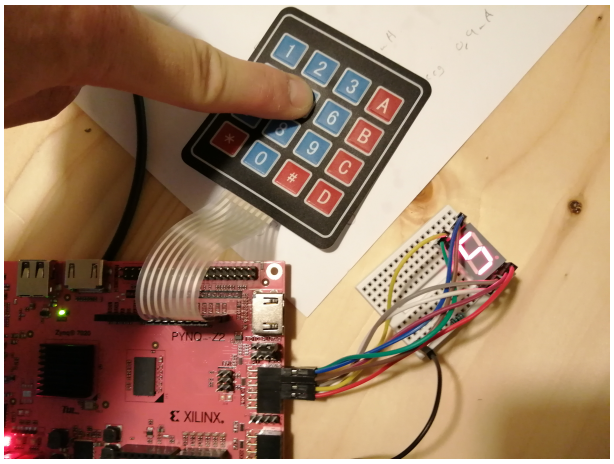

Figure 3: Display output when button '5' is pressed

4 Implementation

Implementation of the design involves generation of the bit-stream that when downloaded to the FPGA instructs it to make the necessary connections between its configurable logic blocks and a system of programmable interconnections called the fabric. Before generating the bit-stream it is important to assign the input/output signals of the design, ie. *clk_1_0, rst_0, out1_0, y_0 and in1_0* from figure 1, to physical input/output pins of the FGPA. The clk_1_0 clock signal input connects pin H16 of the FPGA, where an onboard 125MHz clock generator, normally intented for the boards ethernet module, is present. The rst_0 reset input connects to pin D19, where an external toggle button is connected. The 4 signal bundles in1_0 and y_0 connects to the boards' AR0-7 GPIO header, with pin assignment according to the datasheet. The 7-segment display basically consists of 8 parallel coupled LEDs with all cathodes connected to a common ground. The common ground is connected to one of the boards ground pins through a 220Ωresistor. This resistor serves as current limiting to protect the FPGAs output pins.
The 3.3V supply voltage to the display and the 1.8V forward voltage of the LEDs gives a 1.5V drop accross the resistor yielding current I of $(3.3 - 1.8)/220 = 7mA$. When only 1 segment of the display is lit, that LED forwards the entire 7mA. When all 8 LEDs are lit, each LED forwards 0.9mA. The higher current is well within the maximum 10mA current per LED where as the lower current is 0.1mA too little, according to the datasheet. Theoretically too little current dims the LEDs, but in practice this is not noticeable. The anodes of the LEDs connect to GPIO header A1-4 and AR10-13 with pin assignment according to the datasheet.
Testing the implementation is carried out by pushing each button, one at a time, and recording the display output. The implementation is deemed successful when button labels and display output matches. Figure 3 demonstrates the output when button '5' is pressed.

5 Conclusion

It is important to realize that the 125MHz clock frequency is just too fast and all sorts of random behavior occures, from ignoring keypad inputs to writing other than the expected value to the display. Another thing to notice is the keypads sensitivity to capacitive interference. Just being near the keypad, without even touching it, results in an output that the FPGA can read. Even with debounce this behaviour can be hard to mitigate.
During troubleshooting I realized that I had made the same error in both the multiplexer, decoder, latch and look-up table. I used the "when others" command as the last statement in every "with select" statement, and set it to 'Z', the high impedance state. This resulted in a completely unusable system as the output from all of these were almost always high impedance. On the other hand the system works and is even quite robust as it seems to react to every button input.