Lab Assignment 4
Hardware Implementation of a Pushbutton-Controlled Counter

Zhenming Yang, Connor McEleney
yang.zhenm@northeastern.edu, mceleney.c@northeastern.edu
Instructor: Prof. Zagieboylo
d.zagieboylo@northeastern.edu

Due Date: October 12th, 2023
Submit Date: October 12th, 2023

Abstract

In this lab, we showcased reading the status of hardware-based pushbuttons and mitigating signal bouncing to control LEDs and display Counters and control its properties.

Introduction

In this lab session, we demonstrated the process of reading the status of hardware-based pushbuttons and addressing the issue of signal bouncing that can arise from these input devices. Our initial focus will be on a simple setup where a pushbutton control the state of an LED. As we progress, we will expand this design to incorporate pushbuttons and switches for purposes such as initiating or stopping a counter, loading counter values, and modifying the counting direction and speed.

Lab Setup

Pre-Lab

The value calculated for LMP_Compare was found to be 12,500,000 because that is how many ticks of the clock is equal to 250ms. (Found by taking a fourth of 50,000,000 because that is the spec of the clock, as in that many cycles per second, and 250ms is a fourth of a second)

Equipment

DE1-SoC:

- The DE1-SoC is a hardware design platform built around the Altera System-on-Chip (SoC) FPGA. The DE1-SoC is designed for experiments on computer organization and embedded systems. It includes embedded processors, memory, audio and video devices, and some simple I/O peripherals.

Results and Analysis

Results

Part 1: Turning an LED ON/OFF with a Pushbutton

Our initial objective involves setting up the DE1-SoC to manage the status of an LED by using the KEY0 pushbutton. Whenever the button is pressed, the LED switches its state - either turning on or off. This functionality can be achieved by employing a basic 1-bit counter, which is directly connected to the LED. A 1-bit counter can only have values of 0 or 1, and when it reaches its maximum value of 1, it overflows and returns to 0 [1].

To ensure the counter is activated when the button is pressed, we need to enable its input whenever the button input signal experiences a transition from 0 to 1. Implementing this behavior can be challenging due to the inherent physical imperfections in pushbutton contacts, which typically lead to signal bouncing. As depicted in the figure provided, each time the button is pressed, its output signal rapidly shifts between 0 and 1 before settling at a stable value of 1.

To address the issue of pushbutton signal bouncing, we created a cascaded counter. This counter design ensures that the signal originating from a pushbutton remains stable at a value of 1 for a sufficient duration before we can reliably confirm that the button has been effectively pressed.

Based off the design specified on the lab instructions, we **created our schematic for controlling LED with a Pushbutton** (Figure 1) and name it “**debounce_p**”. We got the constant that the first counter compared to be **125000** to achieve the objective of only triggering the LED when the button is pressed for 250ms. The schematic was tested on the board and saved for future usage.

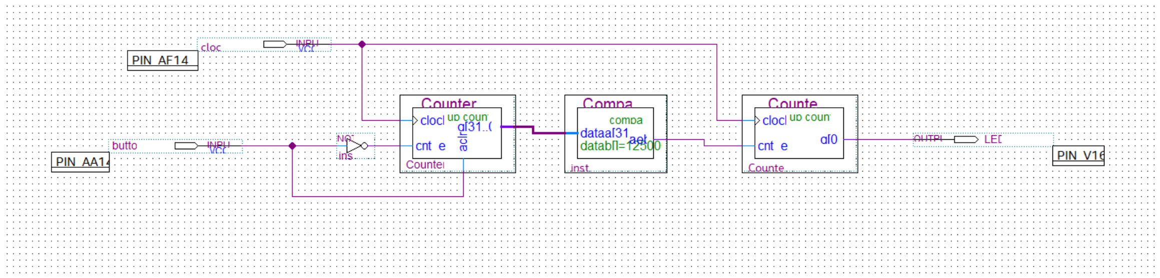

Figure 1: The Schematics to turn on the LED when the button is pressed for 250 milliseconds. Noted the Compare module compares the input data stream with 125000 to achieve this functionality.

We also generate a fresh project featuring a debounce circuit that generates a solitary pulse, as opposed to a toggle. We named this debounce circuit "**debounce_p**" and encapsulate it as a logic block for future utilization.

Part 2: Designing an 8-bit Cascade Counter

Our next objective is to design a configuration featuring an 8-bit counter that operates continuously. The counter's output will be linked to the 7-Segment displays on the DE1-SoC board, ensuring that each time the counter advances to a new state, the 7-Segment displays will update to reflect the current count value.

To create the Cascade Counter, we first created a 50M counter to slow down the clock frequency of 50MHz on the board. We then connected the output of this counter to a compared to trigger the second counter every second to count to 255. The output of the second counter is connected to the display-7 module we created in the previous labs to display the number of the counter (Figure 2). We then tested the module on the board to make sure that the counter counts every second and reset when the number hits 255.

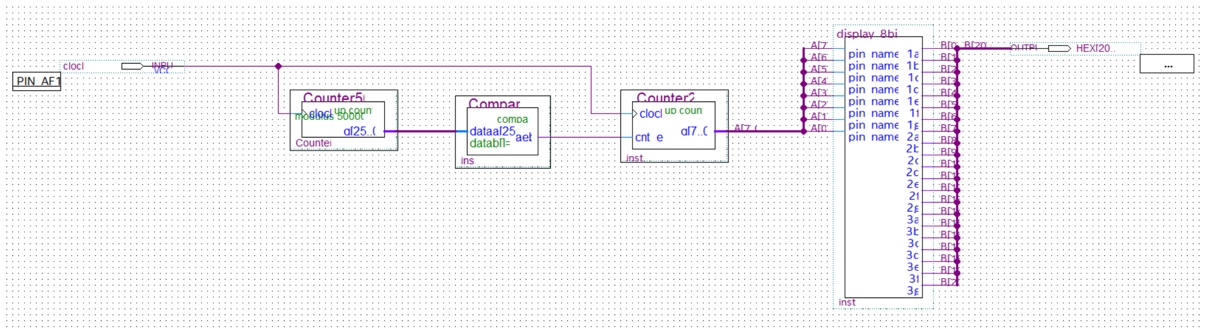

Figure 2: The schematics of the cascade counter. Noted the clock is fed to the first counter and "slow down" the tick to trigger the second counter for the display to show numbers.

Part 2: Pushbutton-Controlled Counter

Based on our previous research, we added a new start and stop push button that controls the counter as we need. The **debounce_t module was used to achieve the function of having the button only triggers the stop/start counter when pressed more than 250ms**. We designed the schematics and tested them on the board (Figure 3). The design was tested on the board to ensure the functionalities work as intended.

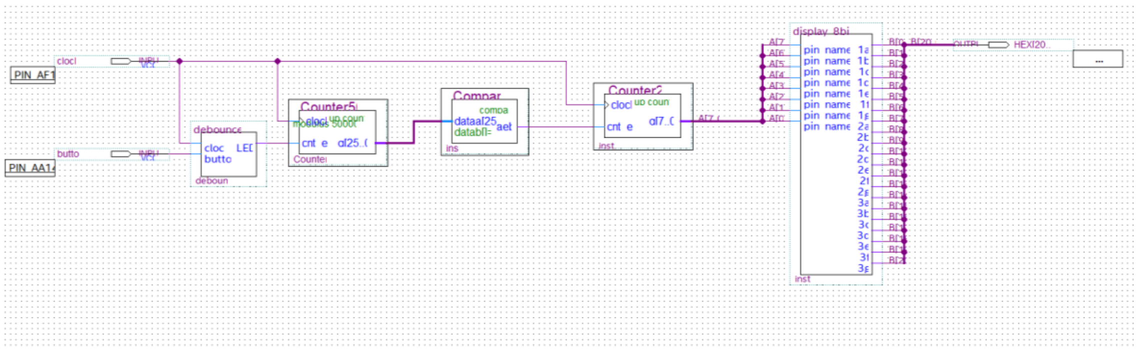

Figure 3: The new button is added to control the counter to start/stop. Noted a debounce_t module was used to allow the pushbutton to work as intended.

We then added a switch to control the direction of the counter. To achieve this functionality, we **added a up/down input for the second counter and added a SW input to control it** (Figure 4). Since the we want the counter to go up first when initialized, **we "Not" the output on the switch** to make sure it starts with a "1" instead of "0." The design was tested again to ensure that it works as intended.

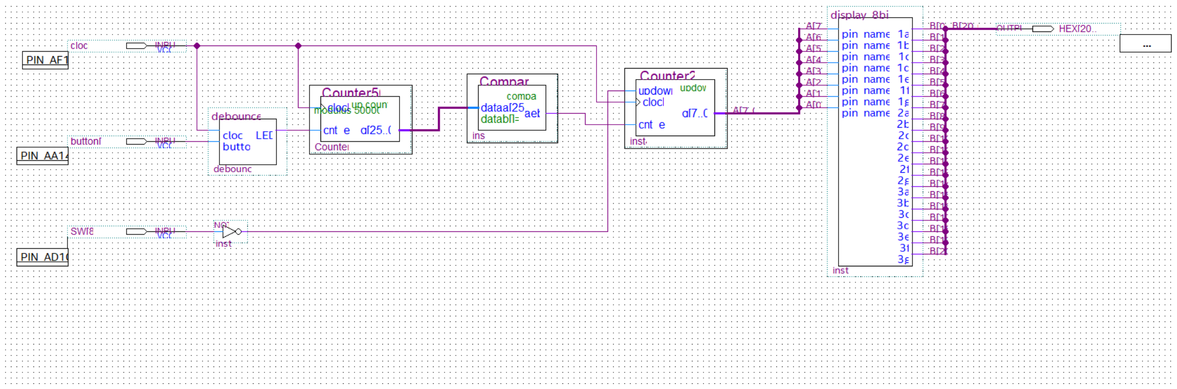

Figure 4: The new button is added to control the direction of the counter. Noted a "Not" gate was used to control the up/down input of the second counter.

To achieve the functionality of loading numbers based on the switches. We needed to first add a new data input and a new async load input into the second module. The **8 switches input is fed to the data input for the counter** to specify the load data and we just need to implement a new push button to control the load. We used the **debounce_p module with the**

pushbutton input to achieve async load because we just to the send a pulse to the counter instead of toggle on the load input (Figure 5).

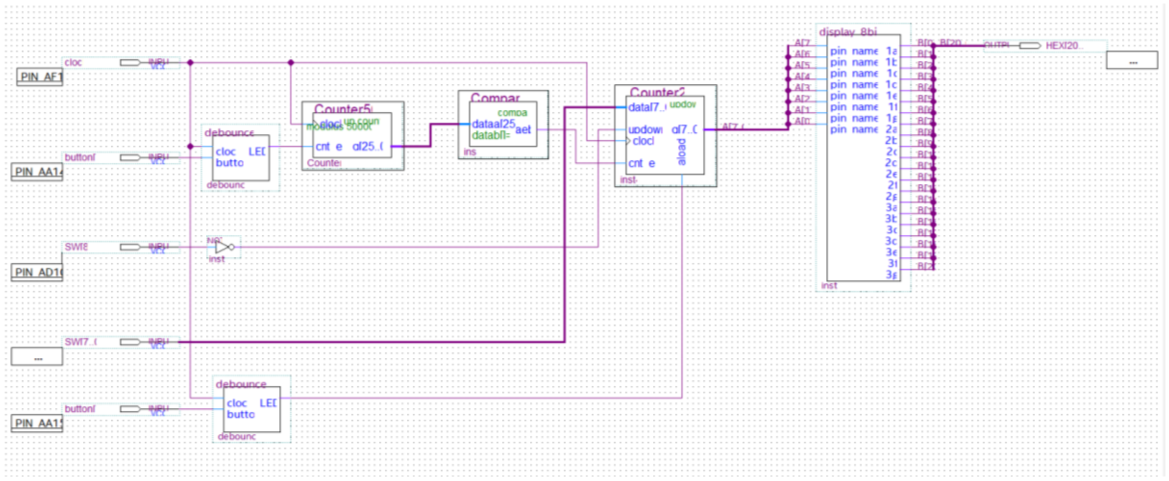

Figure 5: The schematics of the new counter with load number functionality. Noted the debounce_p module was used in this case to make sure the aload input for the second counter is not toggled on when pressed.

Lastly, we implemented the control speed switch to double the speed of the counter when it is on. To achieve this, we **made a new compare module and set it to 25M to async clear the first counter when the output hits 25M**. This way **we half the time between the first counter** output 1 (to compare with our original compare1) **and double the speed for the second counter**. A switch input is used to control this functionality and we used **an "AND" gate to only async clear the first counter if both the switch and the 25M compare is 1** (Figure 6). We tested our design and recorded a video to show all the functionalities.

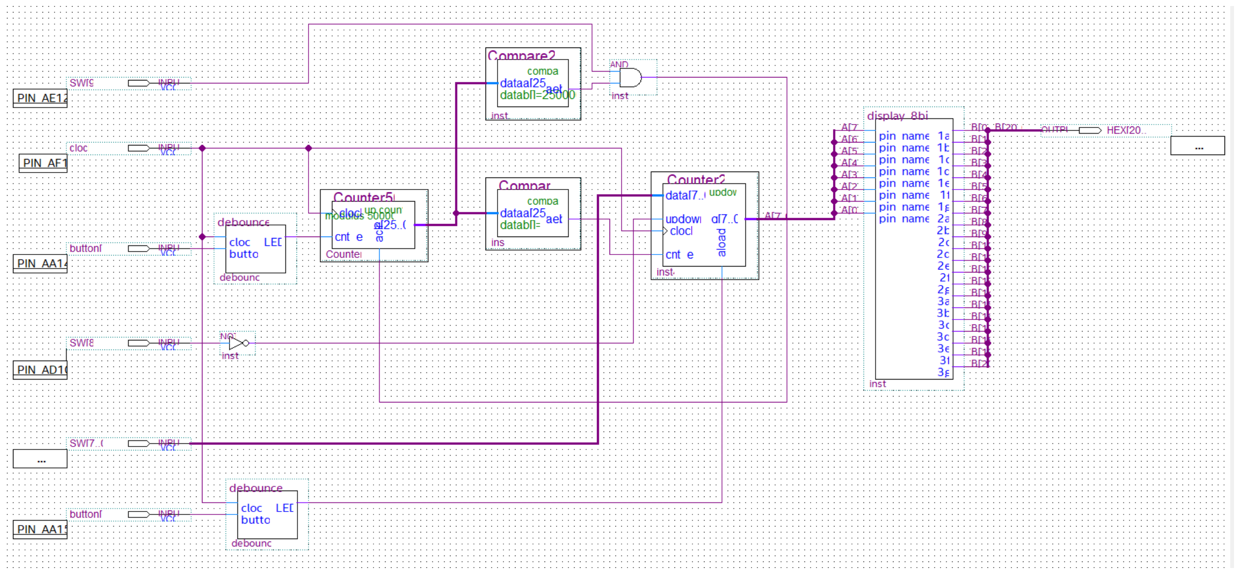
Figure 6: The schematic of the final design for the controlled counter that can stop, reverse, load, and speed up using pushbuttons and switches. Noted the new compare25M module and "AND" gate makes the first counter hits 1 twice as often.

Analysis

For the last part of the lab, we really pushed for a better design that not only achieves the objectives but also reduces the complexity of the circuit if we theoretically needed to manufacture it. Instead of using another counter with less max cap to speed up the counter, we used the same 50M counter but instead used the async clear functionality to clear it every time it hits 25M, effectively doubling the speed the 50M counter output 1. This design philosophy is useful in real life when the time and resources are limited, and we should look for more optimization in the future labs as well.

Conclusion

In this lab we explored new input methods and control modules using the pushbuttons on the DE1-SoC. We were introduced to the concept of noise in the sense the pushbutton would "bounce" a little when pressed, causing unwanted signal propagation after the intended push. To solve this problem, we used the boards 50MHz clock for the first time in conjunction with counting and comparing modules to produce a circuit that would only detect a press after 250ms. Two versions were created, one which would stay activated until switched and one that would output a pulse. Overall, this lab introduced and integrated counters, comparing the syncing of the circuitry to the clock, state memory, and noise. Reflecting on the synced updating of the circuitry and states it can be inferred that a scaled-up circuit using this concept could be used to construct systems like registers for memory.

References

[1] DE1-SoC User Manual