

Lab 9

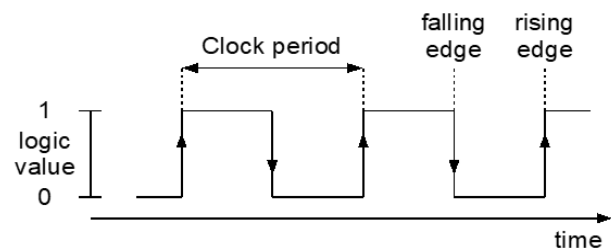
Hardware Implementation of a Pushbuttons-Controlled Counter

1. Introduction

In this lab we will show how to read the state of the pushbuttons by hardware, and how to deal with the signal bouncing effects introduced by these input devices. We will begin with a basic design where the state of an LED is controlled by a pushbutton. This design will be progressively extended to the point where we will utilize the pushbuttons and switches to start/stop a counter, load the counter value, and change its counting direction and speed.

1.1 Pre-Lab Assignment

A clock signal is a periodic signal, constantly transitioning between logic values 0 and 1 with symmetric pulses, as shown in the timing diagram. The following properties of a clock signal can be highlighted:

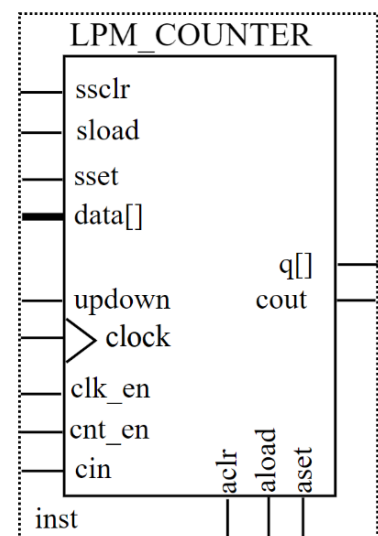


- A falling edge of the clock is a transition from a value of 1 to a value of 0.
- A rising edge is a transition from a value of 0 to a value of 1.
- The clock period is defined as the time between one rising edge and the next (or one falling edge and the next). The clock period of the DE1-SoC's FPGA is $0.02\mu\text{s}$ or 20ns.
- The clock frequency is defined as the reciprocal of the clock period. Therefore, the clock frequency of the DE1-SoC's FPGA is 50MHz.

Clock signals are typically connected to all storage components (i.e., the flip flops) of the sequential circuits in a digital design. Their purpose is synchronizing the transition of these components into their next state. This transition will typically happen at the falling edge of each clock cycle.

An n -bit counter is a logic component storing an n -bit binary number that is incremented on every falling edge of a clock signal. A *free-running* (plain binary) counter is a counter that takes all possible n -bit values incrementally (from 0 to 2^{n-1}) and is reset to 0 when it overflows (after reaching its max value). Its counterpart is a *limited-range* (Modulus) counter, which resets at a configured value (not the default max value, which is 2^{n-1}).

- The LPM_COUNTER IP core is a binary counter that can be used to generate up and down counters with outputs of up to 256 bits wide. The figure shows the ports for the LPM_COUNTER IP core. The counter can be configured following the same steps used to configure the other IP Catalog library components we used in the previous lab.



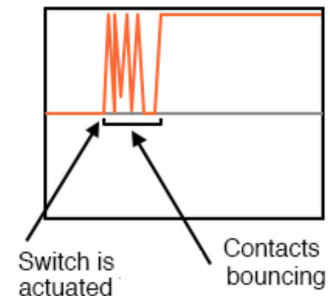
Pre-Lab Reading Assignments

1. Read sections 2.5 *Ports* and 2.6 *Parameters* in the *Intel FPGA Integer Arithmetic IP Cores User Guide* available in the *Lab Resources* section on Canvas. These sections explain the functions of the counter ports and how to configure them.
2. The following reading assignment will help you understand how pushbuttons work:
http://www.allaboutcircuits.com/vol_4/chpt_4/4.html
3. Read section 2 below and calculate the value C needed for the LPM_COMPARE shown in the circuit block diagram figure.

2. Turning an LED ON/OFF with a Pushbutton

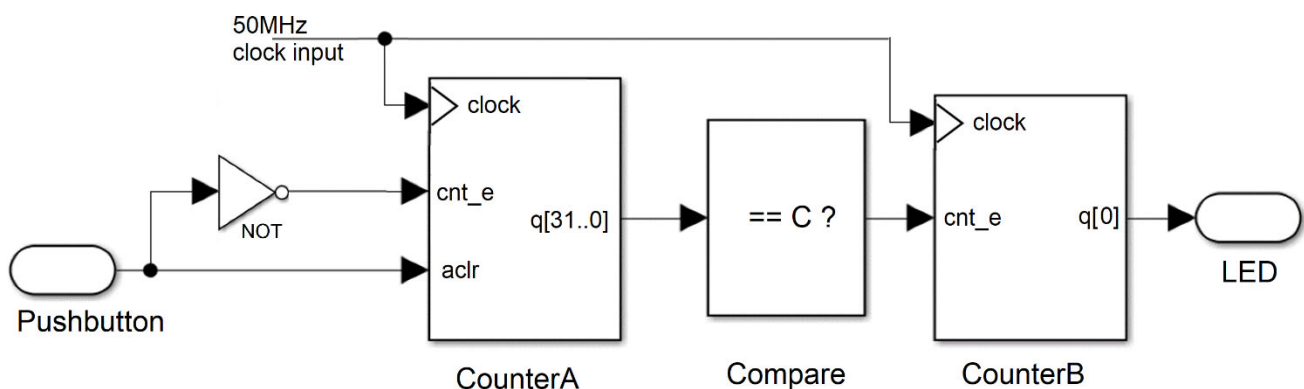
Our first goal is configuring the DE1-SoC to control the state of an LED using the KEY0 pushbutton. Every time this button is pushed, the LED toggles its state, from on to off, or from off to on. This behavior can be achieved with a simple 1-bit counter whose output is connected directly to the LED. A 1-bit counter is one that can take only values 0 and 1. When it reaches its maximum value 1, the counter overflows and goes back to 0.

The *enable* input of the counter should be activated whenever the button is pushed, that is, whenever a transition from 0 to 1 is found in the button input signal. This behavior can be tricky to implement, since the physical imperfection of the contacts within a pushbutton typically causes signal bouncing, as illustrated in the shown figure. Every time a button is pushed, its output signal will quickly transition between 0 and 1 before it takes a final stable value of 1.




To solve the pushbutton debounce problem, we will design a cascaded counter in order to guarantee that a signal coming from a pushbutton has settled with a value of 1 for a period long enough before we consider the button to have been effectively pressed.

1. Create a cascaded counter design as shown below and save it as *debounce.t.bdf* on Quartus schematics.



2. Add an LPM_COUNTER and name it Counter32. Configure it to have 32 bit-wide output, *plain binary*, *Up only* counter, with a *Count Enable* input, and an Asynchronous *Clear input*. As usual, check everything on the *Summary* window.

Hint: you can type “counter” in the IP Catalog search field to find the LPM components.

3. Through the Symbol Tool, add an instant of the Counter32 to your schematic and name it **CounterA**. Connect the *Count Enable* (*cnt_e*) input to an *Input port*, which will later be mapped to the KEY[0] pushbutton pin through a NOT gate. This makes the first counter increment its value for as long as the button is pushed. Note that the pushbuttons are **active low** (i.e., their output is low when pushed).
4. Connect the *aclr* input (asynchronous clear) of the counter to the same KEY[0] input port. This makes the counter go back to 0 when the pushbutton is released (or its output bounces back).
5. Configure an LPM_COMPARE in such a way that it yields 1 only after the button signal has been stable at the pushed status for **250 milliseconds** straight. Recall that a stable button signal on the pushed status keeps **CounterA** counting up without being cleared. As the DE1-SoC board is running at a frequency of 50MHz (50,000,000 clock cycles per second), calculate the value for the LPM_COMPARE's constant *C* in the above figure. Set the LPM_COMPARE to have 32 bits input, equality checker ($a == b$), with *data_b* set to *C* in *decimal* format. As usual, check everything on the *Summary* window.
6. Configure another LPM_COUNTER, and name it Counter1, as a 1-bit *Up only*, *plain binary* counter with a *Count Enable* input. Insert an instant of this counter in your schematic and name it **CounterB**. Connect the output of **CounterB** to an *output port* component, which will be mapped to an LED.
7. Connect the clock inputs for these counters (and all counters used in this lab assignments) to the same clock input port. This input is connected to one of the DE1-SoC FPGA's 50MHz clock input pins.
8. **Compile** the schematic using the menu icon  or using Ctrl + L. Verify there are no errors.
9. Go to **Assignments** → **Pin planner**.
Connect the clock inputs to the 50MHz clock (**PIN_AF14**).
Connect the Pushbutton input to KEY[0] pushbutton (**PIN_AA14**).
Connect the output to LEDR[0] (**PIN_V16**).
10. Use I/O Standards **3.3-V LVTTTL** for all of them.
11. Compile again.

Our future designs using pushbuttons will rely on introducing the signal bouncing control, so it is important to guarantee the correctness of your design on the real hardware before we move on.

Assignment 1

Upload your design to the DE1-SoC board and test it on the LED0 with KEY0 pushbutton as the input. Pushing the button for at least a quarter second then releasing it should toggle the LED (change the state of the LED). Take a screenshot of your design and include it in the lab report

Pack the debounce schematic as a logic block named **debounce_t.bsf** for later use. This design will be used a *debounce* circuit connected to all the pushbutton inputs.

In the design above, the last 1-bit counter holds the value until the button is pressed again. This realizes a toggle behavior on the pushbutton. If you only need a single pulse from a pushbutton, you have to remove that last 1-bit counter for that specific pushbutton. Create a new project with a debounce circuit to produce a single pulse instead of a toggle. Call this debounce circuit *debounce_p* and pack it as a logic block named **debounce_p.bsf** for later use.

3. Designing an 8-bit Cascade Counter

Our next goal is to create a design with a free-running 8-bits counter whose output is connected to the 7-Segment displays on the DE1-SoC board. Every time the counter transitions to a new state, the 7-Segment displays will reflect the new count value.

The DE1-SoC's FPGA has a default clock frequency of 50MHz. This means that synthesizing a counter on the FPGA will make it change its state to the next count at this specific frequency. The state transitions would happen so fast that the intermediate values on the 7-Segment displays wouldn't even be observable by the human eye. We will leverage cascaded counters to reduce the counting speed.

1. Create a new project CascadeCounters then create a new schematic named **counter_8bits.bdf** and make sure to add it to the project by **Project → Add Current File to Project**.
2. Configure a new LPM_COUNTER (name it **Counter50M**) to count up to 50,000,000, which makes it overflow and reset to zero every one second. Set the counter's output number of bits to the minimum needed for 50,000,000. It should be *Up only, Modulus* counter with a count modulus value set to 50,000,000, and leaving everything else as default. Check everything on the Summary window as usual. Add an instant of this counter in the schematic file.
3. Configure a new LPM_COMPARE block (name it **Compare1**) to compare the output of **Counter50M** with a constant 1. Set the compare block to be an equality checker ($a == b$) with the same number of bits as the counter determined above, and *data_b* set to a constant 1.
4. Configure a second LPM_COUNTER (name it **Counter255**) as an 8-bits *Up only* counter, with *Count Enable* input, and leaving everything else as default except the Summary window where you need to check everything. This counter should be enabled with the output of **Compare1**.
5. Display the output value of **Counter255** on the 7-Segment displays. Utilize the **display_8bits** block you implemented in Lab 8 (make sure to copy all related files to this block such as the *dispal7_en*, *div8u*, etc.)
6. Before compiling your project and as the schematic file has a different name than the project, make sure to right click on the file name, *counter_8bits.bdf* → select *Set as Top-Level Entity*. You need to apply this setting in case your project has multiple schematic circuit files before compiling and testing any of the schematic circuits.

Assignment 2

Test the design on the DE1-SoC FPGA, and make sure that the speed of the counter is the one intended. Include a screenshot of your Schematic design in your report.

4. Pushbutton-Controlled Counter

The goal is to develop a pushbutton-controlled counter, similar to the one implemented in software in Lab 3. This time, you will design the counter in Quartus Prime schematic, and run it on the board FPGA. You will start with the cascaded counter controlling the LEDs that you already designed. The counting will be controlled by the pushbuttons. The overall functionality and switch assignment is as follows:

KEY0:	start or stop counting
KEY1:	load counter value from switches
SW[7..0]:	define the start value of the counter
SW[8]:	change the direction (i.e., increment vs. decrement)
SW[9]:	change the speed of the counter

Do not try to design this all at once. The design would be too complex. As with any design, it is a good idea to start with small steps. The following steps guide you through the steps individually.

4.1 Start / Stop Counter

Starting with the 8-bit cascaded counter design from the previous section, implement KEY0 operation to be able to start or stop the counter using the appropriate debounce system. Double click on the **Counter50M** counter and add the *Count Enable* input. Save changes for this instance of the LPM_COUNTER. Think of how you can use the KEY0 input and the appropriate debounce circuit to start/stop the counting. If prompted to update the symbol say “Yes” → Selected symbol(s) or block(s) → OK. Display the output value of the counter on the 7-Segment displays.

Note: whenever you change the configuration of an LPM instant component, the connections to the component might get disconnected. So, make sure to verify the connections and fix them if needed.

Assignment 3

Validate the functionality on the DE1-SoC board. Add a screenshot of your new design in your report. Describe in your lab report how you achieved the KEY0 functionality.

4.2 Bi-Directional Counter

Expand the design to toggle the direction of counting based on the position of SW[8]. If SW[8] is in the OFF position, the counter should count upwards (increment value), and if in the ON position, the counter should count downwards (decrement value). Double click on the **Counter255** counter and change the counter to be an up/down counter with *updown control* input. Save changes for this instance of the LPM_COUNTER and use the new input as needed. Display the output value of the counter on the 7-Segment displays.

Assignment 4

Validate the functionality on the DE1-SoC board. Add a screenshot of your new design in your report. Describe in your lab report how you achieved the direction control functionality.

4.3 Loading the Counter with a Value

Expand the design to allow the user to set a starting value of the **Counter255** counter through the (SW[7..0]) switches. The **Counter255** counter should load the value from the switches only if KEY1 is pushed. Otherwise, the 8-bit counter should count freely, as designed before.

Note that the load operation should be independent of the clock (asynchronous load). You only need to load the value from the switches once for every pushbutton. Use the appropriate debounce system for the KEY1 input. Double click and modify the **Counter255** counter configuration as needed for this operation. Display the output value of the counter on the 7-Segment displays.

Assignment 5

Validate the functionality on the DE1-SoC board. Add a screenshot of your new design in your report. Describe in your lab report how you achieved the load value operation functionality.

4.4 Controlling Counting Speed

Extend the design to control the counting speed of the **Counter255** counter using SW[9]. When SW[9] is off, **Counter255** counts with the default speed of 1 count per second. If SW[9] is on, **Counter255** counts with the a higher speed of 2 counts per second.

Hint: Think of how you can use the asynchronous *Clear* input of the **Counter50M** counter in the cascaded counter design. Implement a circuit to trigger a reset when the counter reaches a certain value as desired based on SW[9] status.

Display the output value of the counter on the 7-Segment displays.

Assignment 6

Validate the functionality on the DE1-SoC board. Add a screenshot of your new design in your report. Describe in your lab report how you achieved the change of speed functionality. Record a short video showing the functionality of all the pushbuttons and switches on the counter with output on the 7-Segment displays.

3. Lab Submission Instructions

- Each lab consists of a set of pre-lab questions and the lab assignments. You need to submit one lab report with the answers to the pre-lab questions and the lab assignments. If working in a group, only one of the two group members must submit one version of the lab report and the source code files with the names of both students on the report cover page and at the top of each source code.
- Write your lab report following the report template provided on Canvas.
- The pre-lab questions prepare you for the challenges you will be presented with in the actual lab. Have their answers ready before the lab session and include these answers in the lab report together with the lab assignments.
- Your source code must be well commented by explaining what the lines of your program do. Have at least one comment for every 4 lines of code. At the beginning of your source code files write your full name, students ID, and any special compiling/running instruction (if any).
- For each lab, submit the following on Canvas before the announced due date/time:
 - 0 The lab report as a single Word or PDF document
 - 0 The source code (the .cpp or .cc files) of all your programs.
 - 0 Any files the assignments might ask for (e.g., a demonstration video file)
 - 0 Submit each of the above files separately (do not upload compressed files).
- You can submit multiple attempts for this lab, however, only what you submit in the last attempt will be graded (i.e., all required reports and files must be included in this last attempt).