Lab 3: Introduction to State Machines

OBJECTIVES

- Understand the construction and use cases of a debounced switch circuit.
- Learn about the fundamentals of sequential logic design.
- Design counter circuits that have synchronous and asynchronous outputs.

INTRODUCTION

So far, we have only covered how to create combinational logic circuits, where the output of your circuit depends only on the current inputs to the circuit. In this lab, we will begin exploring sequential logic, where the output of your circuit depends on the current inputs to the circuit and the current state of the circuit. For sequential circuits, state refers to a small memory that provides some information about what the current "state" of the circuit. For example, a counter circuit that counts from zero to three, the state could be the current count value. With the current value remembered, it can then determine the next value at the appropriate time. The next state of the counter would become the next number in the sequence on the next active clock edge.

In general, state machines consist of three parts: a memory that holds the current state, some combinational logic that generates the next state from the current state, and possibly some combinational logic to determine the present outputs from the state and possibly some inputs. On every active clock edge, the output of the next-state logic becomes the current state, and a new next state is generated by the next-state logic. For an example output, you could create a signal that only becomes true in a certain state and is false in every other state. It is important to note that next-state logic and output logic are all based on the current state of the circuit.

LAB STRUCTURE

In this lab, you will become familiar with basic sequential logic circuits. In § 1, you will learn about how to create a safe clock signal using a debounced switch circuit and compare your debounced switch to a normal SPST switch circuit. You will design your first state machine in § 2 by creating a 2-bit counter circuit. The concepts learned from the 2-bit counter will be extended into a more complex 3-bit counter in § 3. Finally, you will encounter alternate ways to change your counter's next state by using TFFs and JKFFs in § 4.

REQUIRED MATERIALS

- Your entire lab kit (including your DAD)
- UF's DAD *Waveforms Tutorial*
- *Creating Graphical Components*
- Suggested Quartus Components
  - In "others | maxplus2" library
    - 7474: Dual D-flip flops
  - In "primitives | storage" library
    - dff
  - In "primitives | logic" library
    - not, and2, or2, bor2, etc.
  - In "primitives | pin" library
    - input, output
  - In "primitives | other" library
    - vcc, gnd

SUPPLEMENTAL MATERIALS

- *PLD on Breadboard Programming WARNING!*
- *DE10-Lite Pins*
- *DE10-Lite Manual*
- *DE10-Lite Schematic*

PRE-LAB PROCEDURE

1. DEBOUNCED SWITCH CIRCUIT

The switches that you have been using this semester are known as single-pole-single-throw (SPST) switches. When you move the SPST switch in a switch circuit from ON to OFF or from OFF to ON, the resulting output bounces around between low and high voltages for a short time. If the switch circuit output is used as a synchronizing signal (such as a clock) in a digital machine, weird things will happen. If the machine is a counter, the count may seem to jump between states wildly. This is undesirable, so a debouncing circuit must be built. Figure 0 shows a single-pole-double-throw (SPDT) switch. The switch shown on the top left of the figure is mounted on a printed circuit board (PCB) on the top right image for easier use with a breadboard. We have had past issues with SPDT switches like those on bottom of Figure 0, but they may work okay. You will use this debounced switch circuit in this lab and rest of the labs this semester.

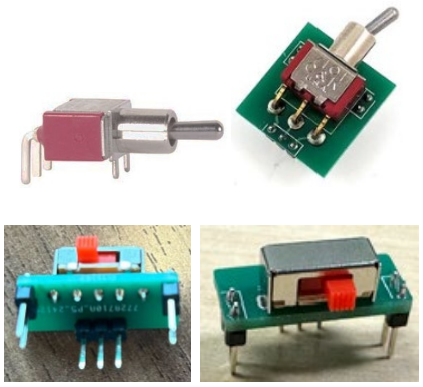

Figure 0: SPDT switch

The SPDT switch has three aligned pins. The center pin is connected to one or the other of the outside pins, depending on the position of the switch. You can use a multimeter on the resistance setting to verify the operation of this switch. There are two pins on the switch in addition to the three aligned pins. These two pins (shown in the middle of the top image in Figure 0) have no useful electrical purpose and are used for mounting only.

Your debounced switch circuit should use the two axial resistors (sometimes incorrectly called radial resistors) in your lab kit. A 1 kΩ ¼ W axial resistor, like the one you should have purchased at the beginning of the semester, is shown in Figure 1.

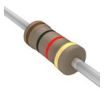

Figure 1: Axial resistor

To assure good connections to the breadboard, when flipping the SPDT switch, hold the switch down with another finger.

1. Design (on paper) a schematic for a debounced switch circuit for your clock input (as discussed in class) using your SPDT switch and other circuitry. Use a NAND chip or a NOR chip (74'00 or 74'02) in your design. You will build this circuit and then test it with your DAD.
2. An alternative is to use the PLD itself to provide the two NAND or two NOR gates. You are expected to know how to use the PLD for the two NAND or two NOR gates, i.e., to use only the PLD for your circuit designs (in addition to resistors and an SPDT switch outside the PLD on the DE10-lite). Note that if you use NANDs or NORs inside the PLD, the CLK signal input to the flip-flops can come directly from the internal SR-latch's output. To test this circuit with your DAD, you will need the CLK signal to be an output.
3. There is no easy way to test your debounced circuit without an oscilloscope. Luckily, the DAD has an oscilloscope function (called Scope). See the Oscilloscope (Scope) section of the DAD Tutorial for help in determining how to measure the switch bouncing. Quartus' simulation **CANNOT** be used to simulate the debouncing circuit that was taught in class because there are no resistor components available in Quartus. A voltmeter will not help, since the bounce rate is in the order of milliseconds. **Use your DAD to measure the bouncing of a normal (SPST) switch circuit.** See the Appendix for information on settings that may be helpful to observe the switch bouncing. Move the switch from one position to another and get a screen shot of the bouncing with an appropriate time base. Move the switch back to the original position and get a second screenshot. Move the switch one more time and get a third screen shot. It may be necessary to try each of these a few times to see the bouncing. Put at least three of these screenshots into your lab document and interpret these images by explaining the number of clocks that would occur if this switch was connected to a 5-bit counter that counts from 0 to 31.

4. Now use your DAD to measure the possible bouncing of your debounced circuit (with the SPDT switch) using the **external** 74'00 or 74'02 chip design. Move the switch from one position to another and get a screenshot of the possible bouncing (although there should be no bouncing) with an appropriate time base. Move the switch back to the original position and get a second screenshot.
5. Now use your DAD to measure the possible bouncing of your debounced circuit (with the SPDT switch) using the **internal** NANDs or NORs (inside your DE10-Lite PLD). To read the output of your SR-latch from the DE10-Lite, you can use one of the GPIO headers available on your breadboard using the ribbon cable and breadboard breakout. Move the switch from one position to another and get a screenshot of the possible bouncing (although there should be no bouncing) with an appropriate time base. Move the switch back to the original position and get a second screenshot. Unfortunately, at the present time, I'm not sure if a CLK signal generated internally in the PLD will be properly treated as a clock signal; therefore, at least for now, we will **NOT** use debounced circuits using internal NANDs or NORs (i.e., we will only use **external** NANDs or NORs). [After further investigation, it appears that it is **NOT** treated as a clock, i.e., it does not use the clock routing network, so this should probably **NOT** be used in a critical situation unless it is first thoroughly tested.]
6. When using an external debounced clock circuit, you must use pin 1 on the GPIO Header (PIN_V10 also called GPIO_[0]), as stated in the *DE10-Lite Pins* document.
7. Put all of these screenshots into your lab document and interpret these images explaining the number of clocks that would occur if this switch was connected to a 5-bit counter that counts from 0 to 31.

You will further test your debounced circuit by using it as a clock input to the small counter in the next part of the pre-lab. This will also serve as your first counter design, to assure that you understand the proper design technique before attempting a more complex counter later in this lab.

2. TWO-BIT COUNTER DESIGN

A synchronous counter is a device that progresses through a known sequence of numbers. The counter advances to the next state/number at a rising (or falling) edge of a clock signal. The counter sequence is arbitrary, i.e., it may count up, down, or in some strange sequence. The counter you will design in this lab will have a custom count sequence with some special additional inputs.

All counters, and for that matter, all state machines, should first be forced to a starting state. If the flip-flops have asynchronous presets and preclears, **each can independently** be used to put the counter/state machine into a desirable initial state. Then you can make the presets and preclears false to allow the counter/state machine to progress as required.

1. Design a counter (shown in Figure 2) to count through the sequence 00, 10, 11, 01, 00, … Note that there is only a single input (CLK) and two active-high outputs (Q1 and Q0). (To start the counter at a known value, use the pre-sets and pre-clears of the two flip-flops.)

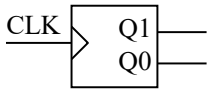

Figure 2: Simple counter block diagram.

   a. Make a next-state truth table for your counter. The "inputs" for this table are Q1 and Q0; the "outputs" are Q1+ and Q0+.
   b. Using D-flip-flops, determine the next state equations for $D_i = Q_i^+ = f(Q1,Q0)$. Use K-maps for each D_i to get MSOP or MPOS equations. Note: There will be two **2-input** K-Maps.
   c. Design the required counter circuit in Quartus (called Lab3_2bit_Cnt). (I suggest that you do it first on paper, but this is not required and will not be submitted.) I recommend that you use "primitives | storage | dff" available in Quartus, shown on the left in Figure 3 or "others | maxplus2 | 7474" on the right in Figure 3.

d. Add a single input, called **Start**, that can asynchronously put the counter in the state with outputs (count) 00. Use the presets or preclears on each of the flip-flops to accomplish this. **Do not allow any of the presets or preclears to remain unconnected. If you don't need one, connect it to "false," i.e., don't leave it floating or connect it "true." (False and true are either Vcc or GND.)**

e. Simulate the circuit and, as always, annotate this simulation. Verify that your design counts as required with each rising CLK edge. (Note that this clock will not bounce, but a normal SPST switch circuit CLK can bounce.) I suggest using the Clock tool in the waveform editor (see Figure 4) to generate the CLK signal. Add a screenshot of the simulation and annotate it (as always) to your lab document. This simulation will be for the case of a completely external debounce circuit.

Figure 4: Clock tool (in the center).

2. Create a **component** in Quartus for the Hex to 7-segment decoder you created in Lab 2. You can test your 2-bit counter with this component**.** See *Creating Graphical Components* for information on how to make a component in Quartus.

3. Test your 2-bit counter design using your debounced switch circuit (with **external**, not internal, NAND or NOR gates) for the clock input (CLK) and an SPST switch circuit as your Start input. Assign the CLK input to pin 1 on the GPIO Header (PIN_V10 also called GPIO_[0]). Use HEX0 on your DE10-Lite to display the current count (Q).

   a. Disconnect the USB connector for the DE10 from your computer.
   b. Connect the output of your debounced CLK circuit (that uses the SPDT switch) on your breadboard to pin 1 on the GPIO Header (PIN_V10 also called GPIO_[0]).
   c. Reconnect the USB connector for the DE10 to your computer.
   d. Program your PLD on the DE10 with your 2-bit counter design (that outputs the count, Q, to HEX0 on the DE10).

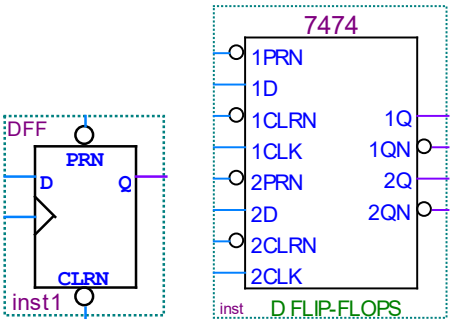

Figure 3: Two D-FF available in Quartus.

   e. Toggle (flip) the SPDT switch used for the debounced CLK input to verify that your counter counts as expected. If your counter output does not exactly match the required count sequence as you toggle the switch input (but it worked in simulation), then your debounced switch circuit is **not designed and/or built correctly**. If necessary, verify your debounced switch circuit design and construction. (The only way to easily test your debounced switch circuit is with a counter or an oscilloscope.)

4. Test your 2-bit counter design using a normal (un-debounced) SPST switch circuit for the CLK input.

   a. Remove the connection of the output of the SPDT circuit to DE10's pin 1 (also known as PIN_V10 and GPIO_[0] on the GPIO Header as shown in *DE10-Lite Pins*).
   b. Disconnect the USB connector for the DE10 from your computer.
   c. Connect the output of your SPST switch circuit's CLK on your breadboard to pin 1 on the GPIO Header).
   d. Reconnect the USB connector for the DE10 to your computer.
   e. Program your PLD on the DE10 with your 2-bit counter design (that outputs the count, Q, to HEX0 on the DE10).
   f. Toggle (flip) the SPST switch used for CLK. Write down the outputs with ten successive clocks. Compare each successive count to what you **should** get. How does counting with an un-debounced clock input compare to counting with a debounced clock input? Put this info in your submitted lab document.

3. THREE-BIT COUNTER DESIGN

In this portion of the lab, you will design a counter (called Lab3_3bit_Cnt) that will count forward with the following sequence:

000, 011, 100, 010, 111, 000, …

This counter will also count backward in the reverse order:

000, 111, 010, 100, 011, 000, …

A block diagram for the counter is shown in Figure 5.

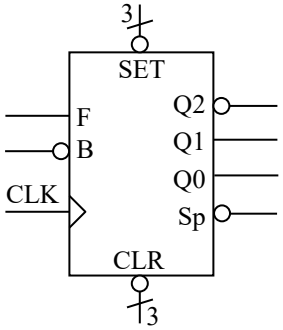

Figure 5: Forward/Back counter block diagram.

Your counter can also pause the counting. These three modes (forward [F], backward [B], pause) will be controlled with two inputs, F and B. When neither forward nor backward is true, the counter will ignore the CLK input and hold its count value, i.e., pause.

F and B should **never** be simultaneously true, so your counter should deal with this case in the most cost-effective fashion. If you assume that a user will never make both inputs true, you can design the least expensive circuit that can accomplish this required goal by using "don't cares." Note that the counter does not include $Q_2Q_1Q_0$=%001, %101 and %110, where % is a prefix for binary. These three counts should contribute "don't cares" in your next-state truth tables and K-maps.

Your counter should have a means to **asynchronously** set and clear **each** bit. SET_i(L) and CLR_i(L) are the inputs to asynchronously set and clear a particular counter bit. These SET and CLR inputs will allow you to start the counter at any desired count. (If you initialize your counter at the count $Q_2Q_1Q_0$=%001, %101 or %110, the next count is not specified in the problem description. The next count will be determined by the values selected for the "don't cares" associated with these counts.)

As you may recall, when we first started discussing circuits with feedback, I stated that with these types of circuits it is often easier to deal with voltages rather than with logic. Let me suggest that you design this (and all) counter(s) with **active-high state-bits** (to generate the next-state circuits) and then generate the appropriate output circuits with the required activation levels. In this case, use active-high Q_2, Q_1, and Q_0 in your design of the counter next state circuits. However, when creating the final circuit, the outputs will be as shown in the block diagram of Figure 5.

Finally, the counter should have an additional output indicating the count is at a "special value," Sp. Special should be true only when the count is "**011**" and F is true or when the count is "**100"** independent of the values of B and F.

1. Make a next-state truth table with the inputs: F, B, Q2, Q1, Q0 and outputs $Q2^+$ $Q1^+$, $Q0^+$, and Sp. (Ignore the SET and CLR for the design. These are controlled directly with the FF set and clear inputs.)
2. Using D-flip-flops, determine the next state equations for $D_i = Q_i^+ = f(F,B,Q2,Q1,Q0)$. Use a K-map for each output to determine MSOP or MPOS equations. Note: There will be three **5-input** K-Maps.
3. Determine the equation for the output Sp. Use a K-map to get an MSOP or an MPOS equation.
4. Design the required counter circuit (called Lab3_3bit_Cnt) in Quartus. (I suggest that you do it first on paper, but this is not required and will not be submitted.) Don't forget to include the SET_i and CLR_i inputs in your circuit, for i = 0, 1, and 2.
5. Simulate the circuit and add annotations.
   a. Verify that your design counts forward, counts backwards and holds the count with the appropriate input combinations.
   b. Verify that each bit can be set and cleared by using the SET_i and CLR_i inputs.
   c. What is the next count for each value of F and B for $Q_2Q_1Q_0$=%001, %101, and %110?
   d. As always, include the circuit schematic (with PLD pin numbers) and annotated Quartus simulation results in your lab document. (Also as usual, submit your archived Quartus file.)
6. Build this circuit on your breadboard. You can undo the 2-bit counter if you would like to, since you will not demo this in lab. Reprogram your DE10 and verify that this counter operates properly. Use a debounced switch circuit for the CLK input. Use appropriate switch circuits for

the other inputs and your DAD for the count (Q_2, Q_1, Q_0) and Sp outputs. Note that the DAD does not deal directly with active-low outputs, so Sp on the DAD will be opposite of what it would be if you used an active-low LED circuit. In addition to using the DAD for the outputs, you must also use the 7-segment display (with the Hex to-7-segment Decoder designed in Lab 2) for the Q outputs. If you want to have both counters work at the same time (the two-bit counter from Part 2 and the three-bit counter from this section), then use HEX1 for this section. The decimal version of the Q state-bits should be directly viewed using HEX0 (or HEX1) on the DE10. Use the dot on HEX0 (or HEX1) for the Sp output (but remember that it will be opposite of what it would be if Sp was active-high).

4. TWO-BIT COUNTER WITH ALTERNATE FLIP-FLOPS

Now re-design the **2-bit** counter (from part 2, Lab3_2bit_Cnt) using a T-FF for state bit 1 (the most significant bit) and a JK-FF for state bit 0 (the least significant bit). This new design Quartus (called Lab3_2bit_JK_T) will require that you add to your next-state truth table from part 2a, determine equations for the T1, J0, and K0 inputs, draw and simulate the new circuit diagram in Quartus, and verify with the simulation that it counts properly. You do **NOT** need to build/demo this circuit on your breadboard, but you should submit the archive file, the annotated simulation, and all of your work, as usual.

PRE-LAB PROCEDURE SUMMARY

1. Learn about how we can debounce user inputs to create a clock signal in § 1. Use the DAD to visualize the bouncing of an SPST circuit and the no bouncing of a proper SPDT debouncing circuit.
2. Design a simple two-bit counter in § 2 with various clock inputs.
3. Design a three-bit counter with more complicated next-state logic in § 3.
4. Learn how to use different types of flip-flops to implement a counter in § 4.

IN-LAB PROCEDURE

1. Complete the lab quiz.
2. Demonstrate the correct function of the 3-bit counter designed in § 3.

APPENDIX

In order to see the switch bouncing, use the following DAD settings.

- Time Base: 50 us/div (or 20 us/div)
- Offset: 0
- Level: 1.5 V
- Condition: Either
- Mode: Repeated, Normal
- Range: 1 V/div

Figure A.1 shows the time base at 1 ms/div. Note that the bouncing is apparent, but the amount of bouncing cannot be determined because of the too large time base.

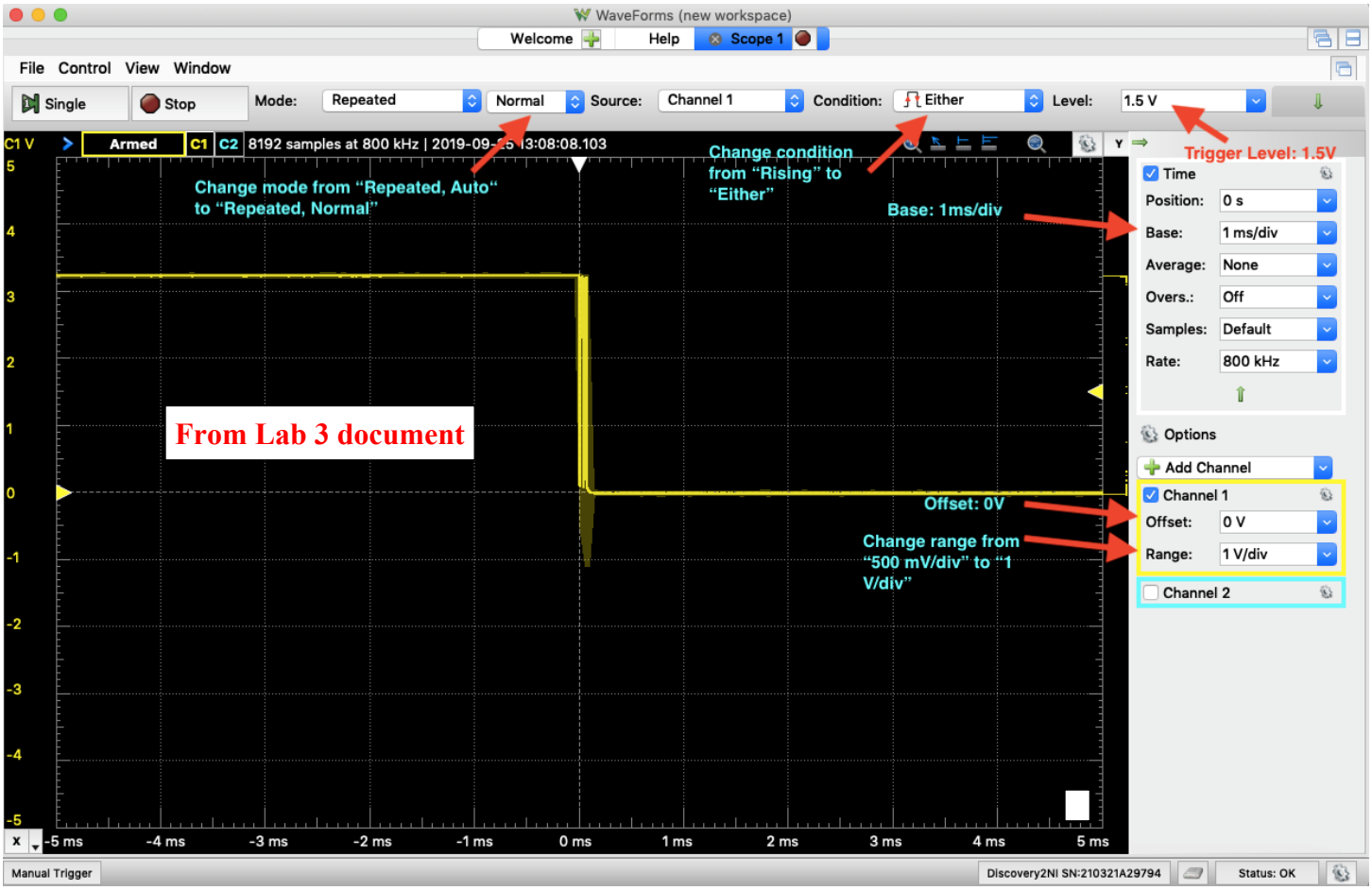

Figure A.1: Bouncing with time base of 1 ms/div.

Figures A.2 and A.3 show bouncing with the time base at 50 us/div and 20 us/div, respectively.

A recording of switch bouncing with different time base settings is available at the following location on our class website: https://mil.ufl.edu/3701/docs/Bouncing.mov.

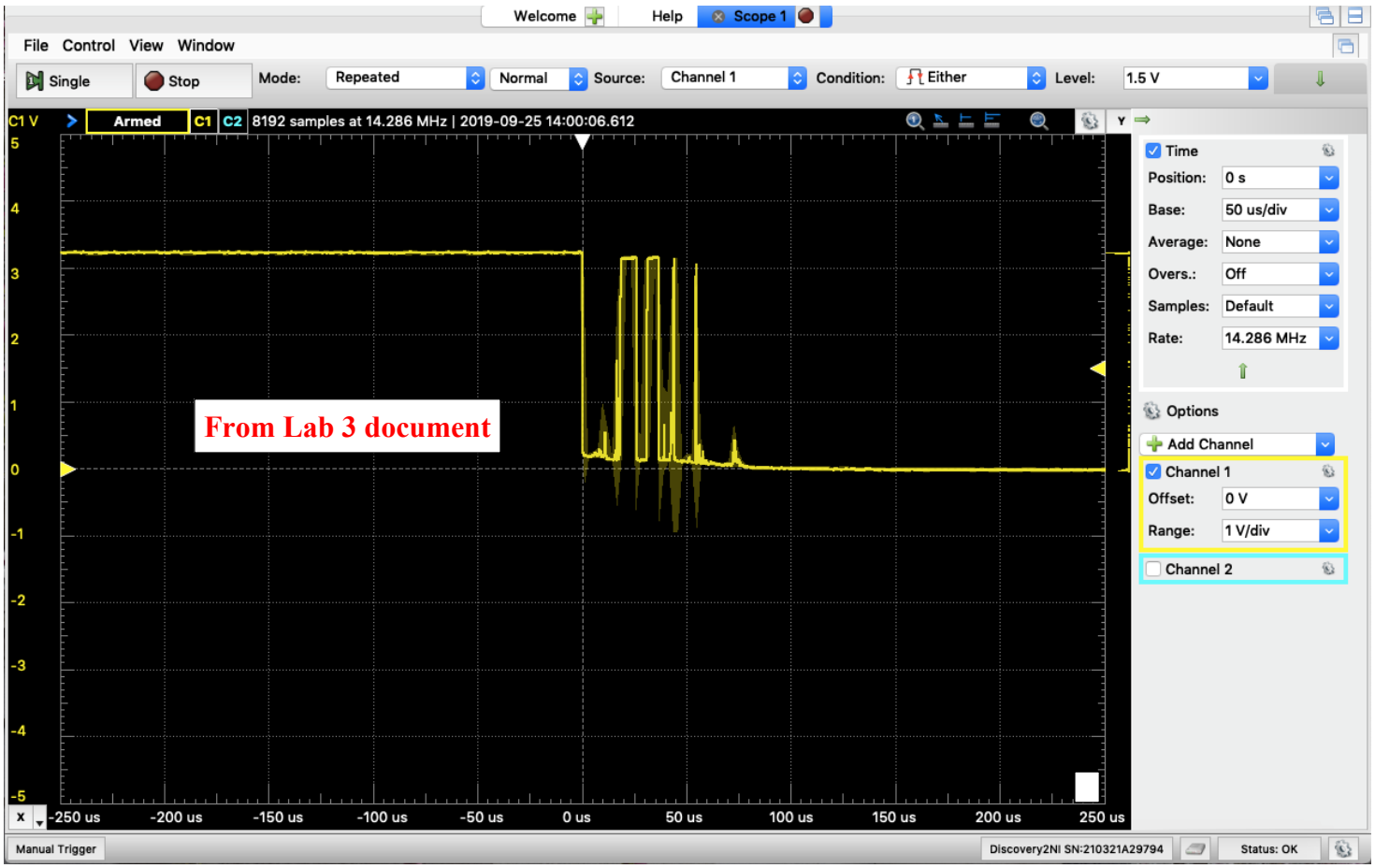

Figure A.2: Bouncing with time base of 50 us/div.

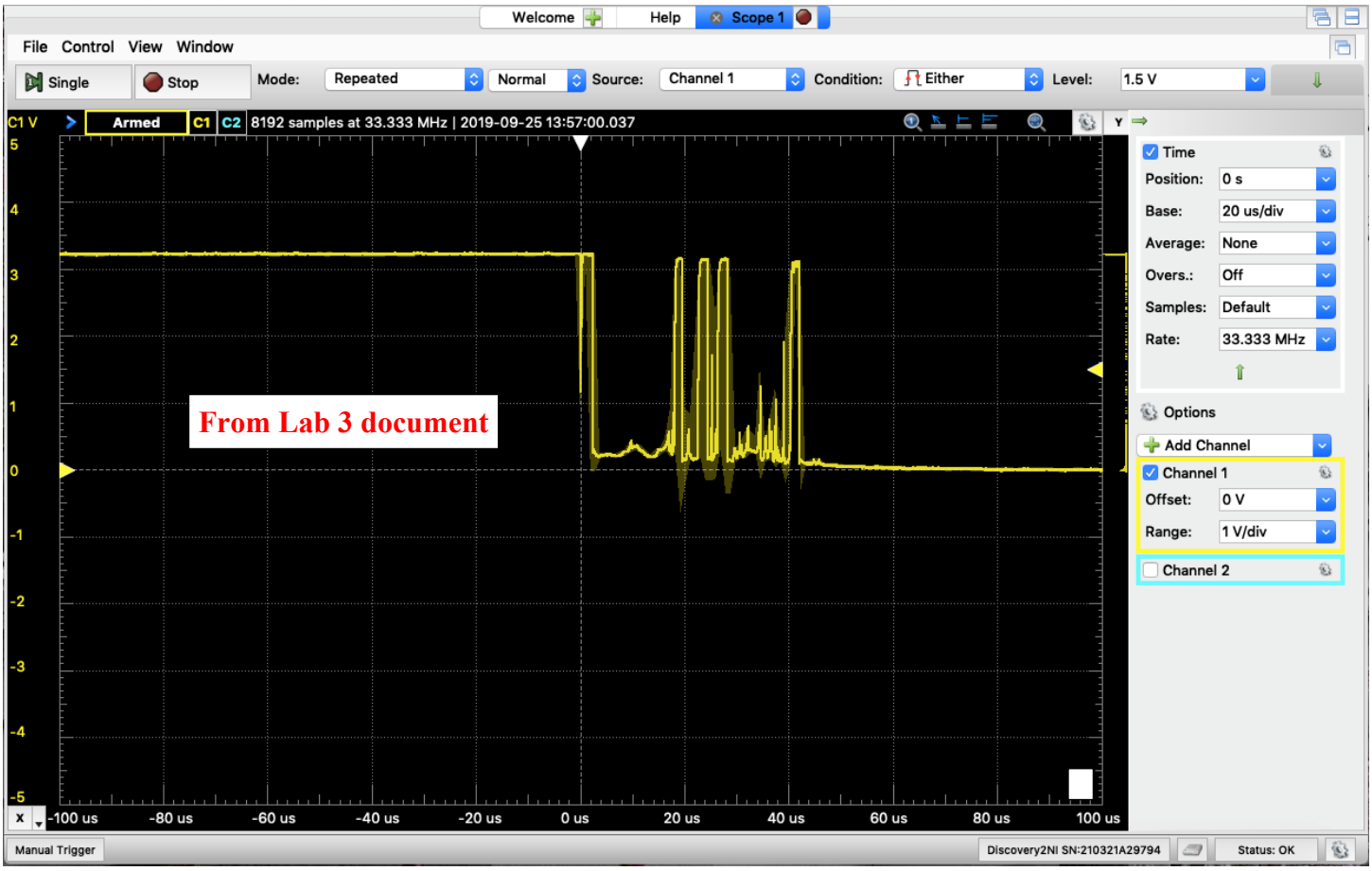

Figure A.3: Bouncing with time base of 20 us/div.