Lab Sheet for Week 1

Lab Instructor: **RSP**

Lab 1: Debugging MCU Behavior Using an Oscilloscope

Objective

- Refresh practical skills using lab instruments such as digital oscilloscopes, function generators and logic analyzers.
- Learn how to use basic and advanced trigger functions of a digital oscilloscope.
- Learn how to debug the behavior of an MCU by monitoring / analyzing its I/O signals.

Equipment

- Arduino Board (Arduino Uno or Nano) + USB Cable
- Digital Oscilloscope + Measurement Probes (two or more)
- Digital Function Generator
- Jumper Wires
- Computer with Arduino IDE software (version 2.x)

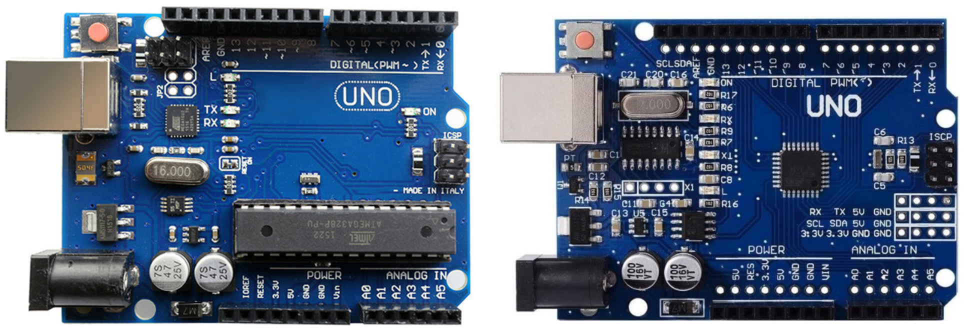
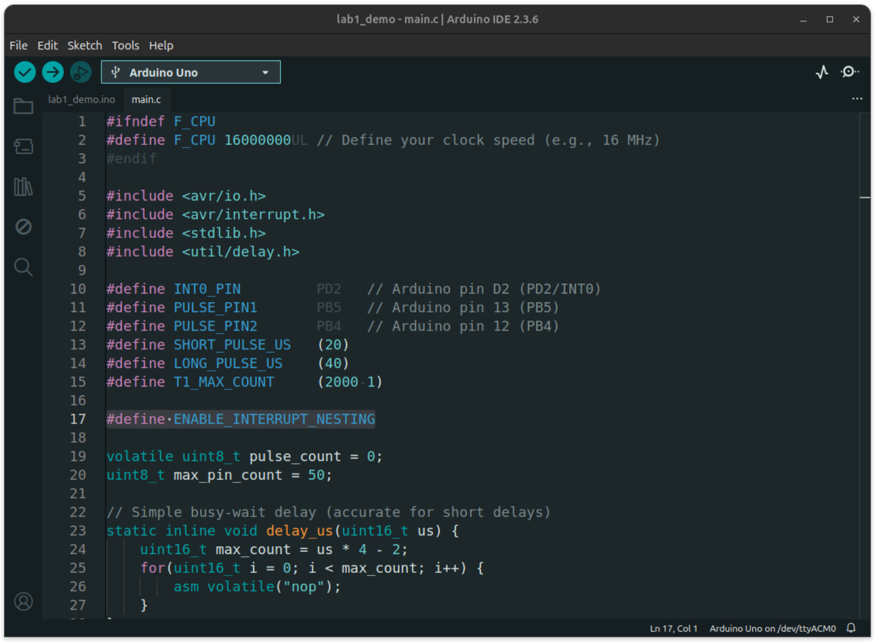


Figure 1: Arduino Uno boards (ATmega328P in PDIP-28 and TQFP-32)

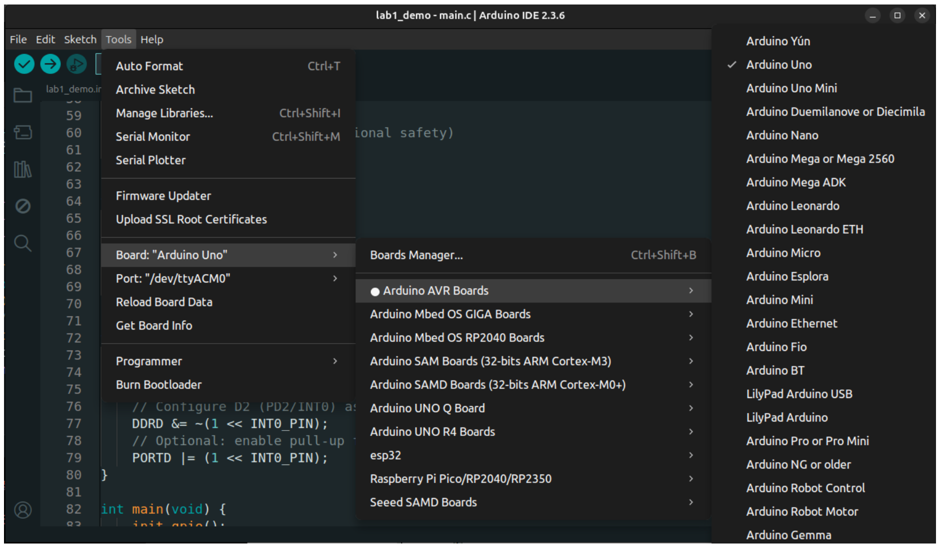
Lab Procedure

1. Open the **Arduino IDE**.
    - Delete all code in the default **.ino** sketch.
    - Create a new tab and add a source file named **main.c.**
    - Use the source code provided as an example (see **Code Listing 1**).
2. Upload the compiled Arduino sketch (the **.hex** bin) to the **Arduino board.**
    - After building the Arduino sketch, upload the generated **.hex file** to the Arduino board.
    - Choose the right serial COM port corresponding to the attached Arduino board.
3. Set up the digital oscilloscope and verify that all probes are working and properly connected.
    - Connect the ground of the scope to the Arduino ground (common ground).
    - Connect a probe on **CH1** to the **Arduino D12 pin** (used as an output pin)
    - Adjust the trigger settings or conditions to obtain a stable waveform.


lab1_demo - main.c | Arduino IDE 2.3.6
File Edit Sketch Tools Help
Arduino Uno
lab1_demo.ino
main.c
Ln 17, Col 1
Arduino Uno on /dev/ttyACM0

```c
#ifndef F_CPU
#define F_CPU 16000000UL // Define your clock speed (e.g., 16 MHz)
#endif

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <util/delay.h>

#define INT0_PIN         PD2   // Arduino pin D2 (PD2/INT0)
#define PULSE_PIN1       PB5   // Arduino pin 13 (PB5)
#define PULSE_PIN2       PB4   // Arduino pin 12 (PB4)
#define SHORT_PULSE_US   (20)
#define LONG_PULSE_US    (40)
#define T1_MAX_COUNT     (2000-1)

#define ENABLE_INTERRUPT_NESTING

volatile uint8_t pulse_count = 0;
uint8_t max_pin_count = 50;

// Simple busy-wait delay (accurate for short delays)
static inline void delay_us(uint16_t us) {
    uint16_t max_count = us * 4 - 2;
    for(uint16_t i = 0; i < max_count; i++) {
        asm volatile("nop");
    }
```

Figure 2: Arduino IDE (with demo source code in **main.c)**, running on **Ubuntu**



```c
ional safety)
    // Configure D2 (PD2/INT0) as
    DDRD &= ~(1 << INT0_PIN);
    // Optional: enable pull-up
    PORTD |= (1 << INT0_PIN);
}

int main(void) {
```

Figure 3: Selecting the Arduino target board and the correct serial COM port

4. Measure the **high pulse width** (in microseconds) of the output signal on the **PB5 pin (Arduino D13 pin)** using a **digital oscilloscope**.
   - Record the waveforms displayed on the oscilloscope's screen for inclusion in the lab report.

5) Measure the pulse rate, the pulse-width values. Capture the corresponding waveform in each for inclusion in the lab report.
   - **Pulse rate (pulses / sec)**
   - **Minimum pulse width (usec)**
   - **Maximum pulse width (usec)**

6) Assume that the expected behavior of the MCU is to periodically generate pulses with a width of approximately t_{PW} = 25 µs (±2.5 µs). Describe how to use a digital oscilloscope with appropriate trigger settings to detect any invalid pulses, and explain the procedure for performing this task.

```
#ifndef F_CPU
#define F_CPU 16000000UL // Define your clock speed (e.g., 16 MHz)
#endif

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <util/delay.h>

#define INT0_PIN         PD2   // Arduino pin D2 (PD2/INT0)
#define PULSE_PIN1       PB5   // Arduino pin 13 (PB5)
#define PULSE_PIN2       PB4   // Arduino pin 12 (PB4)
#define SHORT_PULSE_US   (20)
#define LONG_PULSE_US    (40)
#define T1_MAX_COUNT     (2000-1)

#define ENABLE_INTERRUPT_NESTING

volatile uint8_t pulse_count = 0;
uint8_t max_pin_count = 50;

// Simple busy-wait delay (accurate for short delays)
static inline void delay_us(uint16_t us) {
    uint16_t max_count = us * 4 - 2;
    for(uint16_t i = 0; i < max_count; i++) {
        asm volatile("nop");
    }
}

ISR(INT0_vect) { // ISR for external interrupt 0 (INT0)
    // Toggle D13 twice
    PINB = (1 << PULSE_PIN1);
    PINB = (1 << PULSE_PIN1);
}

ISR(TIMER1_COMPA_vect) { // ISR for Timer1 Compare Interrupt
    pulse_count++;
    // Toggle PB5 and PB4 simultaneously
    PINB = (1 << PULSE_PIN2);
#ifdef ENABLE_INTERRUPT_NESTING
    sei(); // enable globla interrupts
#endif
    if (pulse_count >= max_pin_count) {
        delay_us(LONG_PULSE_US);   // 40 us
        pulse_count = 0; // Reset pin count
    } else {
        delay_us(SHORT_PULSE_US);  // 20 us
    }
    // Toggle again after delay
    PINB = (1 << PULSE_PIN2);
}

// Code continues on the next page...
```

Code Listing 1: AVR C code for **main.c** (**Part 1**)

```
// Code continues from the previous page…

void init_external_interrupt(void) {
    // Configure INT0 (PD2) to trigger on any logical change
    // ISC01 = 0, ISC00 = 1
    EICRA |= (1 << ISC00); // Any edge
    EICRA &= ~(1 << ISC01);
    // Enable INT0 interrupt
    EIMSK |= (1 << INT0);
    // Clear interrupt flag (optional safety)
    EIFR |= (1 << INTF0);
}

void init_timer1(void) {
    TCCR1A = 0;
    TCCR1B = (1 << WGM12);
    OCR1A  = T1_MAX_COUNT;
    TCCR1B |= (1 << CS11);  // prescaler = 8
    TIMSK1 = (1 << OCIE1A); // enable compare match interrupt
}

void init_gpio() {
    // Set PB5 and PB4 as outputs
    DDRB |= (1 << PULSE_PIN1) | (1 << PULSE_PIN2);
    PORTB &= ~((1 << PULSE_PIN1) | (1 << PULSE_PIN2));
    // Configure D2 (PD2/INT0) as input
    DDRD &= ~(1 << INT0_PIN);
    // Optional: enable pull-up for PD2 if needed
    PORTD |= (1 << INT0_PIN);
}

int main(void) {
    init_gpio();
    init_external_interrupt();
    init_timer1();
    sei(); // enable global interrupts
    while (1) {
      _delay_ms(100);
      max_pin_count = 20 + rand() % 100;
    }
    return 0;
}
```

Code Listing 1: AVR C code for **main.c** (**Part 2**)

Lab 2: Digital Signal Measurement with Oscilloscopes

Objective

- Refresh practical skills using lab instruments such as digital oscilloscopes, function generators and logic analyzers.
- Learn how to use basic and advanced trigger functions of a digital oscilloscope.
- Learn how to debug the behavior of an MCU by monitoring / analyzing its I/O signals.

Equipment

- Arduino Board (Arduino Uno or Nano) + USB Cable
- Digital Oscilloscope + Measurement Probes (two or more)
- Digital Function Generator
- Jumper Wires
- Computer with Arduino IDE software (version 2.x)

Lab Procedure

Part I

1. Use the same **Arduino sketch** as in **Lab 1**.
2. Disconnect the **Arduino board (Uno / Nano)** from the **5V DC supply** by unplugging the USB cable.
   - Use a jumper wire to connect the **Arduino D12** pin to the **Arduino D2 pin**.
   - The **D12** pin is a digital output pin, whereas **D2** is a digital input pin.
   - The output on **D12** is used as a test input signal to the **D2** pin.
3. Connect a probe on **CH1** to the **D12** pin, and connect a probe on **CH2** to the **D13** pin (used as a digital output pin).
4. Reconnect the **Arduino board** to the **5V DC supply**.
5. Adjust the trigger settings or conditions to obtain a stable waveform.
6. Measure the delay between the rising or falling edge of the signal on **CH1** and the rising edge of the signal on **CH2**. Record the waveform for inclusion in the lab report.
7. Measure the pulse width of the signal on **CH2**. Record the waveform for inclusion in the lab report.
8. Describe the pulse shape of the output signal (**D13**) on **CH2**.
9. Test the Arduino sketch with the **Wokwi simulator,** use a virtual logic analyzer to measure the I/O signals. Use a software program such as **GTKwave** or **Surfer** to visualize the waveform file (.vcd).

Part II

1. Setup a **digital function generator** to produce a **pulse signal** with the following parameter:

- **Vpp = 5V**
- **Voffset = 2.5V**
- **Frequency = 1kHz**
- **Duty cycle = 50%**

Warning: Ensure that the input voltage is between 0V and 5V (and *no input negative voltages*).

2. Apply the pulse signal to test the **Arduino** board.

- Connect the test signal to **D2** pin.
- Connect the **GND** of the function generator and the oscilloscope to the **Arduino GND** (***common ground***).

3. Use the digital oscilloscope to measure measure the signals.

- **CH1**: test signal from the function generator
- **CH2**: response signal from the **D13** pin on the Arduino board.

4. Measure the delay between the rising or falling edge of the signal on **CH1** and the rising edge of the signal on **CH2**.

5. Measure the pulse width of the output signal on **CH2**.

6. Explain the behavior of the Arduino sketch running on the Arduino board.

- Describe when a pulse occurs on the output signal.
- If the frequency of the input signal is increased to higher values such as 100kHz, 200kHz and 500kHz, does the MCU behave as expected?

6. Explain the behavior of the MCU if the **C macro** `ENABLE_INTERRUPT_NESTING` is not defined by default. Use the oscilloscope to analyze the I/O signals. If differences can be observed, record the waveforms and explain them.

How to set the trigger condition to detect pulse width wider than 25usec.

=> Use Pulse as trigger type with pulse width >= 27usec.

Lab 1:

- The output signal is a periodic signal with pulses of 25 usec +/-2.5usec, but in some cases, the pulse width is wider (approx. 50usec).

Lab 2 Part I:

- Feedback: The output on D12 pin is used as the input signal for D2 pin.
- The output on D13 pin is the response of the MCU. Each time there is a rising on falling edge on D2 pin, the MCU generate a short pulse on D13.

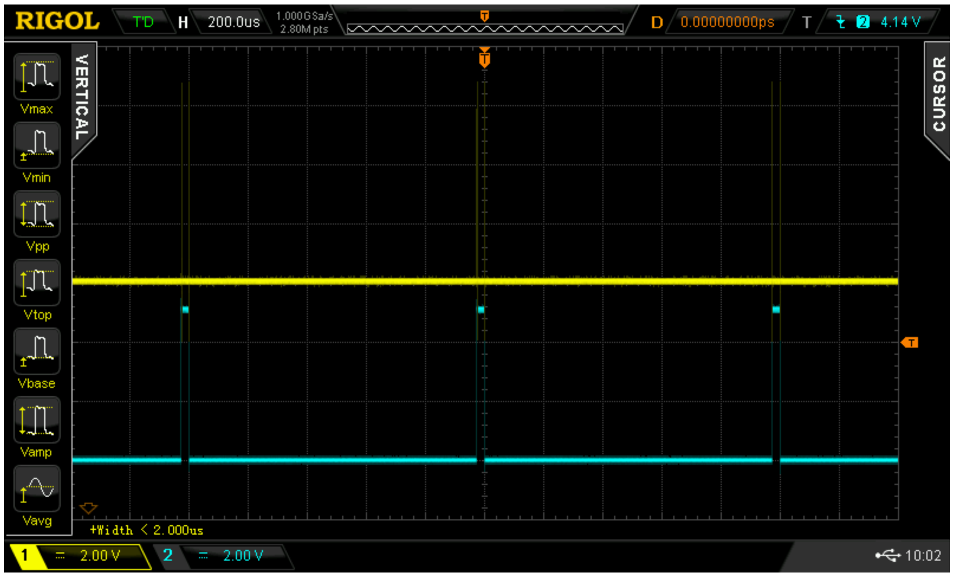


Figure: CH2 = pulse signal, interval = 1000 usec (1msec) or 1k pulses/sec

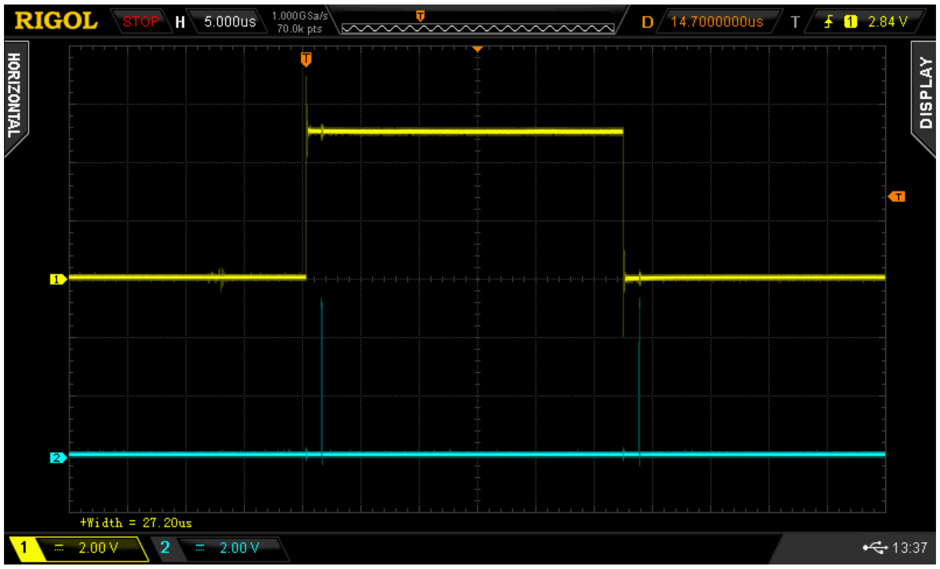


Figure: CH1 = Output signal with pulse width of around **27usec**

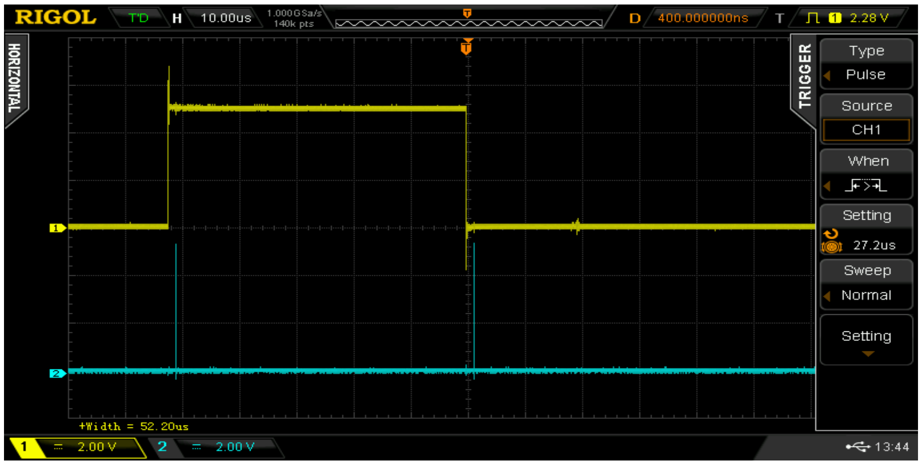


Figure: CH1 = Output signal with pulse width of around **52usec**

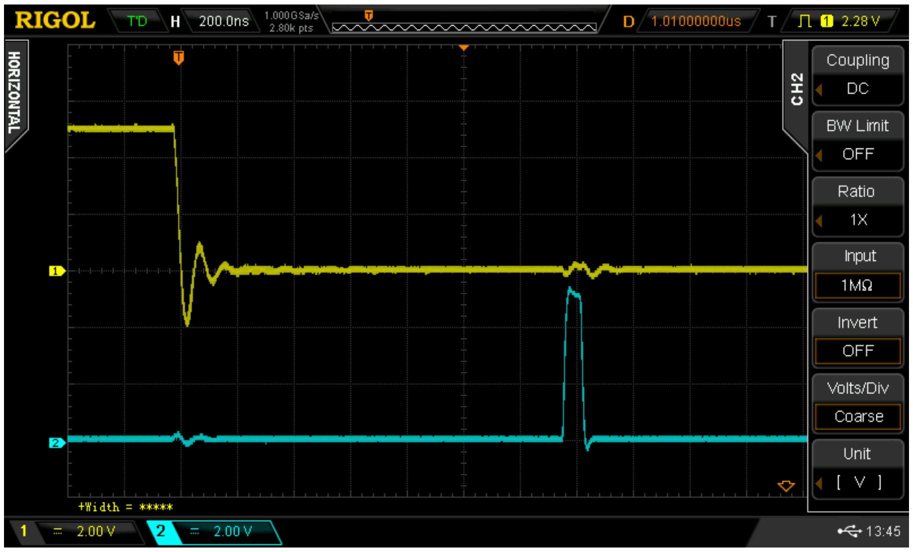


Figure: CH1 = input signal (falling edge), CH2 = output signal on D13 (response)

Measurement

- Output pulse width: 27 usec (min) and 50.2 usec (max)
- Pulse rate: 1k Samples / sec
- Delay between the input and output (response time): ~1.4 usec
- Pulse width: ~0.25 usec

If the C macro is not defined.

- The pulse width is reduced, close to 25 usec and 50 usec, but the response time (delay) for falling edge is increased from 1.4 usec to 2.2 usec.
- The MCU responds only to the falling edge on the input signal and the rising edge is ignored.

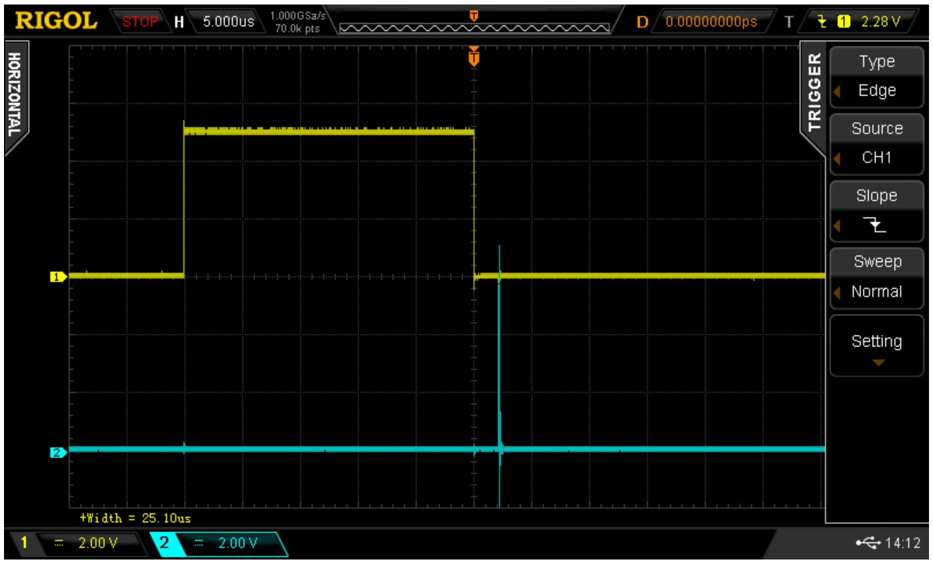


Figure: CH1 = pulse width 25 usec

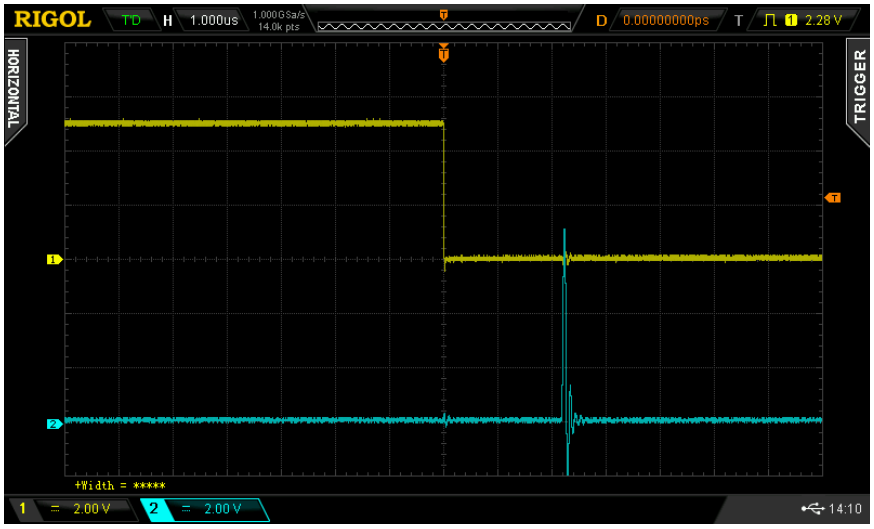


Figure: CH1 = pulse width 25 usec, CH2 delay = 2.2usec

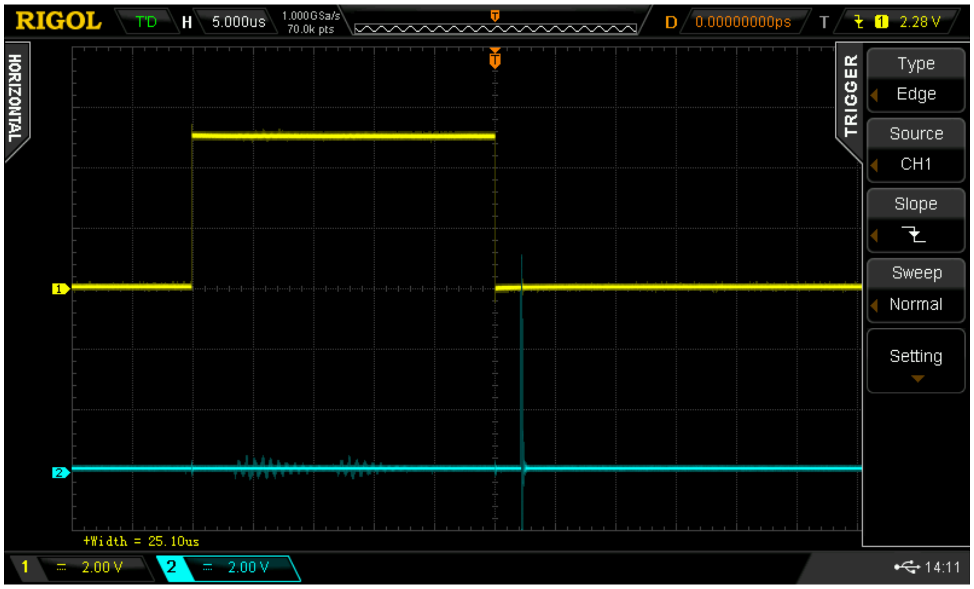


Figure: CH1 = pulse width 50 usec

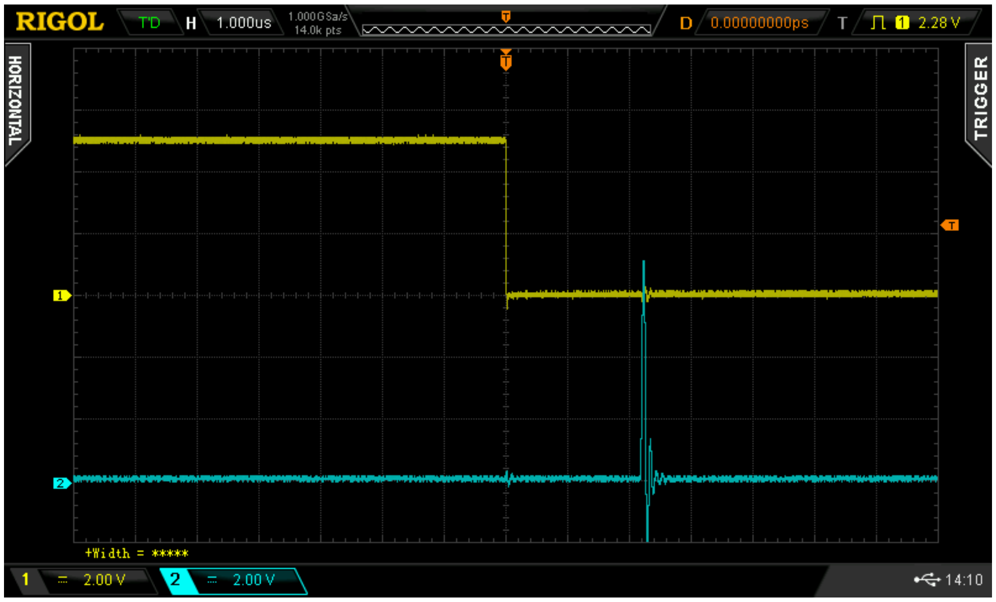


Figure: CH1 = pulse width 50 usec, CH2 delay = 2.2 usec

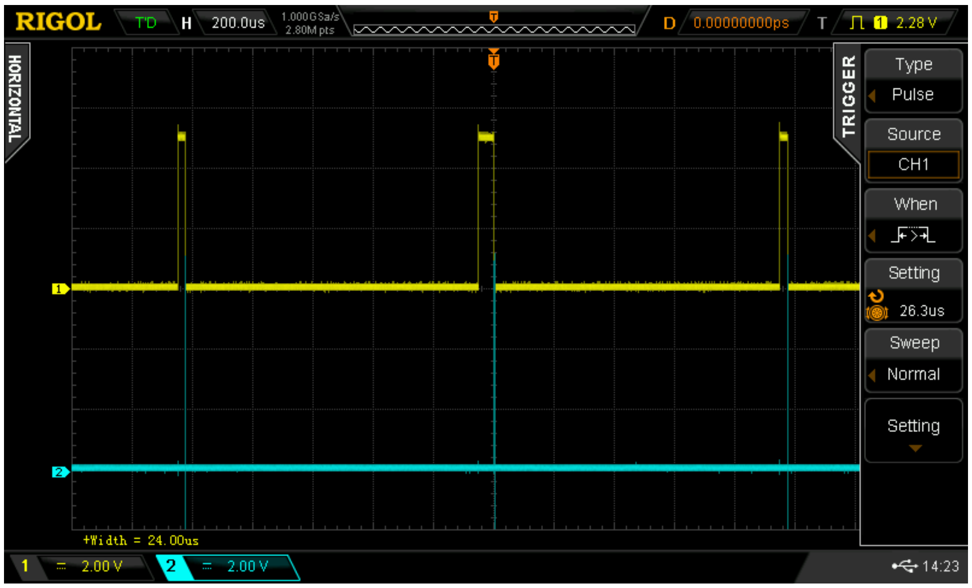


Figure: CH1 input signal, CH2 output signal
Output pulses occur only after falling edge of the input signal.