

The background is a close-up, slightly blurred image of a green printed circuit board (PCB). A large, square, gold-colored microcontroller chip is the central focus, with its pins visible. Other components like smaller chips, capacitors, and a circular component are also visible on the board.

MCT 4334

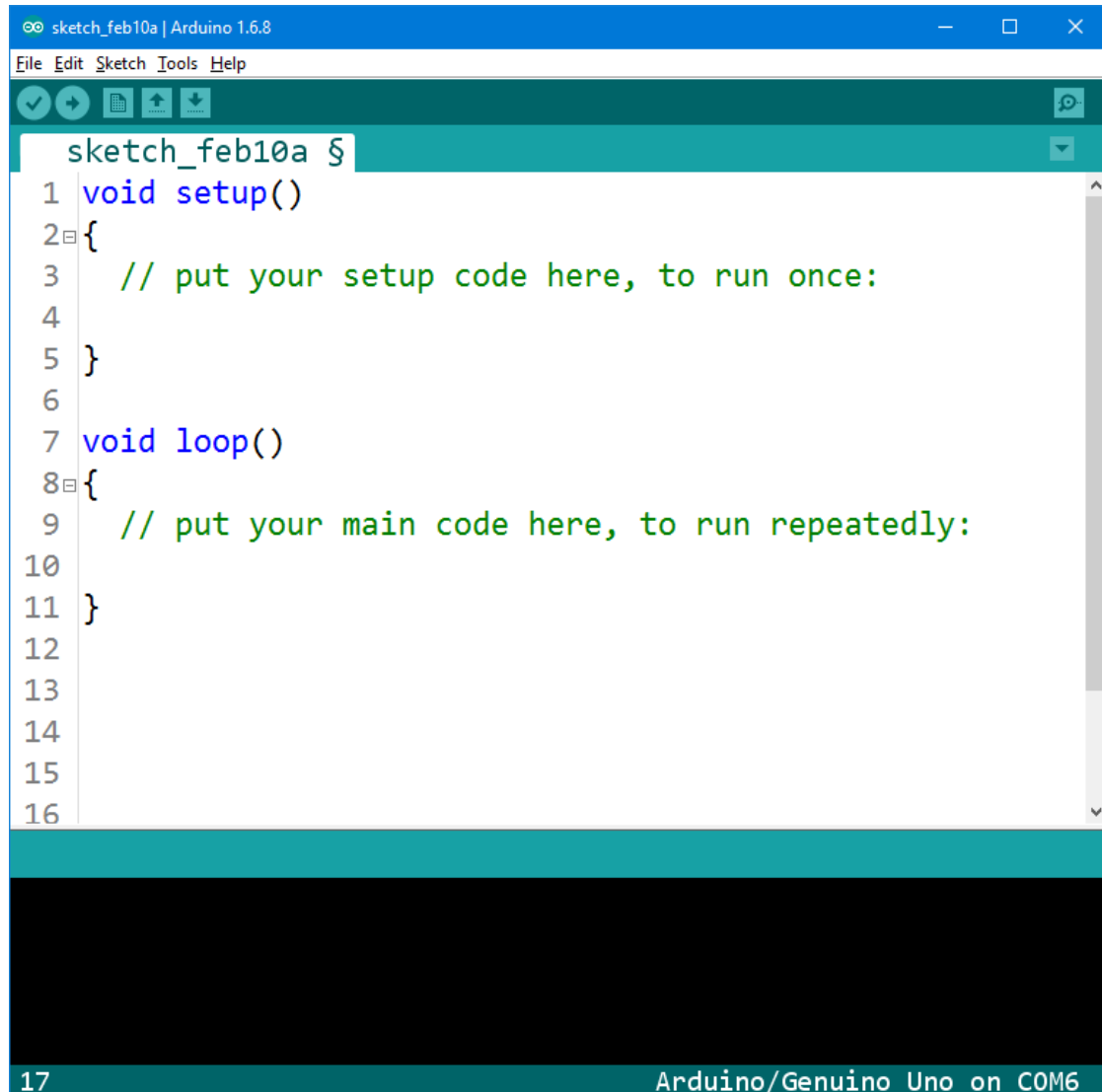
Embedded System Design

Week 04 GPIO

Outline

- Accessing GPIO using Arduino library
- Theoretical background on GPIO
- Accessing GPIO directly
- Programming examples

Arduino program

A screenshot of the Arduino IDE interface. The window title is "sketch_feb10a | Arduino 1.6.8". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for opening, saving, and running. The main text area shows a C++ sketch with the following code:

```
1 void setup()  
2 {  
3   // put your setup code here, to run once:  
4  
5 }  
6  
7 void loop()  
8 {  
9   // put your main code here, to run repeatedly:  
10  
11 }  
12  
13  
14  
15  
16
```

The status bar at the bottom indicates "17" on the left and "Arduino/Genuino Uno on COM6" on the right.

- The simplest Arduino program
- It does nothing.

Writing digital output

```
void setup()  
{  
    pinMode(13, OUTPUT);  
}  
  
void loop()  
{  
    digitalWrite(13, HIGH);  
    delay(1000);  
    digitalWrite(13, LOW);  
    delay(1000);  
}
```

- This program turns on the built-in LED on PB5 (Arduino pin #13) for 1000 ms and then turn off for 1000 ms and then repeats.
- The delay function makes the CPU repeatedly loop until the prescribed time in milliseconds has elapsed.

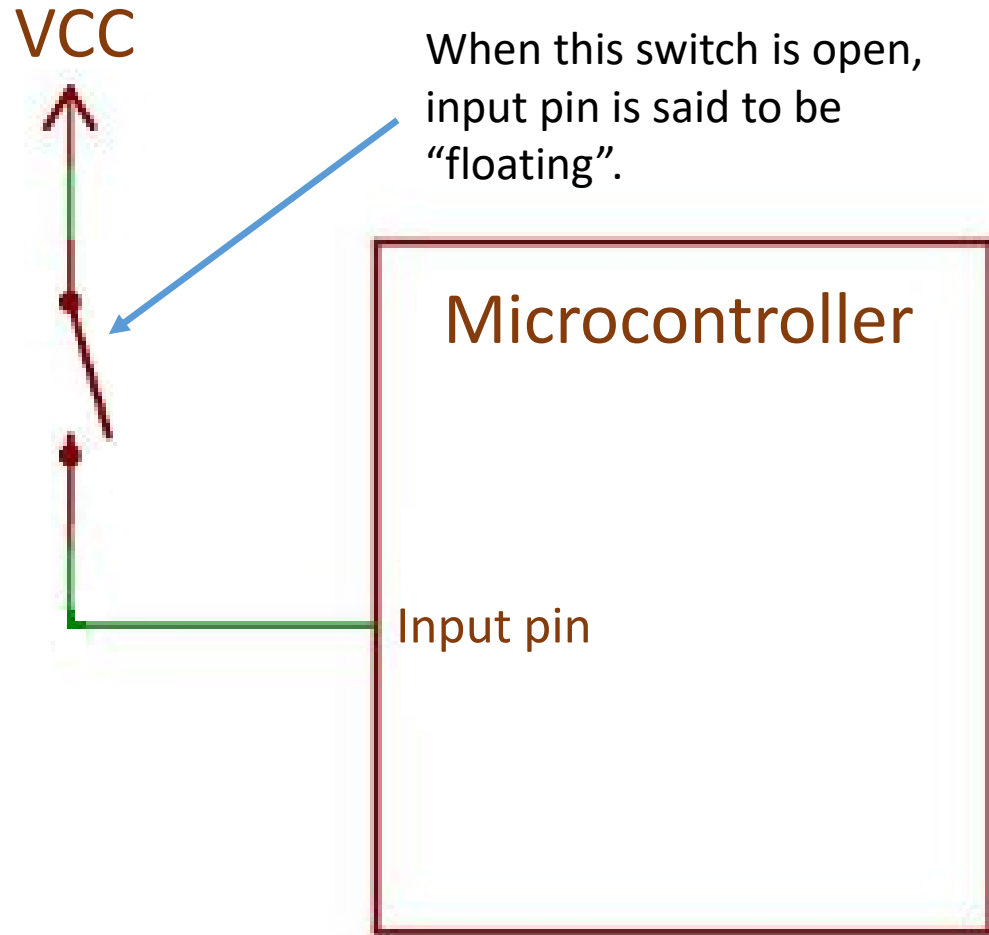
Reading digital input

```
void setup()
{
    pinMode(13, OUTPUT);
    pinMode(12, INPUT);
    Serial.begin(9600);
}

void loop()
{
    int state = digitalRead(12);
    if (state==HIGH)
    {
        digitalWrite(13, HIGH);
    }
    else
    {
        digitalWrite(13, LOW);
    }
}
```

- This program constantly checks the status of digital input on PB4.
- If the digital input is HIGH, it turns on the built-in LED on PB5.
- This concept is known as “polling”.
- The CPU is always busy waiting for input.
- Usually not the best way.

Floating pin



- **This is how NOT to do.**
- CMOS chips are sensitive to floating pins.
- The input pins have to be connected to **either** VCC or GND.

VCC = HIGH
GND = LOW
Floating = indeterminate (unstable)
- The input signal becomes unreliable if the input pin is floating

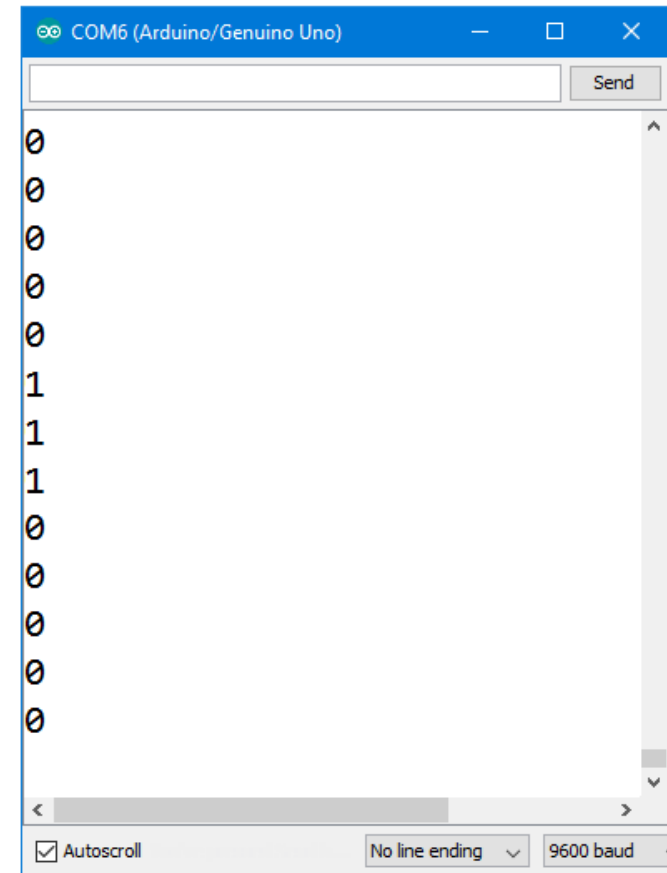
Digital input with floating pin

```
void setup()
{
    pinMode(13, OUTPUT);
    pinMode(12, INPUT);
    Serial.begin(9600);
}

void loop()
{
    int state = digitalRead(12);
    Serial.println(state);

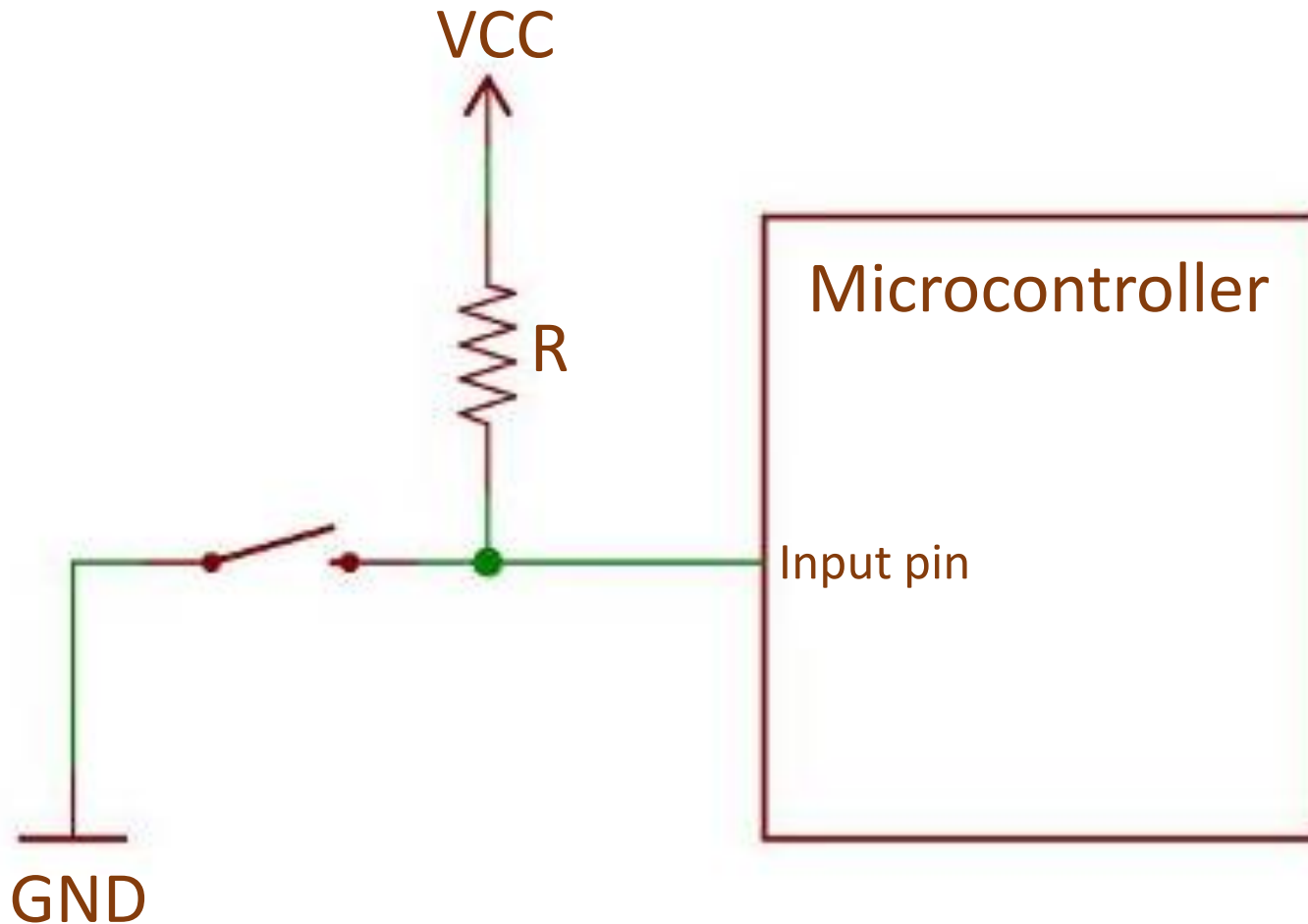
    if (state==HIGH)
    {
        digitalWrite(13, HIGH);
    }
    else
    {
        digitalWrite(13, LOW);
    }
}
```

The state of the pin is unreliable if the pin is floating.



Pull up resistor

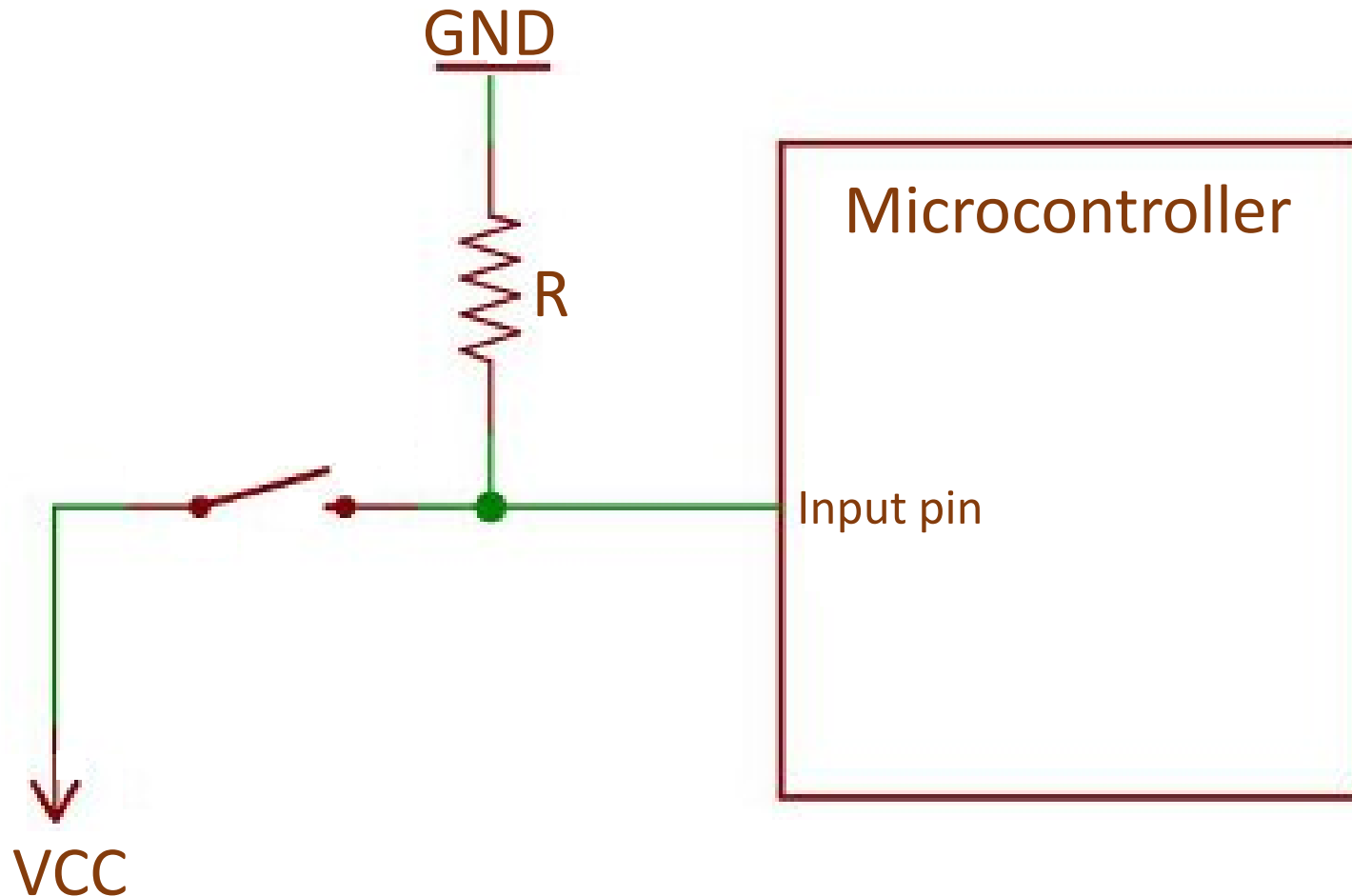
In this set up, the input pin is connected to either VCC or GND regardless of the status of the switch.



- A pull up resistor pulls the input pin to the HIGH level when the switch is open.
- A pull up resistor makes the **default** state of the input pin **HIGH**.

Pull down resistor

In this set up, the input pin is connected to either VCC or GND regardless of the status of the switch.



- A pull down resistor pulls down the input pin to the LOW level when the switch is open.
- A pull down resistor makes the **default** state of the input pin **LOW**.

Internal pull-up resistor

ATmega328p has internal pull-up resistors in GPIOs which can be activated by the command `pinMode(x, INPUT_PULLUP)`

```
void setup()
{
    pinMode(0, INPUT_PULLUP);
}

void loop()
{
    if (digitalRead(0)==HIGH)
    {
        //HIGH (Default state)
    }
    else
    {
        //LOW (Button is pressed)
    }
}
```

GPIO

- Recall that GPIO pins are divided into port B, port C, and port D.
- Each port has 3 control registers

DDRB

PORTB

PINB

DDRC

PORTC

PINC

DDRD

PORTD

PIND

- DDRX controls the data direction (0 for input and 1 for output)
- PORTX is for output
- PINX is for input

Port B registers

PORTB - THE PORT B DATA REGISTER

Bit	7	6	5	4	3	2	1	0
0x25	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- PORTB7-0: GPIO data value stored in bit n .

DDRB - THE PORT B DATA DIRECTION REGISTER

Bit	7	6	5	4	3	2	1	0
0x24	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- DDRB7-0: selects the direction of pin n . If $DDRBn$ is written '1', then $PORTBn$ is configured as an output pin. If $DDRBn$ is written '0', then $PORTBn$ is configured as an input pin.

PINB - THE PORT B INPUT PINS ADDRESS

Bit	7	6	5	4	3	2	1	0
0x23	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
Read/Write	R	R	R	R	R	R	R	R
Default	-	-	-	-	-	-	-	-

- PINB7-0: logic value present on external pin n .

Port C registers

PORTC - THE PORT C DATA REGISTER

Bit	7	6	5	4	3	2	1	0
0x28	-	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- PORTC6-0: GPIO data value stored in bit n .

DDRC - THE PORT C DATA DIRECTION REGISTER

Bit	7	6	5	4	3	2	1	0
0x27	-	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- DDRC6-0: selects the direction of pin n . If DDRC n is written '1', then PORTC n is configured as an output pin. If DDRC n is written '0', then PORTC n is configured as an input pin.

PINC - THE PORT C INPUT PINS ADDRESS

Bit	7	6	5	4	3	2	1	0
0x26	-	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
Read/Write	R	R	R	R	R	R	R	R
Default	0	-	-	-	-	-	-	-

- PINC6-0: logic value present on external pin n .

Port D registers

PORTD - THE PORT D DATA REGISTER

Bit	7	6	5	4	3	2	1	0
0x2B	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- PORTD7-0: GPIO data value stored in bit n .

DDRD - THE PORT D DATA DIRECTION REGISTER

Bit	7	6	5	4	3	2	1	0
0x2A	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- DDRD7-0: selects the direction of pin n . If $DDRDn$ is written '1', then $PORTDn$ is configured as an output pin. If $DDRDn$ is written '0', then $PORTDn$ is configured as an input pin.

PIND - THE PORT D INPUT PINS ADDRESS

Bit	7	6	5	4	3	2	1	0
0x29	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
Read/Write	R	R	R	R	R	R	R	R
Default	-	-	-	-	-	-	-	-

- PIND7-0: logic value present on external pin n .

Example 1: digital write HIGH

Determine the values of the control registers if you want to write PB1 HIGH.

Set

DDRB = XXXX XX1X

PORTB = XXXX XX1X

Example 2: digital write LOW

Determine the values of the control registers if you want to write PD5 LOW.

Set

DDRD = XX1X XXXX

PORTD = XX0X XXXX

Example 3: digital input without pull up

Determine the values of the control registers if you want to read PC1.

Set

DDRC = XXXX XX0X

PORTC = XXXX XX0X

And then read the content of PINC

Example 4: digital input with pull up

Determine the values of the control registers if you want to read PC1 (with internal pull up)

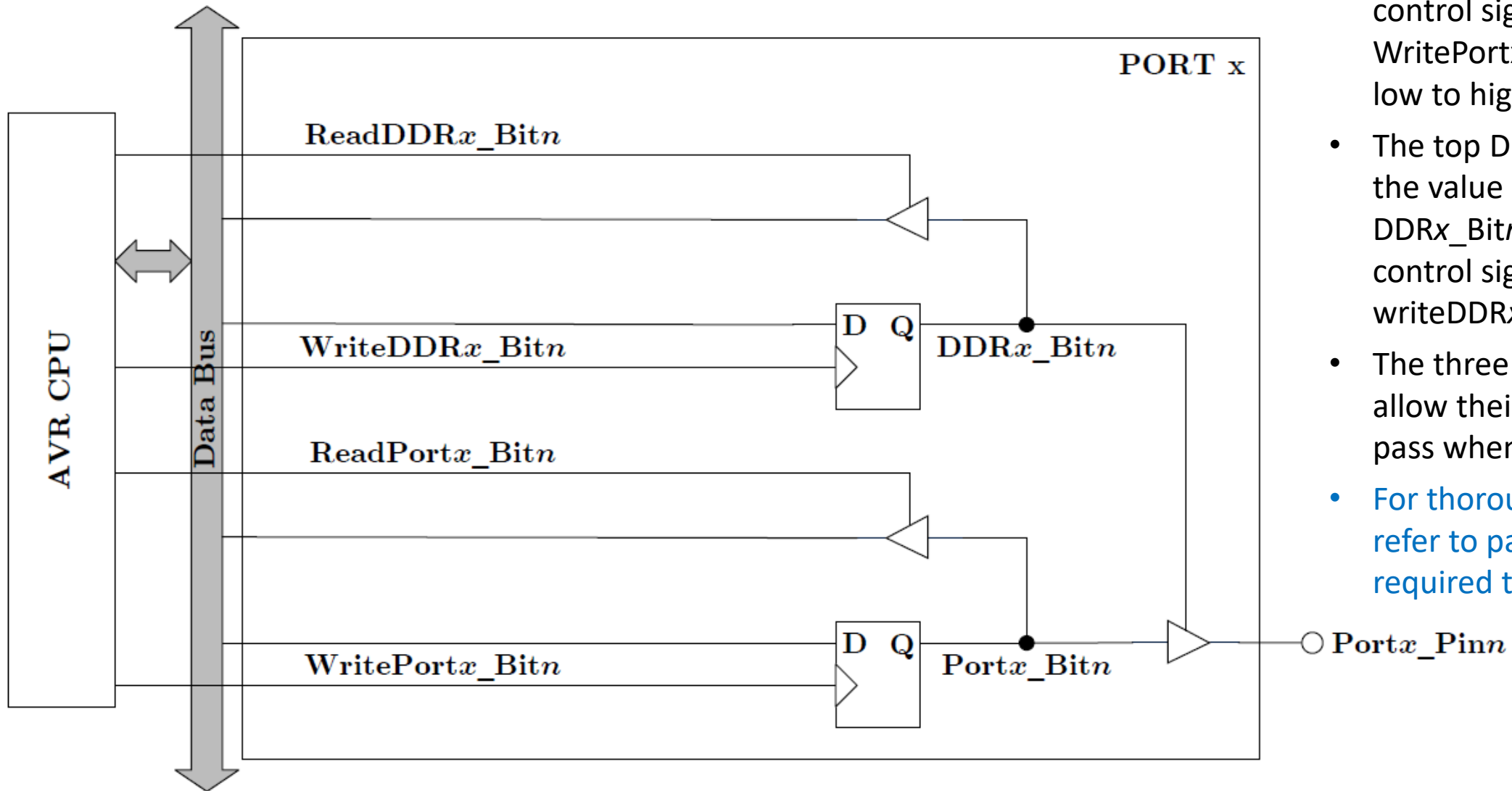
Set

DDRC = XXXX XX0X

PORTC = XXXX XX1X

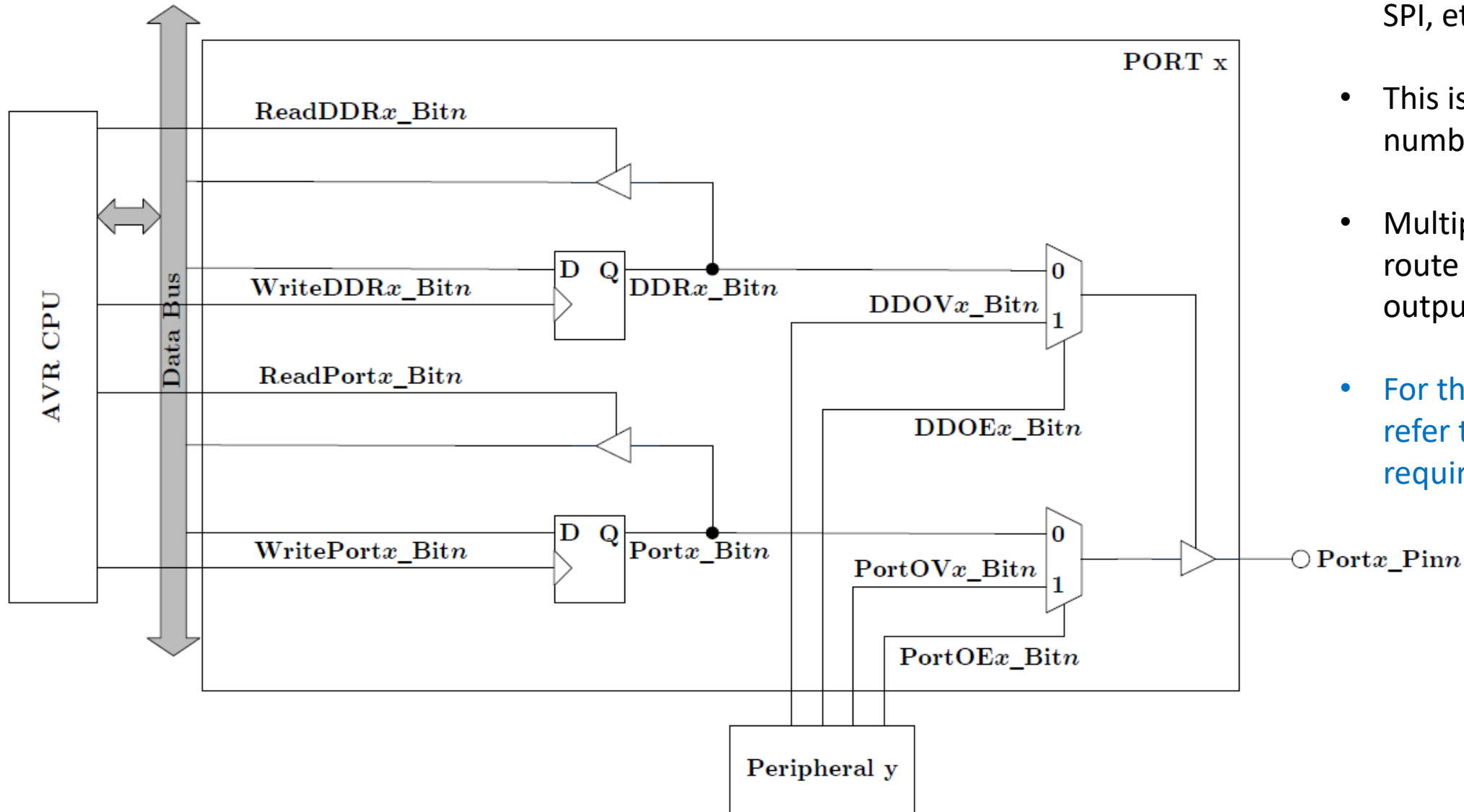
And then read the content of PINC

Schematic of single port pin



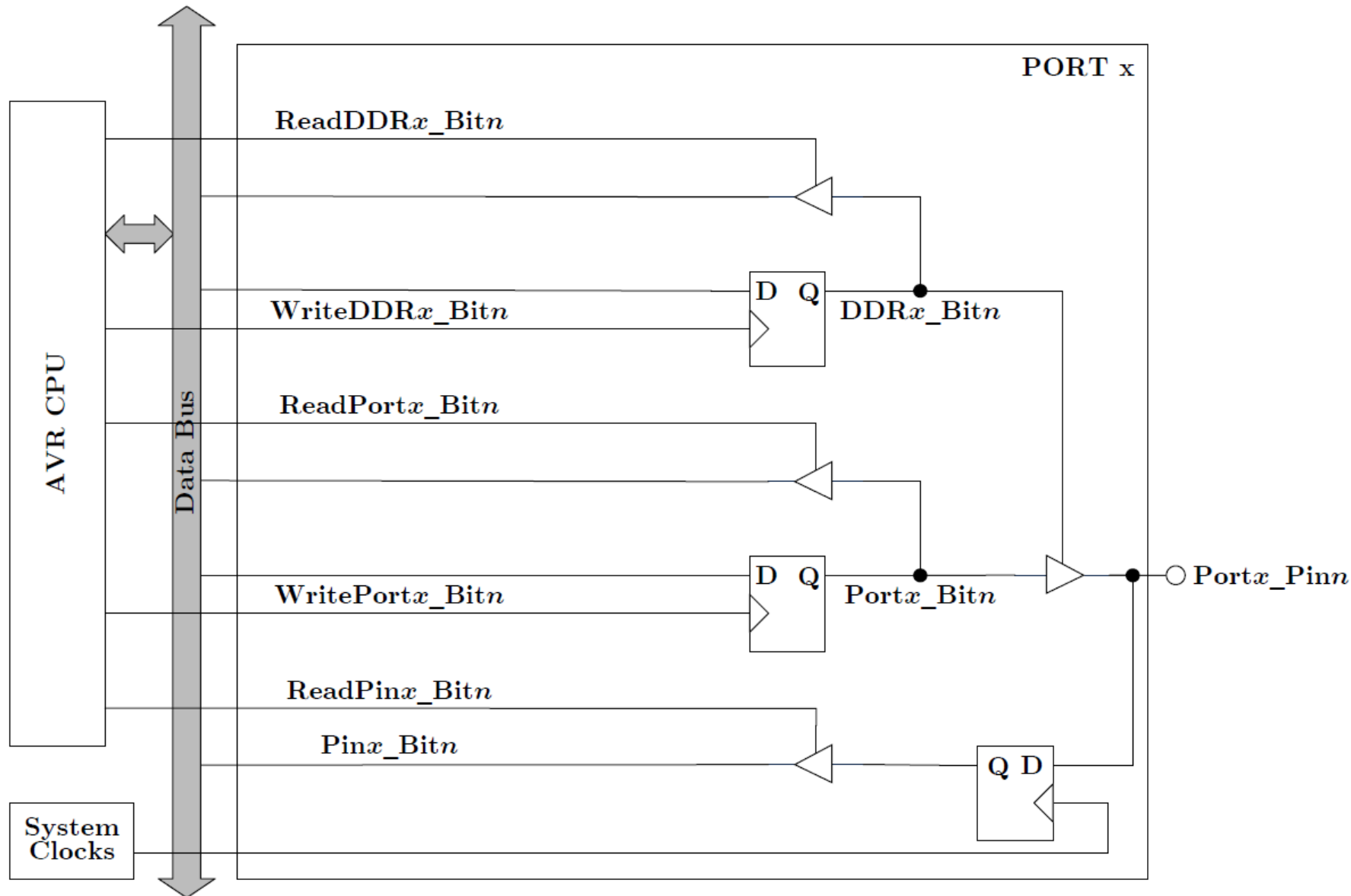
- The bottom D flip-flop stores whatever value is on line Port_x_Bitn when the control signal WritePort_x_Bitn goes from low to high.
- The top D flip-flop stores the value present on DDR_x_Bitn when the control signal writeDDR_x_Bitn occurs.
- The three tri-state buffers allow their input signals to pass when enabled
- For thorough explanation, refer to page 100 of the required textbook.

Schematic of single port pin with mux



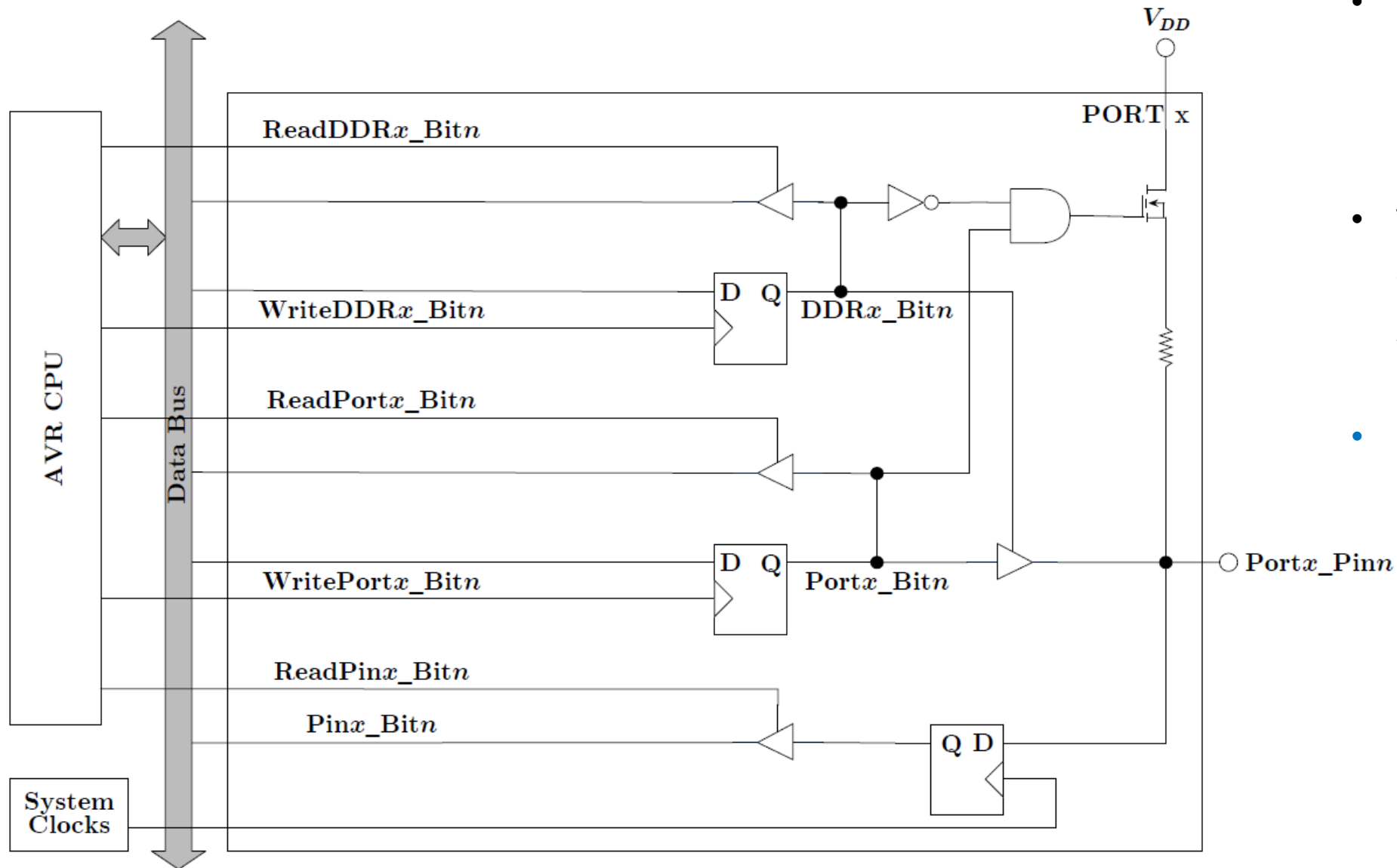
- GPIO pins have other functionality such as ADC, SPI, etc.
- This is to minimize the total number of pins.
- Multiplexers are used to route selected input to the output pin
- For thorough explanation, refer to page 101 of the required textbook.

Schematic of single port pin with input functionality



- The schematic is almost the same as the one two slides earlier.
- Now there is an extra D flip-flop whose input is directly connected to the GPIO pin.
- If the DDR bit is low, the data at the GPIO pin will be routed to input of the bottom tri-state buffer, which lets the signal pass to the data bus when ReadPin command is activated.
- For thorough explanation, refer to page 103 of the required textbook.

Schematic of single port pin with input functionality + pull up register



- There is an AND gate that has inputs of the DDR x _Bit n output via an inverter and the Port x _Bit n output.
- The HIGH output of the AND gate makes the transistor acts like a short circuit, connecting VDD to the GPIO pin.
- For thorough explanation, refer to page 105 of the required textbook.

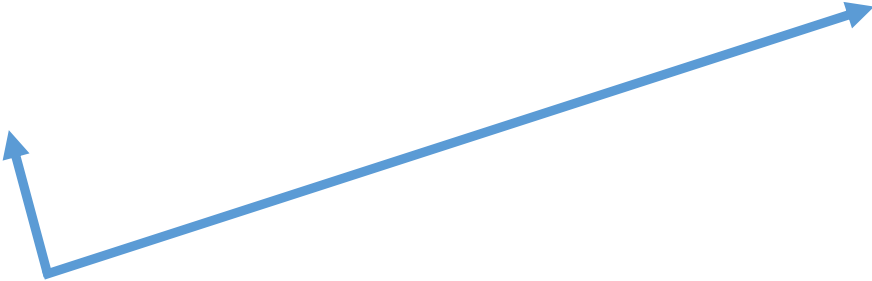
Writing digital output

```
void setup()
{
    unsigned char *dir = (unsigned char*) 0x24;
    *dir = 0b00100000;          //or *dir = 32      or *dir = 0x20
}

void loop()
{
    unsigned char *port = (unsigned char*) 0x25;
    *port = 0b00100000;         //or *dir = 32      or *dir = 0x20
    delay(1000);
    *port = 0;
    delay(1000);
}
```

```
void setup()
{
    DDRB = 32;
}

void loop()
{
    PORTB = 32;
    delay(1000);
    PORTB = 0;
    delay(1000);
}
```



These two pieces of code achieve the same thing.
They blink the built-in LED on PB5 with a period of 2s.

The Arduino library uses AVR library which predefines variables that have the same names as the registers. For example, DDRB is a predefined variable with a memory address of 0x24.

Reading digital input

```
void setup()
{
    unsigned char *dir = (unsigned char*) 0x24;
    unsigned char *writer = (unsigned char*) 0x25;
    unsigned char *reader = (unsigned char*) 0x23;

    *dir = 0b00100000;           //or *dir = 32      or *dir = 0x20

    for (;;)
    {
        if (((*reader) & 0b00010000) != 0)
        {
            *writer = 0b00100000;
        }
        else
        {
            *writer = 0;
        }
    }
}
```

*dir points to DDRB
*writer points to PORTB
*reader points to PINB

- This program constantly checks the status of digital input on PB4.
- If the digital input is HIGH, it turns on the built-in LED on PB5.
- Note, if an infinite loop is placed within the setup function, the loop function becomes unnecessary.

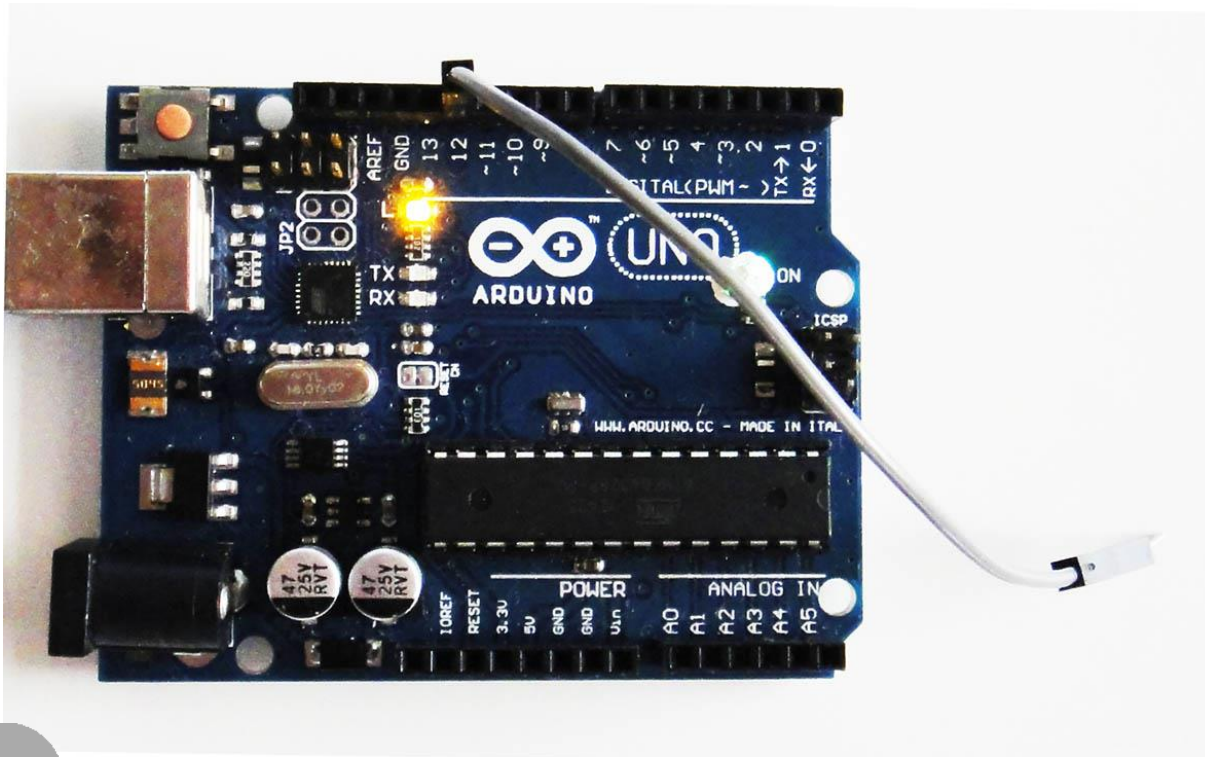
Reading digital input with internal pull up

```
void setup()
{
    unsigned char *dir = (unsigned char*) 0x24;
    unsigned char *writer = (unsigned char*) 0x25;
    unsigned char *reader = (unsigned char*) 0x23;

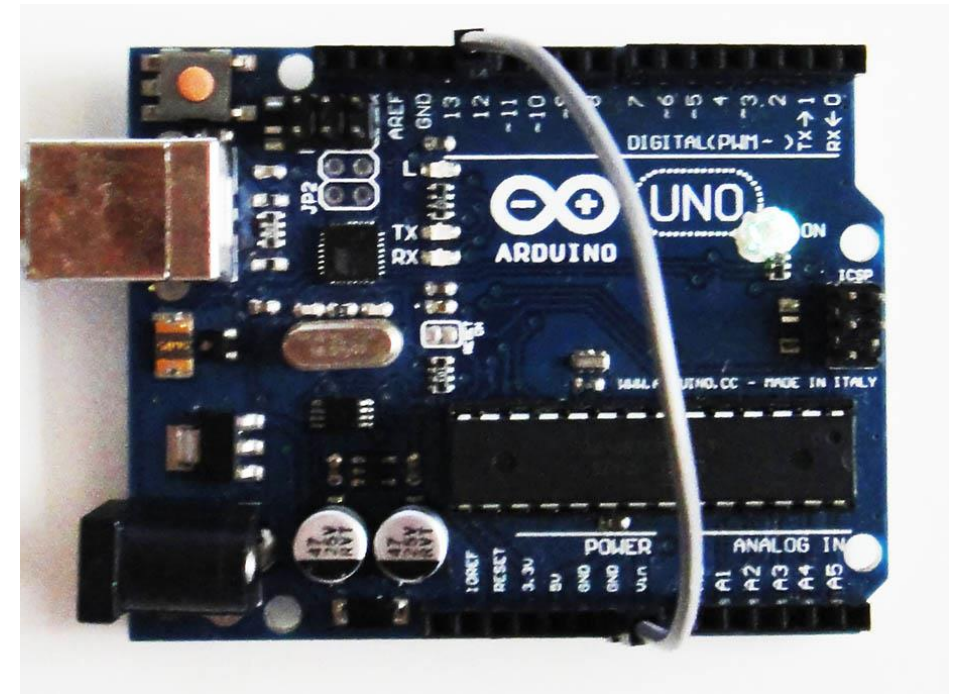
    *dir = 0b00100000;           //or *dir = 32      or *dir = 0x20
    *writer = 0b00110000;

    for (;;)
    {
        if (((*reader) & 0b00010000) != 0)
        {
            *writer = 0b00110000;    //or      *writer /= 0b00100000;
        }
        else
        {
            *writer = 0b00010000;    //or      *writer &= 0b11011111;
        }
    }
}
```

The built-in LED is ON by default
(because of pull up resistor)



When PB4 is connected to GND,
the LED goes OFF



How fast is the Arduino Library?

Benchmarking of digital output (Arduino Library)

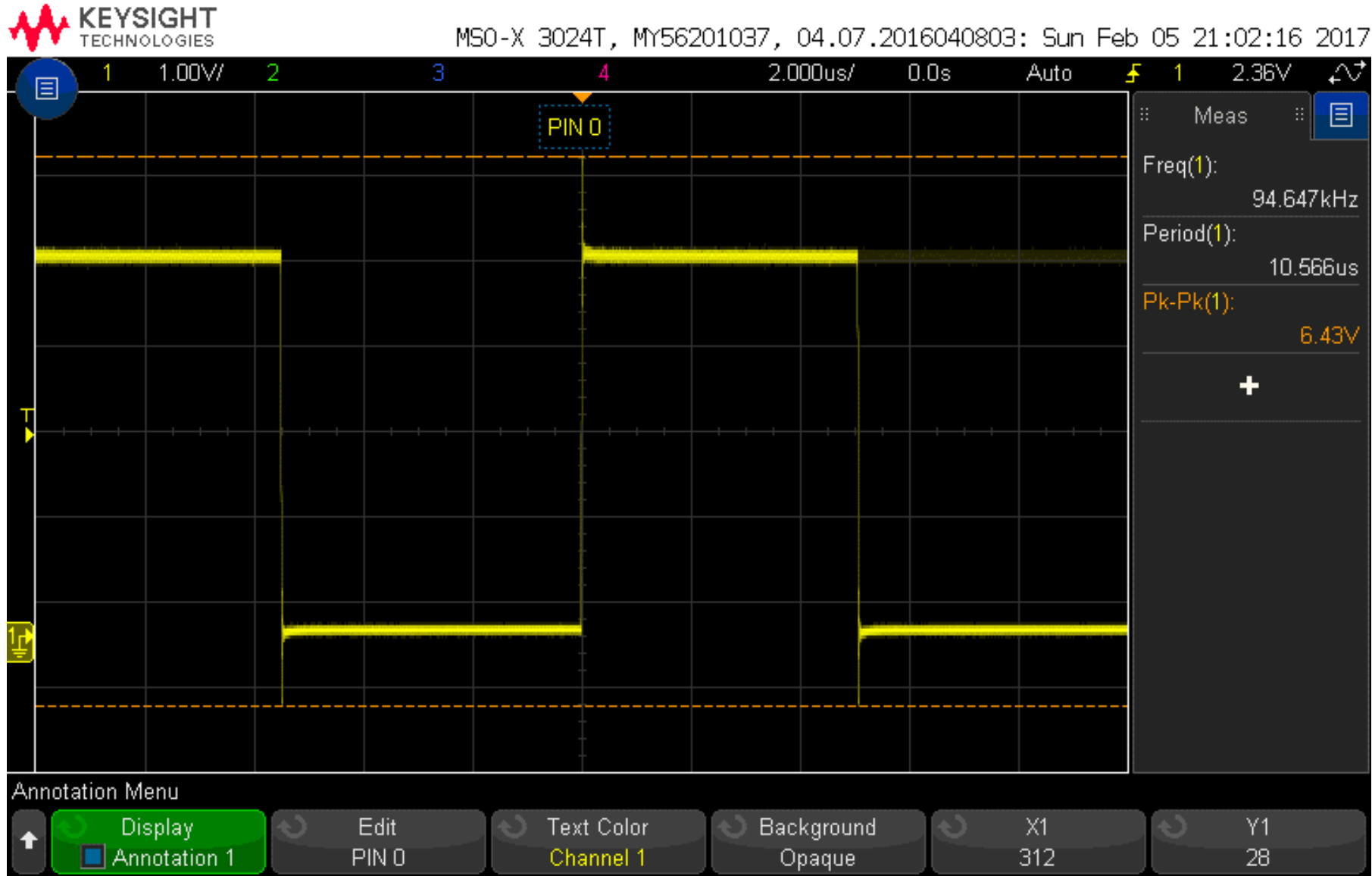
```
void setup()
{
    pinMode(0, OUTPUT);

    for (;;)
    {
        digitalWrite(0, HIGH);
        digitalWrite(0, LOW);
    }
}

void loop()
{
}
```

- This code flips between HIGH state and digital state for a particular pin without any delay.

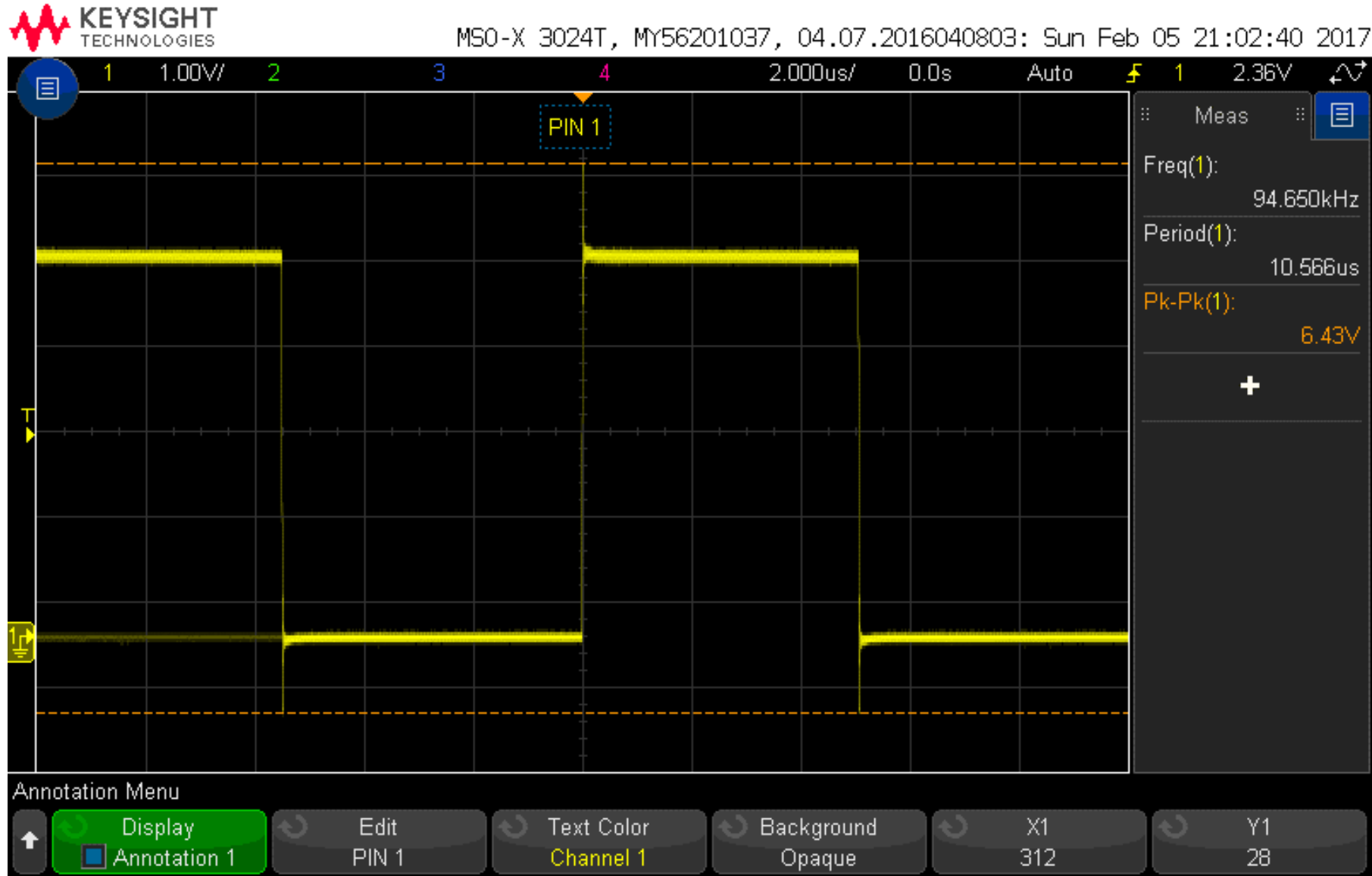
PDO output



Frequency:

94.647 kHz

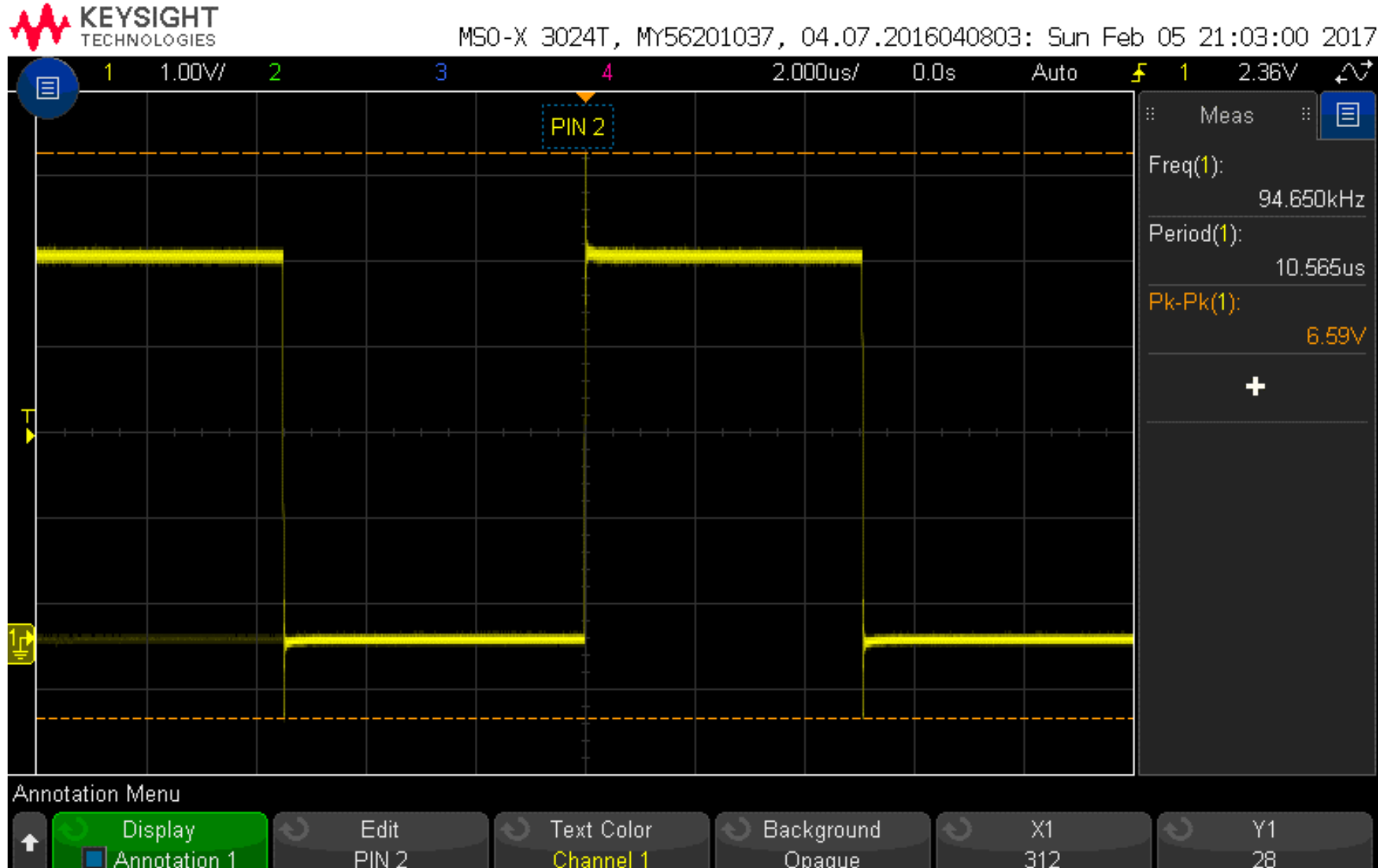
PD1 output



Frequency:

94.650 kHz

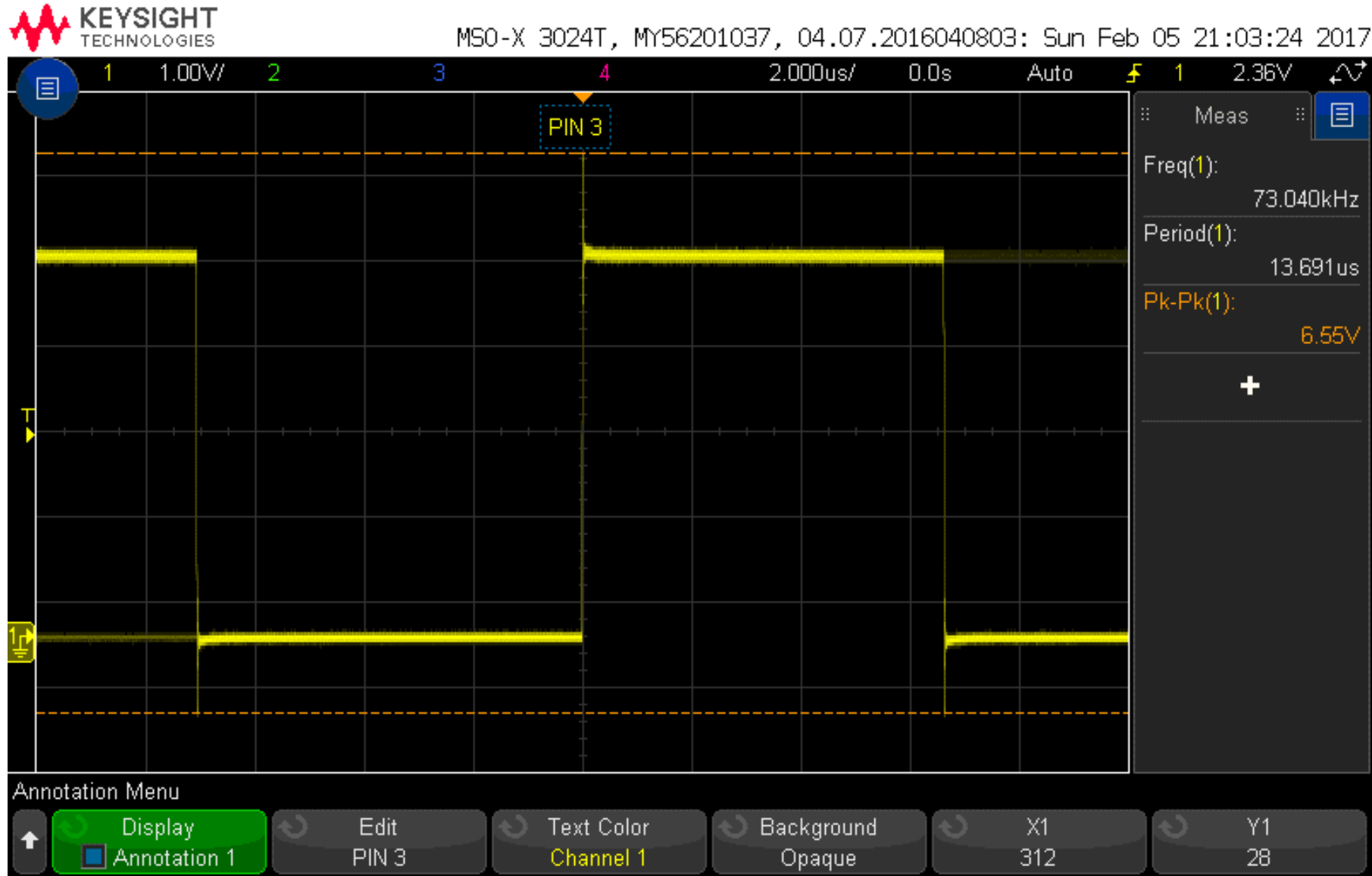
PD2 output



Frequency:

94.650 kHz

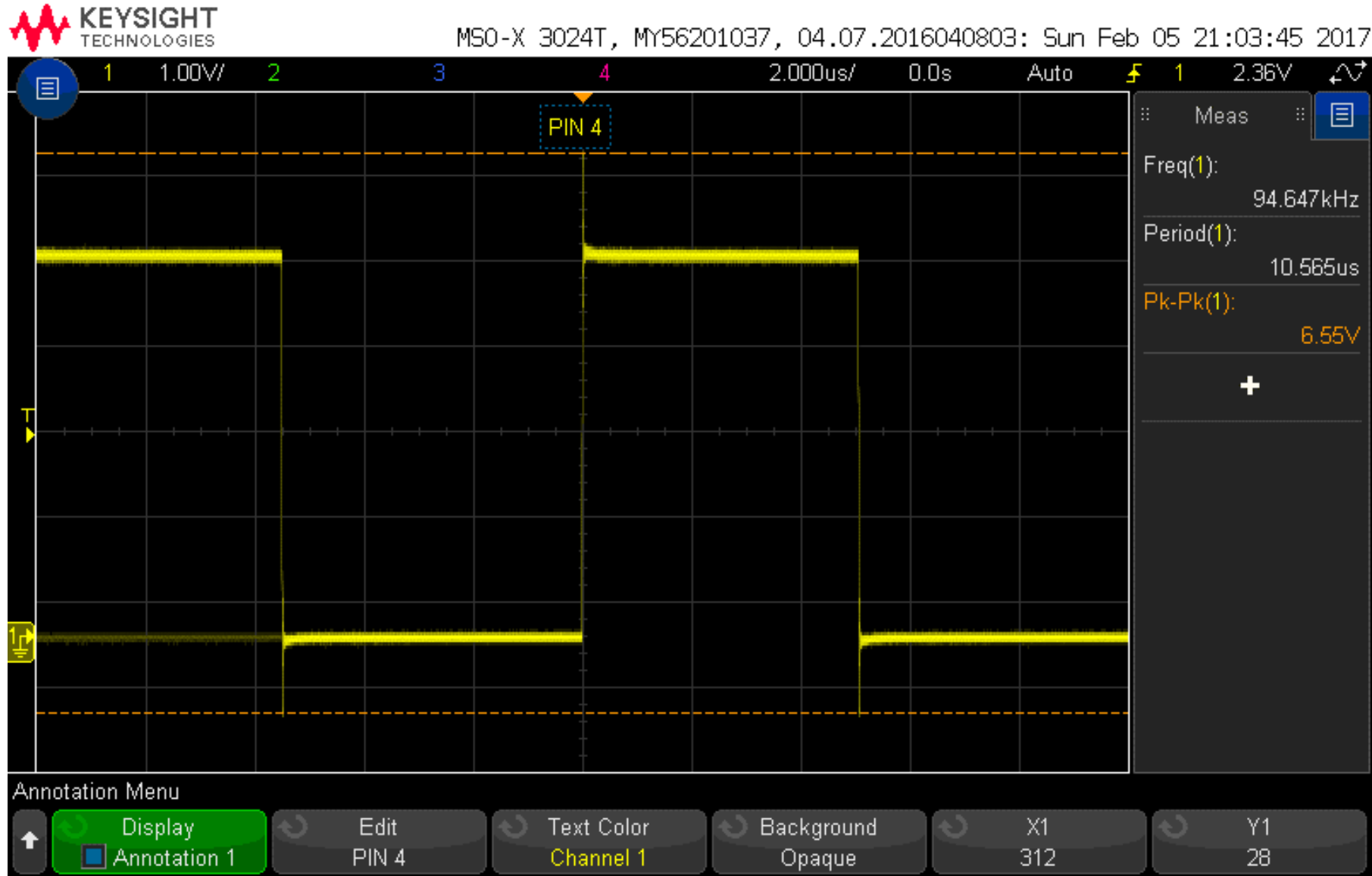
PD3 output



Frequency:

73.040 kHz

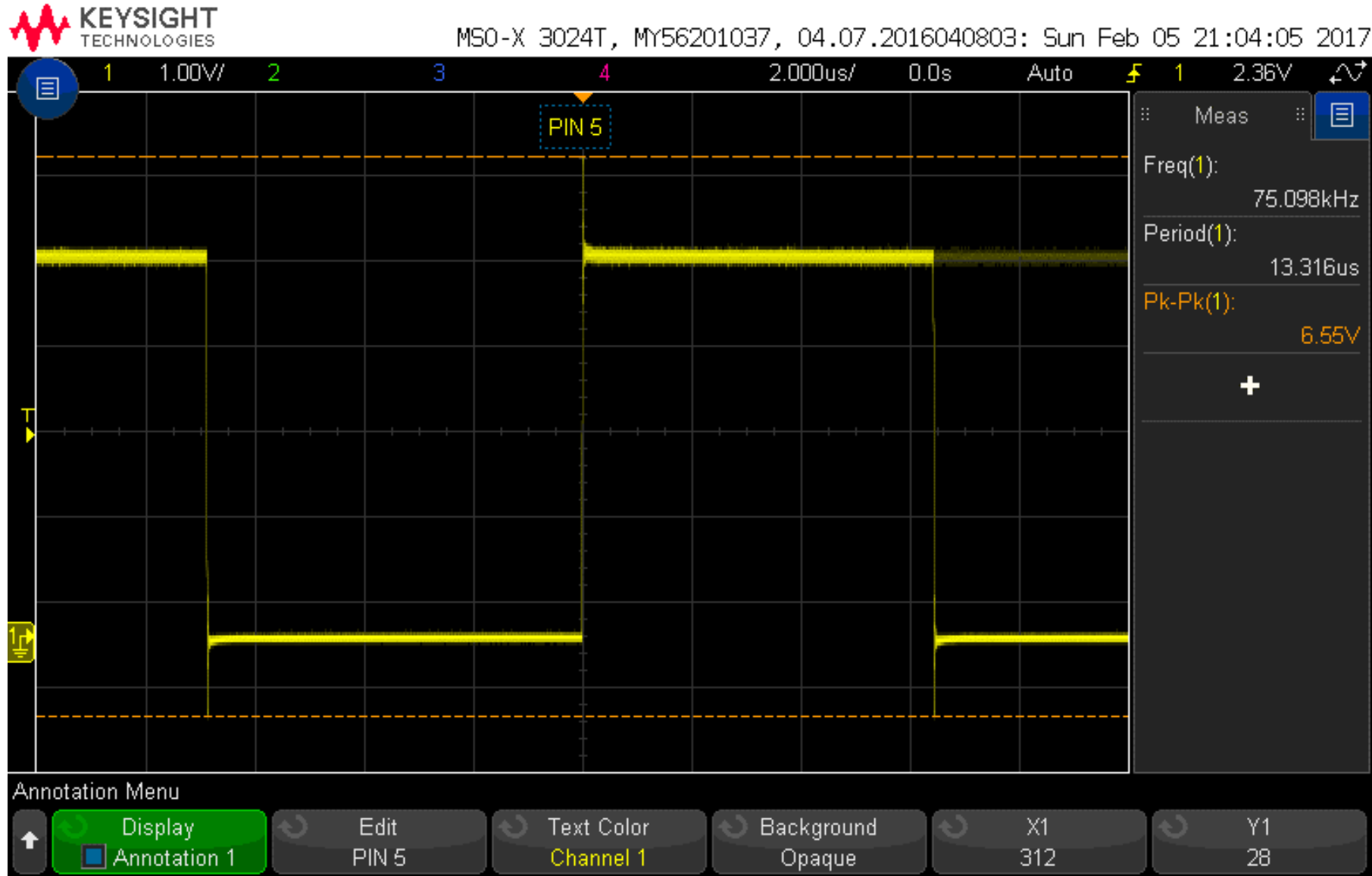
PD4 output



Frequency:

94.647 kHz

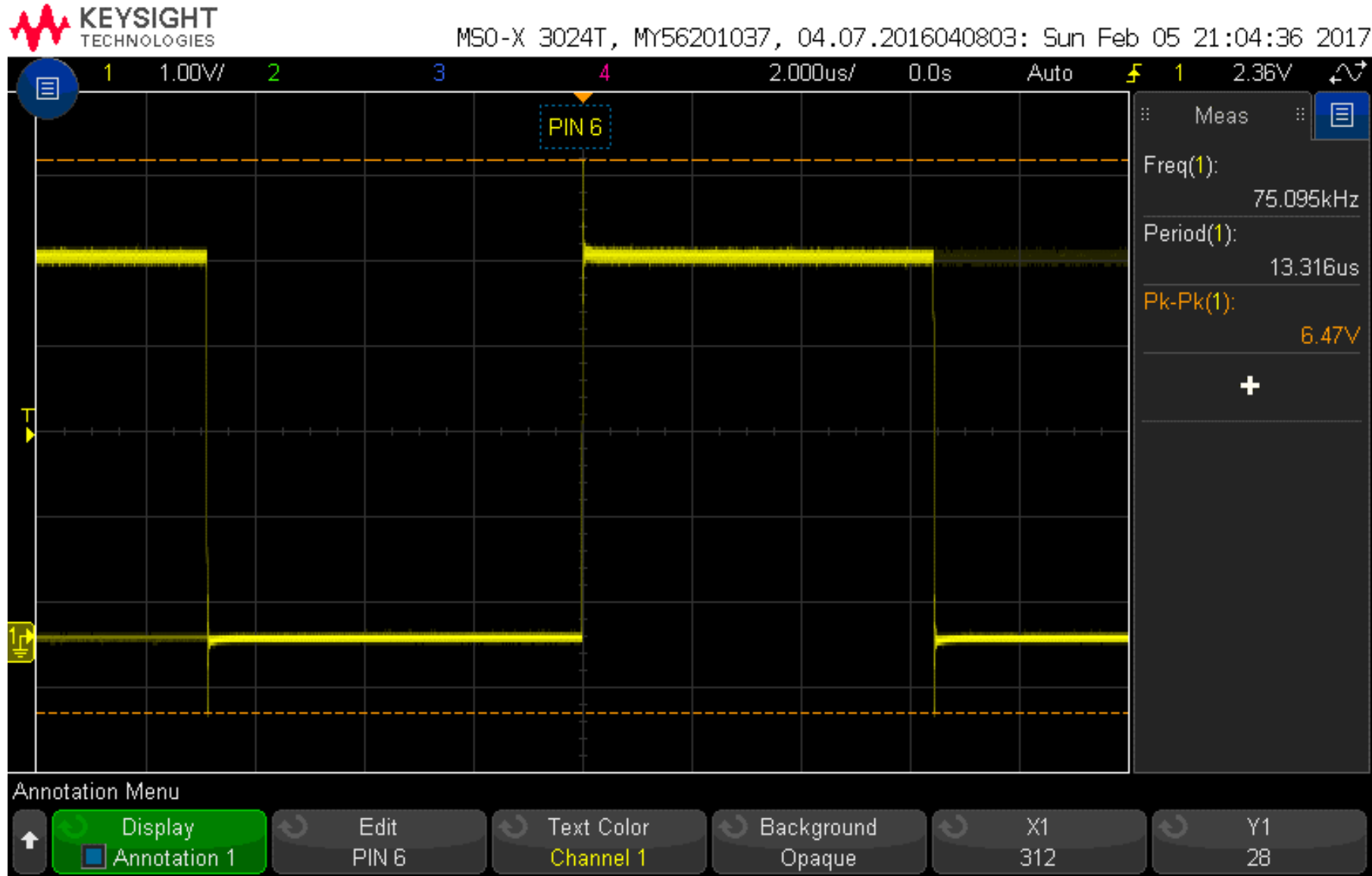
PD5 output



Frequency:

75.098 kHz

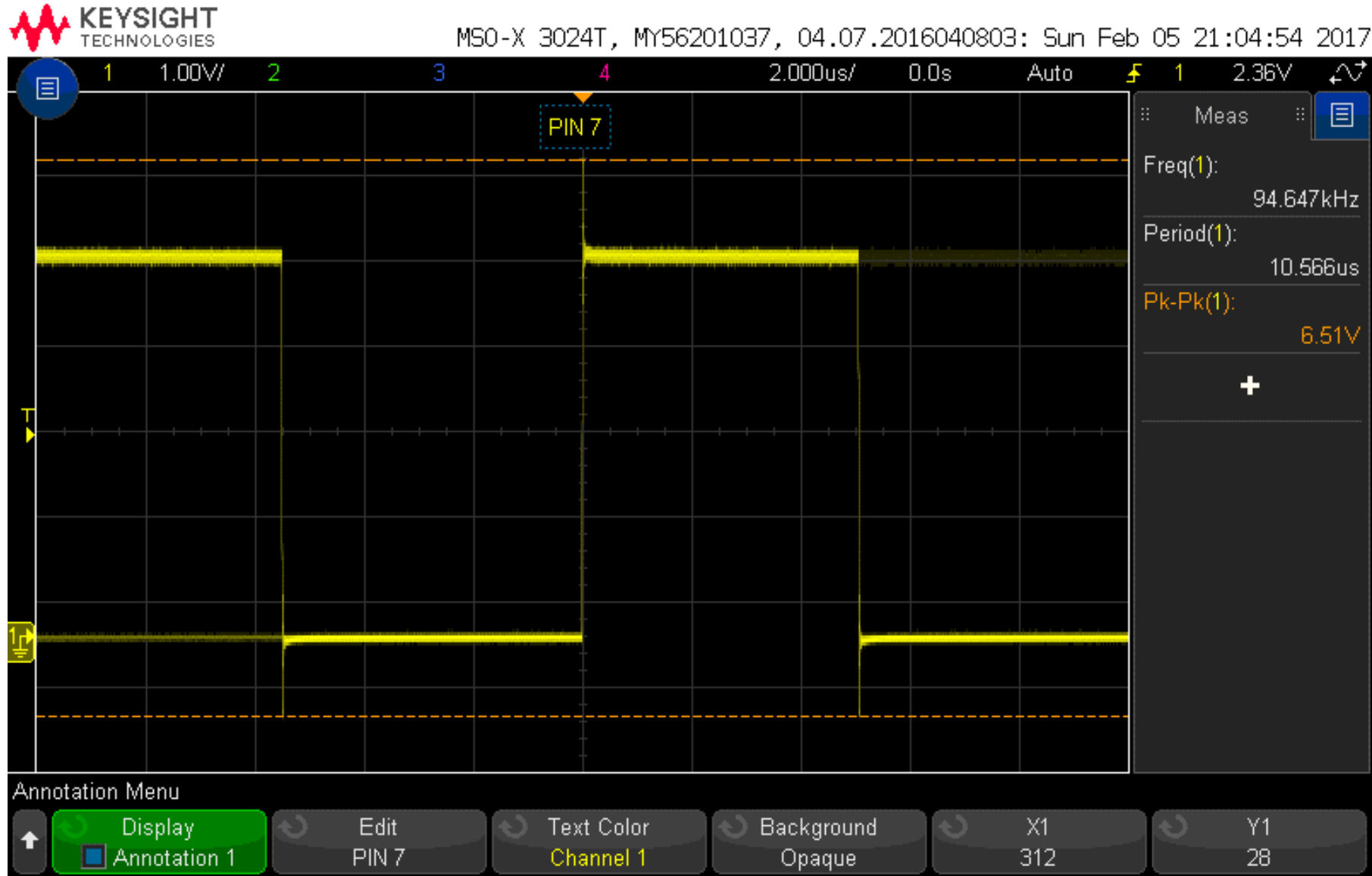
PD6 output



Frequency:

75.095 kHz

PD7 output



Frequency:

94.647 kHz

Benchmarking of digital output (normal C code)

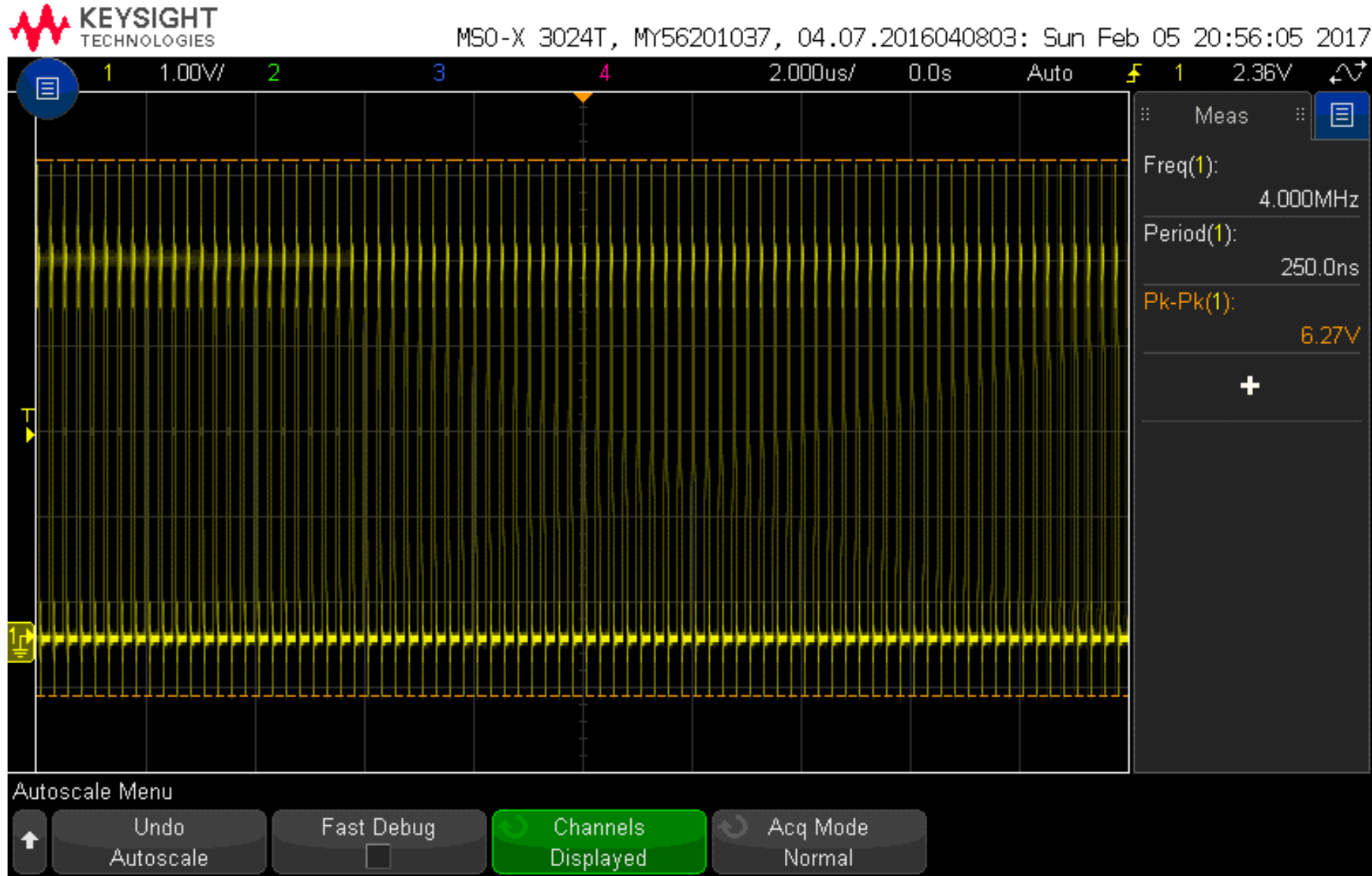
```
void setup()
{
    unsigned char *dir = (unsigned char*) 0x2A;
    *dir = 255;

    unsigned char *port = (unsigned char*) 0x2B;
    for (;;)
    {
        *port = 255;
        *port = 0;
    }
}

void loop()
{
}
```

- This code flips between HIGH state and digital state for all the pins without any delay.

Performance of GPIO using normal C doe



Frequency:

4.000 MHz

for all the port
D pins

Benchmarking of digital input (Arduino Library)

```
void setup()
{
    unsigned long begin_time = micros();

    // Benchmarking starts here
    for (int i=0; i<10000; i++)
    {
        digitalRead(1);
    }
    // Benchmarking ends here

    unsigned long duration = micros() - begin_time;

    Serial.begin(9600);
    Serial.println(duration);
}
```

This code performs digitalRead 10,000 times and then print the total elapsed times in μs to the serial port.

Results

Pin	Time for 10000 digitalRead	Frequency
PD0	40872 μs	244.666 kHz
PD1	40872 μs	244.666 kHz
PD2	54708 μs	237.349 kHz
PD3	42132 μs	182.789 kHz
PD4	52820 μs	237.349 kHz
PD5	54708 μs	189.332 kHz
PD6	52820 μs	189.332 kHz
PD7	42132 μs	237.349 kHz

Benchmarking of digital input (Normal C code)

```
void setup()
{
    unsigned long begin_time = micros();
    unsigned char *reader = (unsigned char*) 0x23;

    // Benchmarking starts here
    for (int i=0; i<10000; i++)
    {
        unsigned char value = *reader;
    }
    // Benchmarking ends here

    unsigned long duration = micros() - begin_time;

    Serial.begin(9600);
    Serial.println(duration);
}
```

This code performs reads the
PIND register 10,000 times
and then print the total
elapsed times in μs to the
serial port.

Results

Total time taken = $3148 \mu s$

Frequency = 3.177 MHz

digitalRead function of Arduino Library

```
int digitalRead(uint8_t pin)
{
    uint8_t timer = digitalPinToTimer(pin);
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);

    if (port == NOT_A_PIN) return LOW;

    // If the pin that support PWM output, we need to turn it off
    // before getting a digital reading.
    if (timer != NOT_ON_TIMER) turnOffPWM(timer);

    if (*portInputRegister(port) & bit) return HIGH;
    return LOW;
}
```

digitalWrite function of Arduino Library

```
void digitalWrite(uint8_t pin, uint8_t val)
{
    uint8_t timer = digitalPinToTimer(pin);
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    volatile uint8_t *out;

    if (port == NOT_A_PIN) return;

    // If the pin that support PWM output, we need to turn it off
    // before doing a digital write.
    if (timer != NOT_ON_TIMER) turnOffPWM(timer);

    out = portOutputRegister(port);

    uint8_t oldSREG = SREG;
    cli();

    if (val == LOW) {
        *out &= ~bit;
    } else {
        *out |= bit;
    }

    SREG = oldSREG;
}
```

pinMode function of Arduino Library

```
void pinMode(uint8_t pin, uint8_t mode)
{
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    volatile uint8_t *reg, *out;

    if (port == NOT_A_PIN) return;

    // JWS: can I let the optimizer do this?
    reg = portModeRegister(port);
    out = portOutputRegister(port);

    if (mode == INPUT) {
        uint8_t oldSREG = SREG;
        cli();
        *reg &= ~bit;
        *out &= ~bit;
        SREG = oldSREG;
    } else if (mode == INPUT_PULLUP) {
        uint8_t oldSREG = SREG;
        cli();
        *reg &= ~bit;
        *out |= bit;
        SREG = oldSREG;
    } else {
        uint8_t oldSREG = SREG;
        cli();
        *reg |= bit;
        SREG = oldSREG;
    }
}
```

millis function of Arduino Library

```
unsigned long millis()
{
    unsigned long m;
    uint8_t oldSREG = SREG;

    // disable interrupts while we read timer0_millis or we might get an
    // inconsistent value (e.g. in the middle of a write to timer0_millis)
    cli();
    m = timer0_millis;
    SREG = oldSREG;

    return m;
}
```

millis() will roll back after approximately 49.7 days (4,294,967,296 milliseconds)

It uses timer interrupt.

micros function of Arduino Library

micros() will roll back after approximately 71.6 minutes (4,294,967,296 microseconds)

```
unsigned long micros()
{
    unsigned long m;
    uint8_t oldSREG = SREG, t;
    cli();
    m = timer0_overflow_count;
    #if defined(TCNT0)
        t = TCNT0;
    #elif defined(TCNT0L)
        t = TCNT0L;
    #else
        #error TIMER 0 not defined
    #endif

    #ifdef TIFR0
        if ((TIFR0 & _BV(TOV0)) && (t < 255))
            m++;
    #else
        if ((TIFR & _BV(TOV0)) && (t < 255))
            m++;
    #endif

    SREG = oldSREG;
    return ((m << 8) + t) * (64 / clockCyclesPerMicrosecond());
}
```

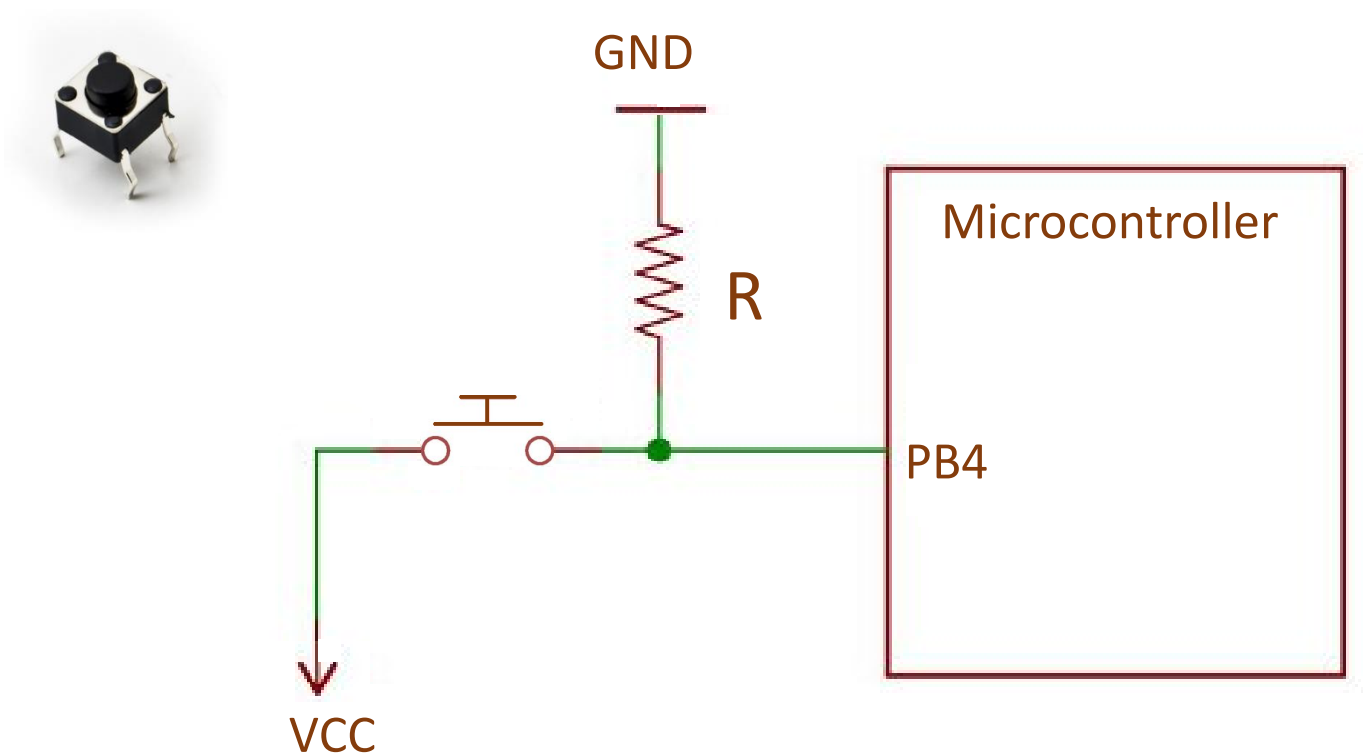
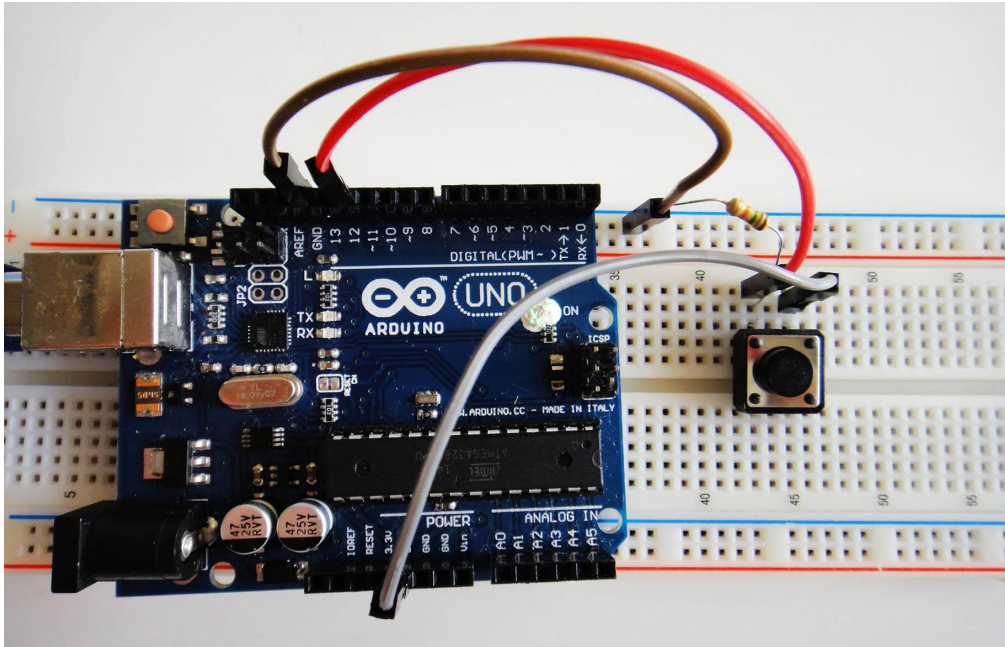

delay function of Arduino Library

```
void delay(unsigned long ms)
{
    uint32_t start = micros();

    while (ms > 0) {
        yield();
        while ( ms > 0 && (micros() - start) >= 1000) {
            ms--;
            start += 1000;
        }
    }
}
```

Example 5

A tactile button (push-to-make switch) is connected to PB5 of ATmega328p via a pull down resistor. The microcontroller needs to keep track of how many times the button has been pressed (LOW to HIGH). Whenever a button has been pressed, the microcontroller needs to send out the number (of times the button has been pressed) through the serial port. Program the microcontroller to accomplish this.

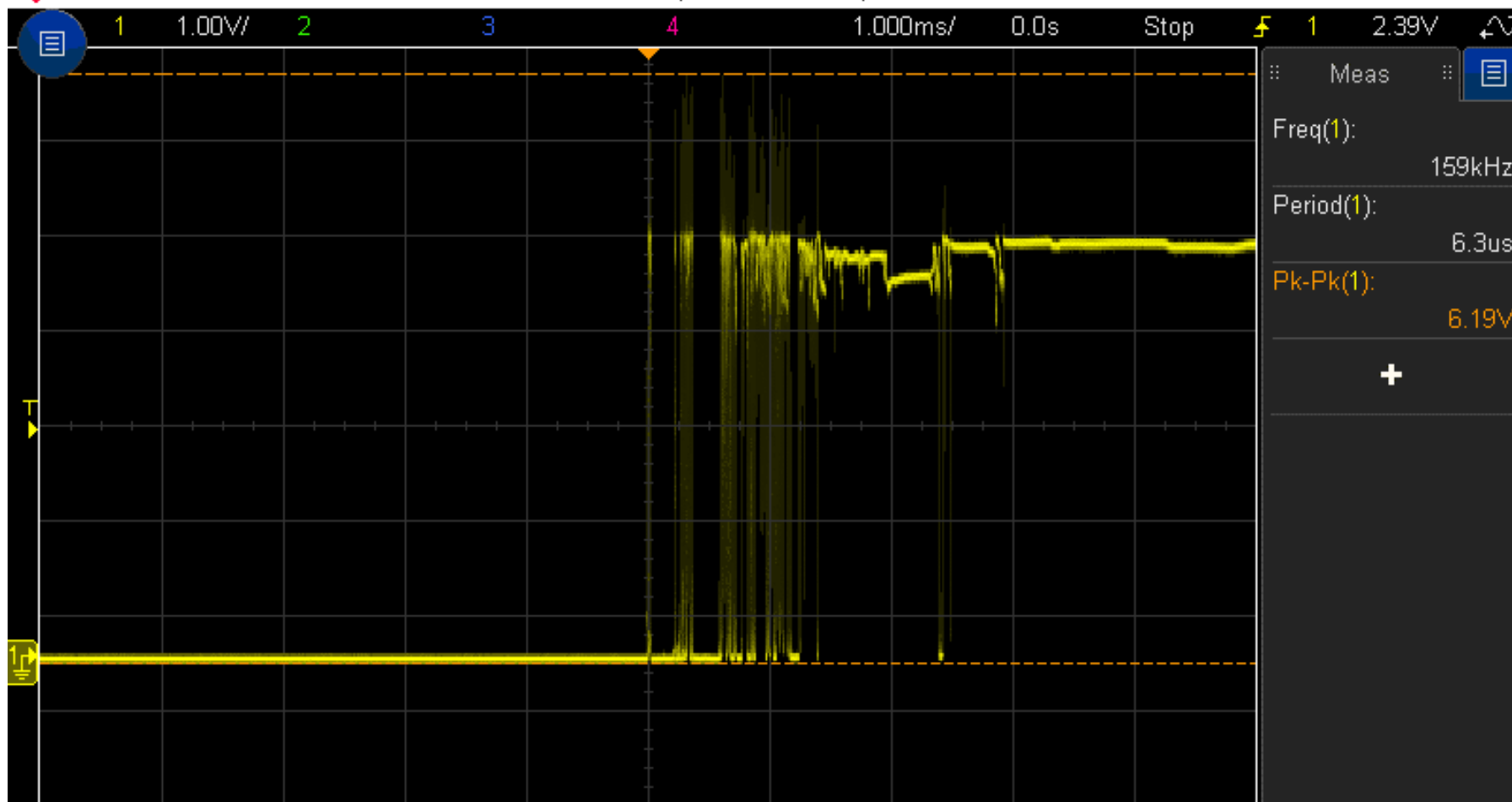


```
unsigned int count;
bool previous;
unsigned char *reader = (unsigned char*) 0x23;

int main()                //To run on Arduino, just change this function to void setup()
{
    Serial.begin(9600);    //For now, we are still using the Serial library

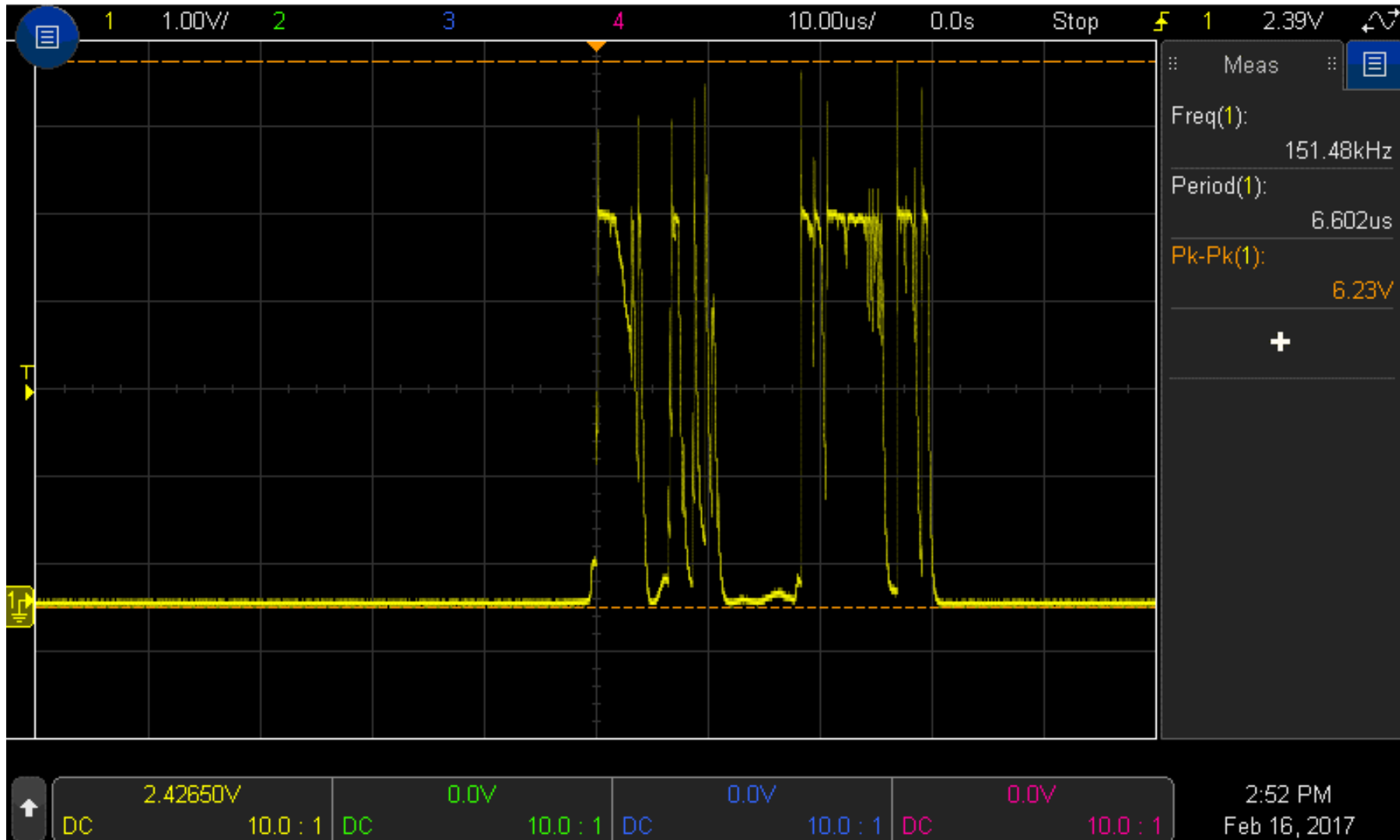
    for (;;)
    {
        bool current = (((*reader) & 0b00010000) != 0);
        if (current && !previous)    //If the pin is currently HIGH and previously LOW
        {
            count++;
            Serial.println(count);
        }
        previous = current;
    }
}
```





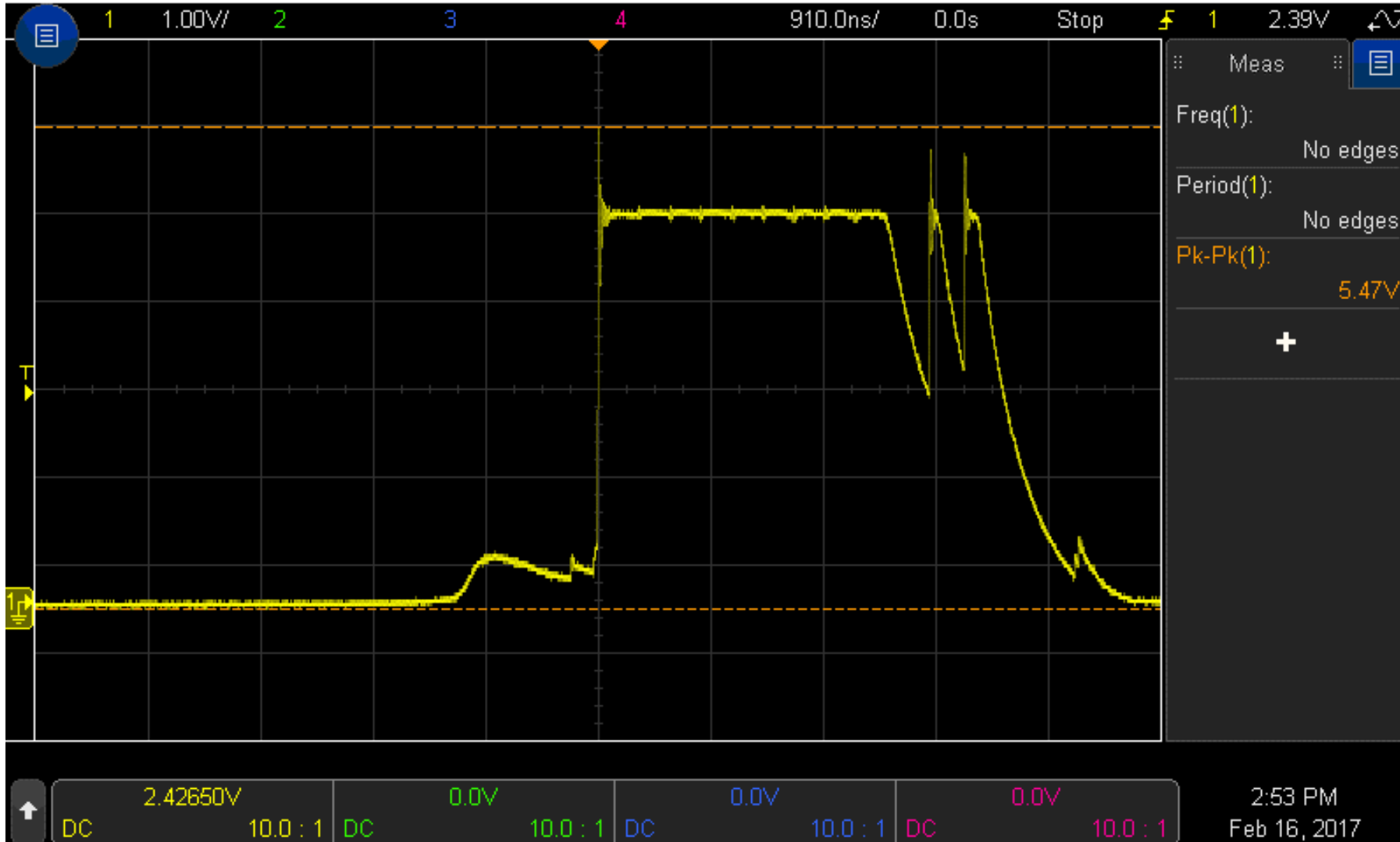
Typical
waveform of a
button press

(1 ms/div)



Typical
waveform of a
button press

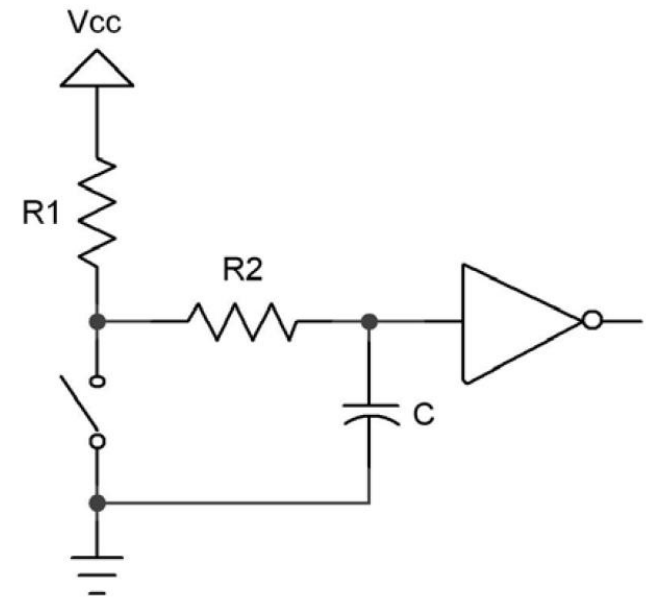
(10 μ s/div)



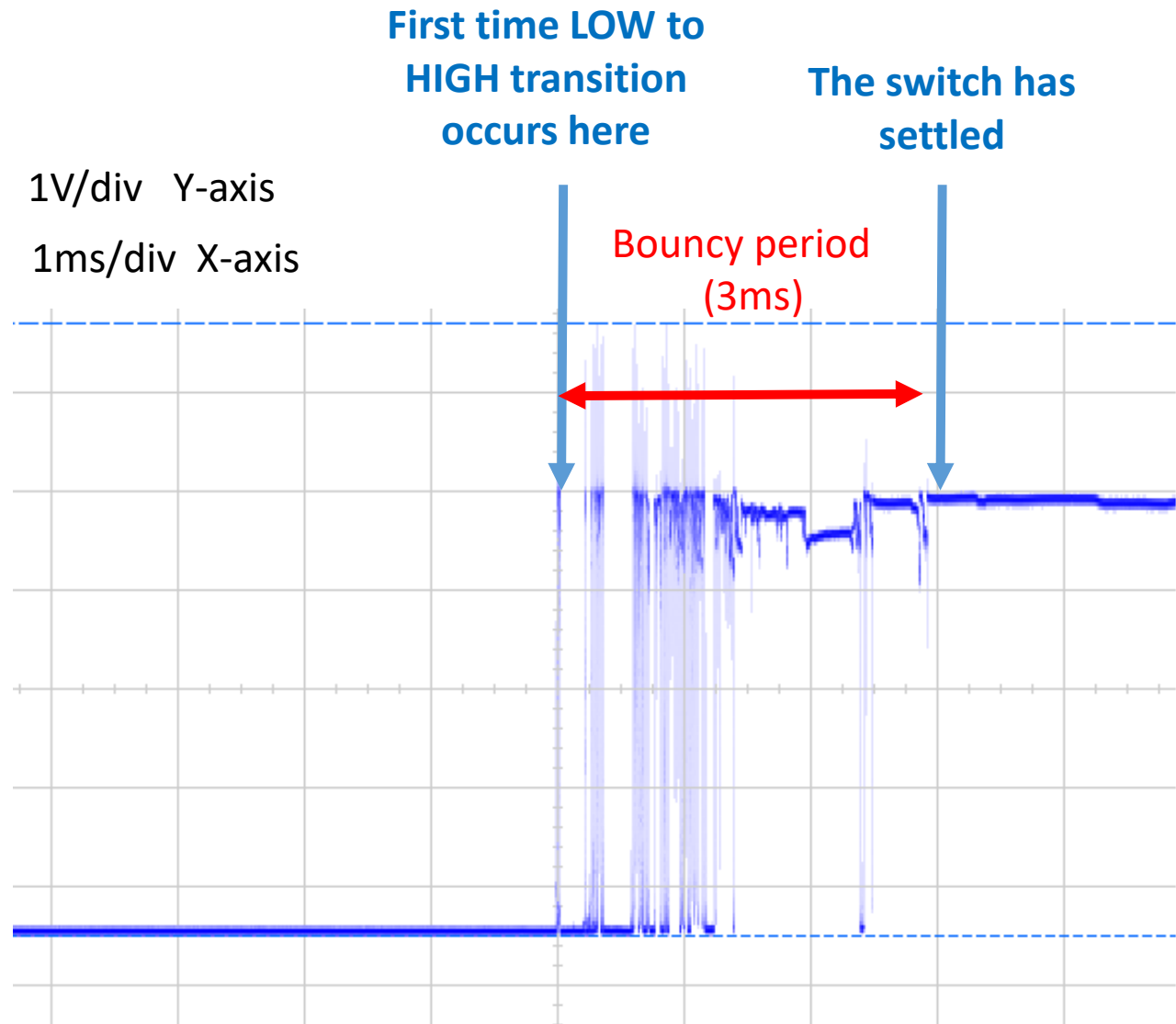
Typical
waveform of a
button press

(910 ns/div)

- Switches can be debounced in **hardware** or **software**.
- There are many ways to debounce (within each category)



Example of a hardware debouncer.



One possible way to software-debounce (LTH):

Whenever a LOW to HIGH (LTH) transition occurs,

- Do nothing and wait for a certain amount of time
- Check status of the pin again after the prescribed time has passed
- If it is still HIGH, then it is a real transition

Another possible way to software-debounce (LTH):

Whenever a LOW to HIGH (LTH) transition occurs,

- Start a count-down timer
- If the logic level goes back to LOW, clear/stop the timer and then quit.
- If the timer has fired, then it is a real transition.


```

unsigned int count;
bool previous;
unsigned char *reader = (unsigned char*) 0x23;

int main()                //To run on Arduino, just change this function to void setup()
{
    Serial.begin(9600);    //For now, we are still using the Serial library

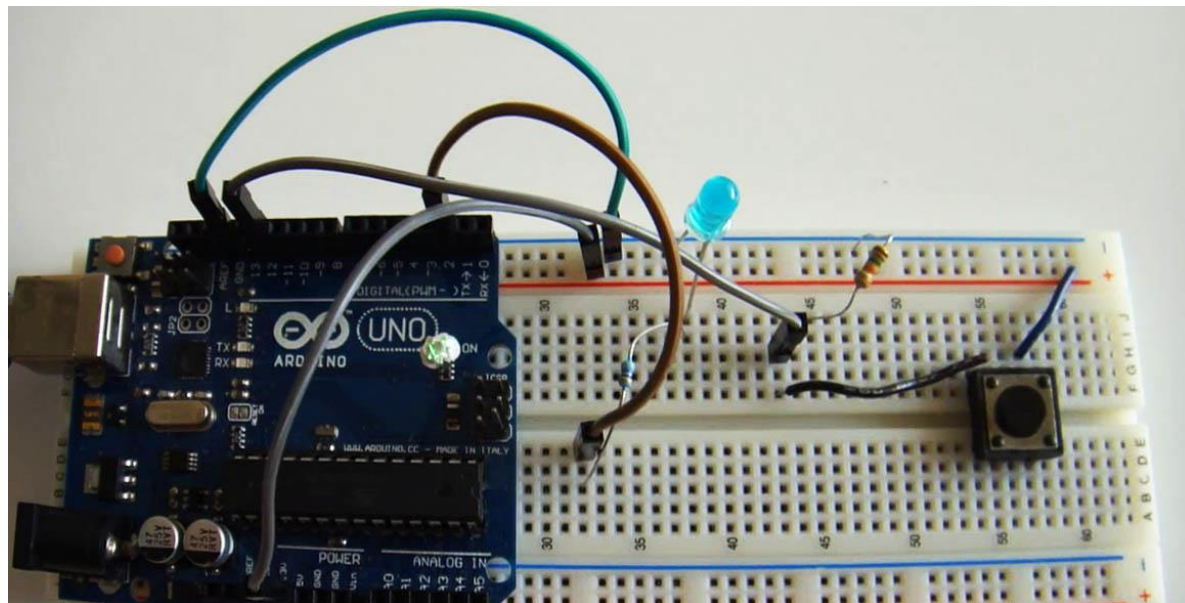
    for (;;)
    {
        bool current = (((*reader) & 0b00010000) != 0);
        if (current && !previous)    //If the pin is currently HIGH and previously LOW
        {
            delay(20);    //Do nothing and wait for 20ms.
            if (((*reader) & 0b00010000) != 0) //If the pin is still HIGH
            {
                count++;
                Serial.println(count);
            }
        }
        previous = current;
    }
}

```

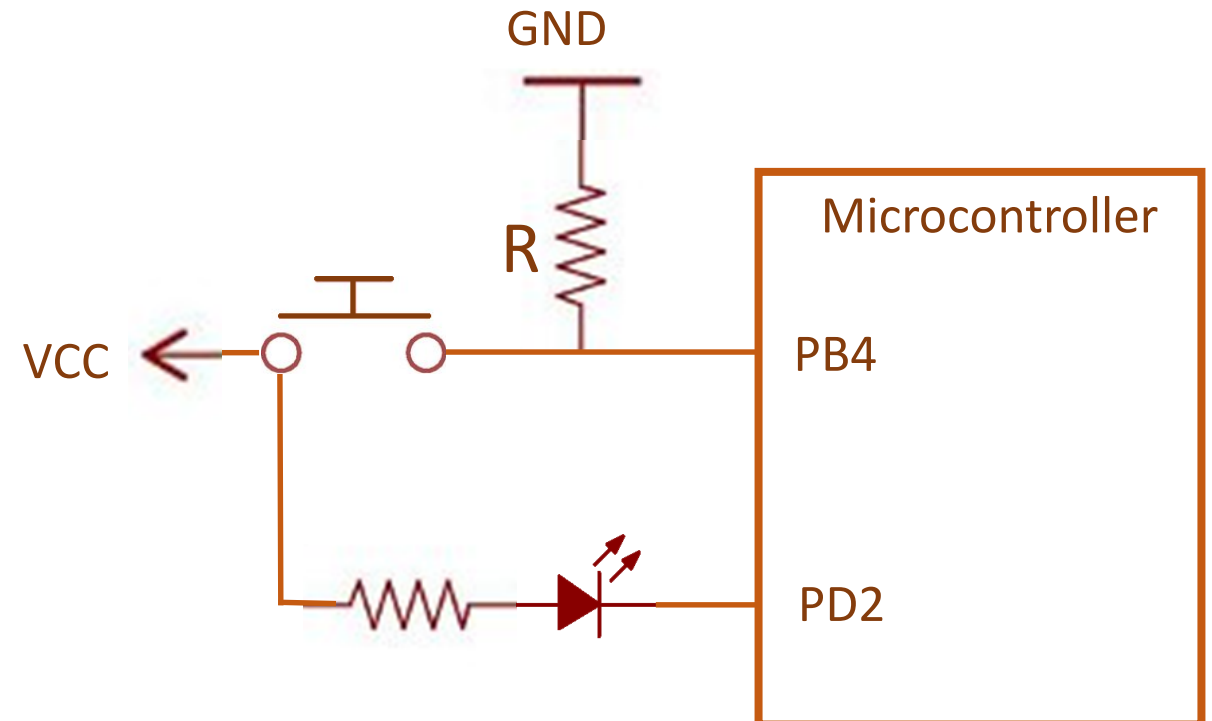


Example 6

The microcontroller needs to accomplish the same task as in the previous example. This time, an LED attached to PD2 needs to be flashing with a period of 4 seconds and 50% duty cycle. Program the microcontroller to accomplish this.



Note: there is a little mistake in my wiring. PD2 is sourcing current instead of sinking (as intended).



Note: GPIP pins can both source and sink current

```

unsigned int count;
bool previous;
unsigned char *B_reader = (unsigned char*) 0x23; //Points to PINB
unsigned char *D_dir = (unsigned char*) 0x2A; //Points to DDRD
unsigned char *D_writer = (unsigned char*) 0x2B; //Points to PORTD

int main() //To run on Arduino, just change this function to void setup()
{

```

```

    Serial.begin(9600);
    *D_dir = 0b00000100;
    for (;;)
    {

```

This piece of code does blinking

```

        *D_writer |= 0b00000100;
        delay(2000);
        *D_writer &= 0b11111011;
        delay(2000);

```

This piece of code handles button pressing

```

        bool current = (((*B_reader) & 0b00010000) != 0);
        if (current && !previous)
        {
            delay(20);
            if (((*B_reader) & 0b00010000) != 0)
            {
                count++;
                Serial.println(count);
            }
        }
        previous = current;
    }
}

```

Note: this is how NOT to do it



- The previous program does not work.
- Problem is the delay function
- Solution: never use the delay function!

Blinking LED without delay

```
unsigned long previousTime;
unsigned char *D_dir = (unsigned char*) 0x2A;    //Points to DDRD
unsigned char *D_writer = (unsigned char*) 0x2B;  //Points to PORTD

int main()                                       //To run on Arduino, just change this function to void setup()
{
    *D_dir = 0b00000100;
    for (;;)
    {
        unsigned long currentTime = millis();

        if (((*D_writer) & 0b00000100) == 0)    //If the LED is currently OFF
        {
            if (currentTime - previousTime >= 2000)    //2000 is OFF duration
            {
                *D_writer |= 0b00000100;              //Turn ON the LED
                previousTime = currentTime;              //Take note of the time
            }
        }
        else                                       //LED is currently ON
        {
            if (currentTime - previousTime >= 2000)    //2000 is ON duration
            {
                *D_writer &= 0b11111011;              //Turn OFF the LED
                previousTime = currentTime;              //Take note of the time
            }
        }
        //Code to perform other functions (such as button debouncing) can be inserted here
    }
}
```

Debouncing without delay

```
unsigned int count;
bool previous;
bool waiting;
unsigned long pressed_time;
unsigned char *B_reader = (unsigned char*) 0x23; //Points to PINB

int main() //To run on Arduino, just change this function to void setup()
{
    Serial.begin(9600);
    for (;;)
    {
        //Code to perform other functions (such as LED blinking) can be inserted here
        if (waiting)
        {
            if (millis() - pressed_time >= 20) //20ms is the debounce period. (If the debounce period has elapsed)
            {
                waiting = 0;
                bool current = (((*B_reader) & 0b00010000) != 0); //current = HIGH if input PB4 is HIGH
                if (current) //If PB4 is still HIGH, it means it is a real LOW to HIGH transition
                {
                    count++;
                    Serial.println(count);
                }
                previous = current;
            }
        }
        else
        {
            bool current = (((*B_reader) & 0b00010000) != 0); //current = HIGH if input PB4 is HIGH
            if (current && !previous) //If the PB4 is currently HIGH and previously LOW (LOW to High transition)
            {
                pressed_time = millis(); //Take note of the time and then start waiting
                waiting = 1;
            }
            previous = current;
        }
    }
}
```

Combining blinking + debouncing (multi-tasking)

Code for blinking LED and debouncing can simply be merged.

```
////////// Variables for blinking LED //////////
unsigned long previousTime;
unsigned char *D_dir = (unsigned char*) 0x2A;
unsigned char *D_writer = (unsigned char*) 0x2B;

////////// Variables for debouncing//////////
unsigned int count;
bool previous;
bool waiting;
unsigned long pressed_time;
unsigned char *B_reader = (unsigned char*) 0x23;

int main()
{
    *D_dir = 0b00000100;

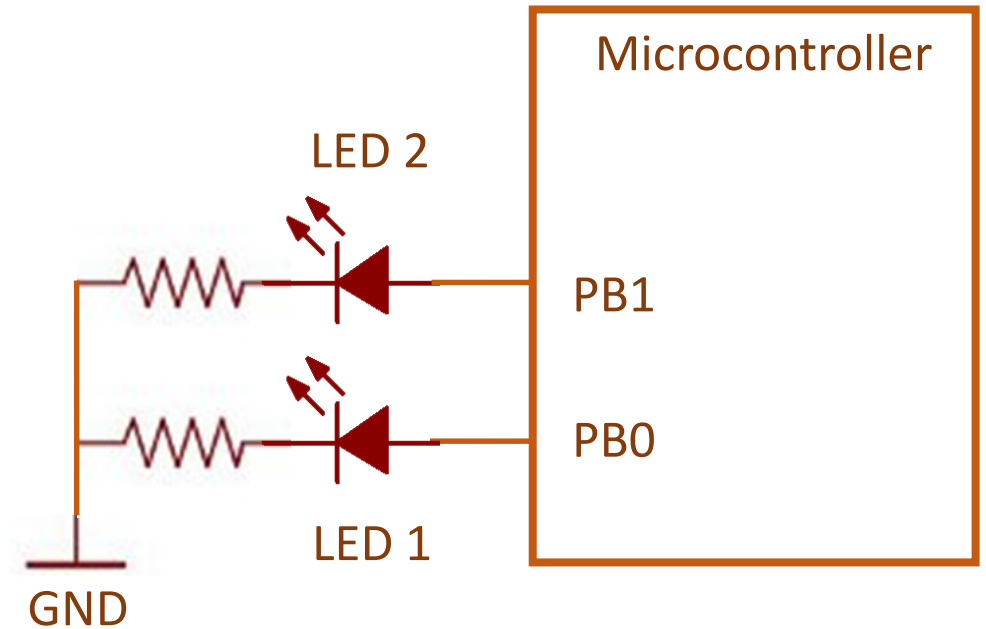
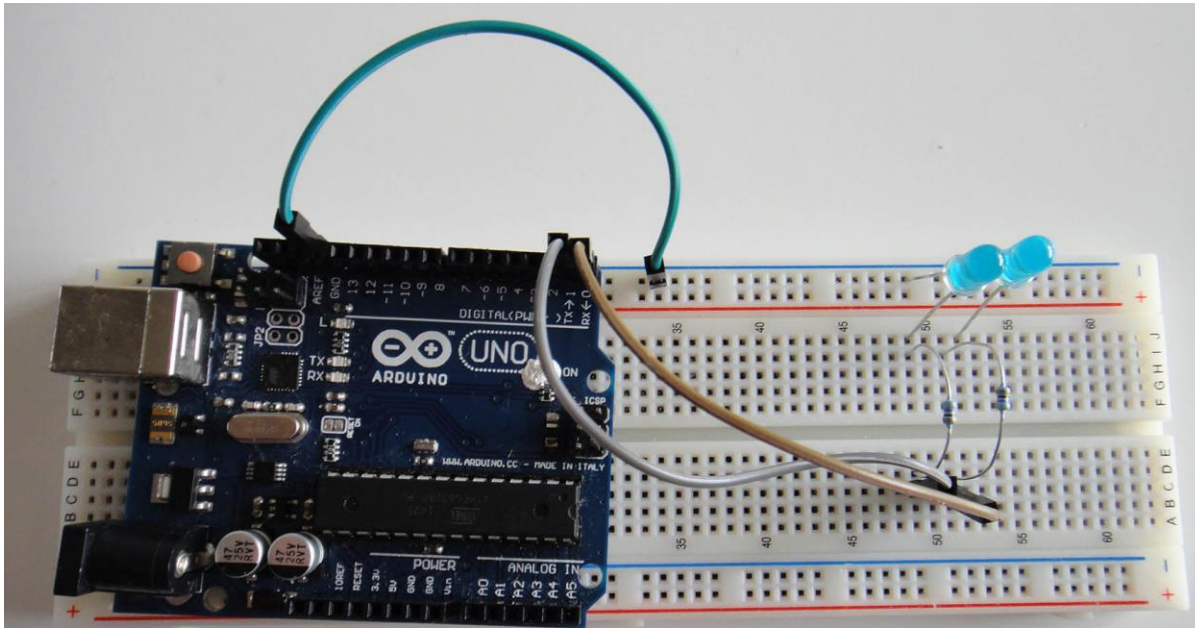
    Serial.begin(9600);

    for (;;)
    {
        BlinkLED();           //BlinkLED handles stuff for blinking PD2
        DebounceButton();     //DebounceButton handles stuff for debouncing PB4
    }
}
```



Example 7

There are two LEDs whose anodes are connected to PB0 and PB1 respectively. The LED 1 needs to be flashing with ON time 750 ms and OFF time 350 ms. The LED 2 needs to be flashing with ON time 400 and OFF time 600 ms.



Port Pin# ON duration OFF duration

```
Blinky led0('D', 0, 750, 350);  
Blinky led1('D', 1, 400, 600);
```

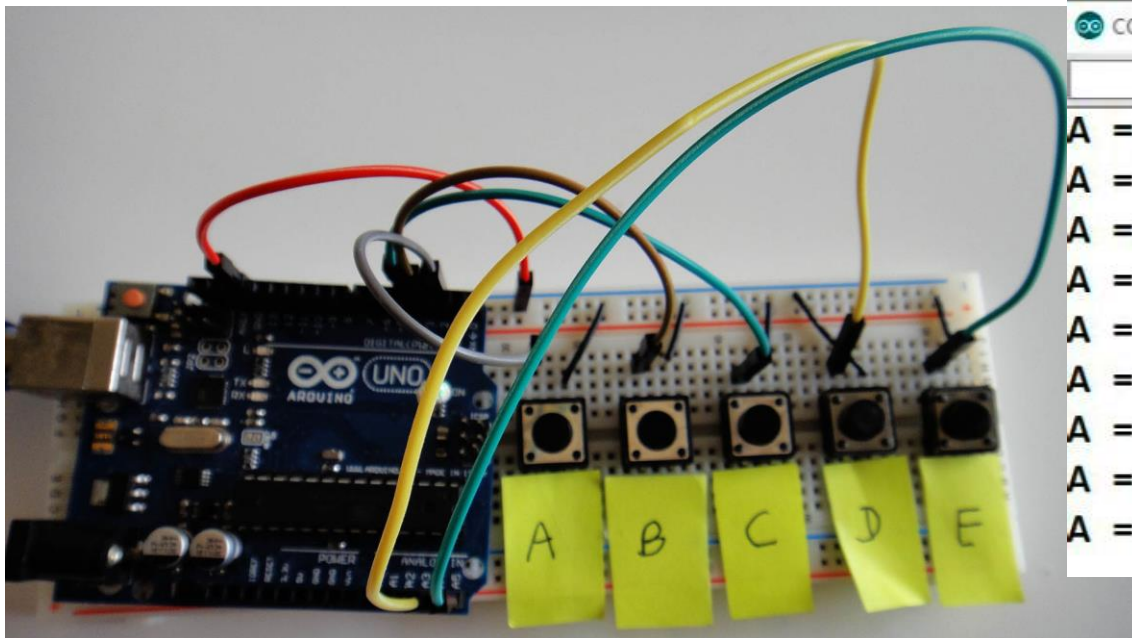
```
int main()  
{  
    while(1)  
    {  
        led0.Refresh();  
        led1.Refresh();  
    }  
}
```

OOP can be exploited to
reduce code complexity.



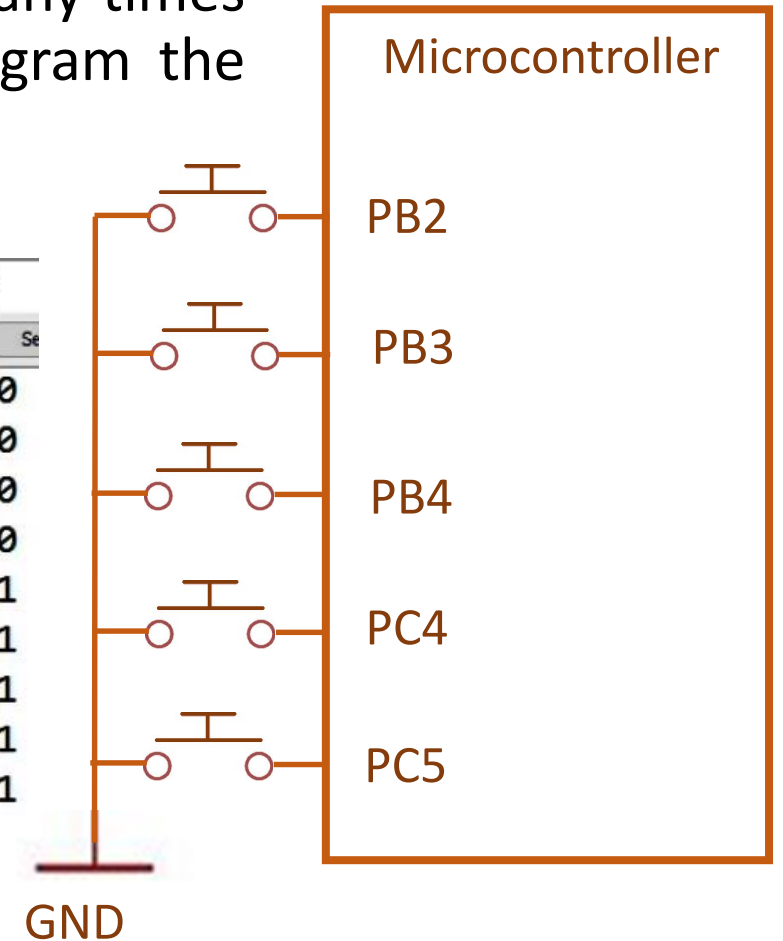
Example 8

Five switches (A,B, C, D, and E) are connected to ATmega328p as shown below. The microcontroller needs to keep track of how many times each button has been pressed. Whenever any button has been pressed, the microcontroller needs to send out the information (of how many times each button has been pressed) through the serial port. Program the microcontroller to accomplish this.



```
COM6 (Arduino/Genuino Uno)

A = 1 B = 0 C = 0 D = 0 E = 0
A = 1 B = 1 C = 0 D = 0 E = 0
A = 1 B = 1 C = 1 D = 0 E = 0
A = 1 B = 1 C = 1 D = 1 E = 0
A = 1 B = 1 C = 1 D = 1 E = 1
A = 2 B = 1 C = 1 D = 1 E = 1
A = 2 B = 2 C = 1 D = 1 E = 1
A = 2 B = 2 C = 2 D = 1 E = 1
A = 2 B = 3 C = 2 D = 1 E = 1
```



```
unsigned int A_count, B_count, C_count, D_count, E_count;
```

```
Button A('D', 2, 20, 1, 1), B('D', 3, 20, 1, 1), C('D', 4, 20, 1, 1),  
        D('C', 4, 20, 1, 1), E('C', 5, 20, 1, 1);
```

```
void setup()  
{
```

```
    Serial.begin(9600);
```

```
    A.Pressed = A_gets_pressed;
```

```
    B.Pressed = B_gets_pressed;
```

```
    C.Pressed = C_gets_pressed;
```

```
    D.Pressed = D_gets_pressed;
```

```
    E.Pressed = E_gets_pressed;
```

```
    while(1)
```

```
    {
```

```
        A.Refresh();
```

```
        B.Refresh();
```

```
        C.Refresh();
```

```
        D.Refresh();
```

```
        E.Refresh();
```

```
    }
```

```
}
```

Port

Pin #

Debounce
period

Normally
high or low?

Use
internal
pull up?

“Pressed” is an event of the button class. A function called `A_gets_pressed()` will be invoked whenever button A has been pressed. Other examples of events are *Released* and *DoublePressed*.

In OOP, objects can have

- properties (normal variables)
- methods (functions)
- events (function pointers)

```
void A_gets_pressed()
{
    A_count++;
    print();
}
void B_gets_pressed()
{
    B_count++;
    print();
}
void C_gets_pressed()
{
    C_count++;
    print();
}
void D_gets_pressed()
{
    D_count++;
    print();
}
void E_gets_pressed()
{
    E_count++;
    print();
}
```

```
void print()
{
    Serial.print("A = ");
    Serial.print(A_count);

    Serial.print(" B = ");
    Serial.print(B_count);

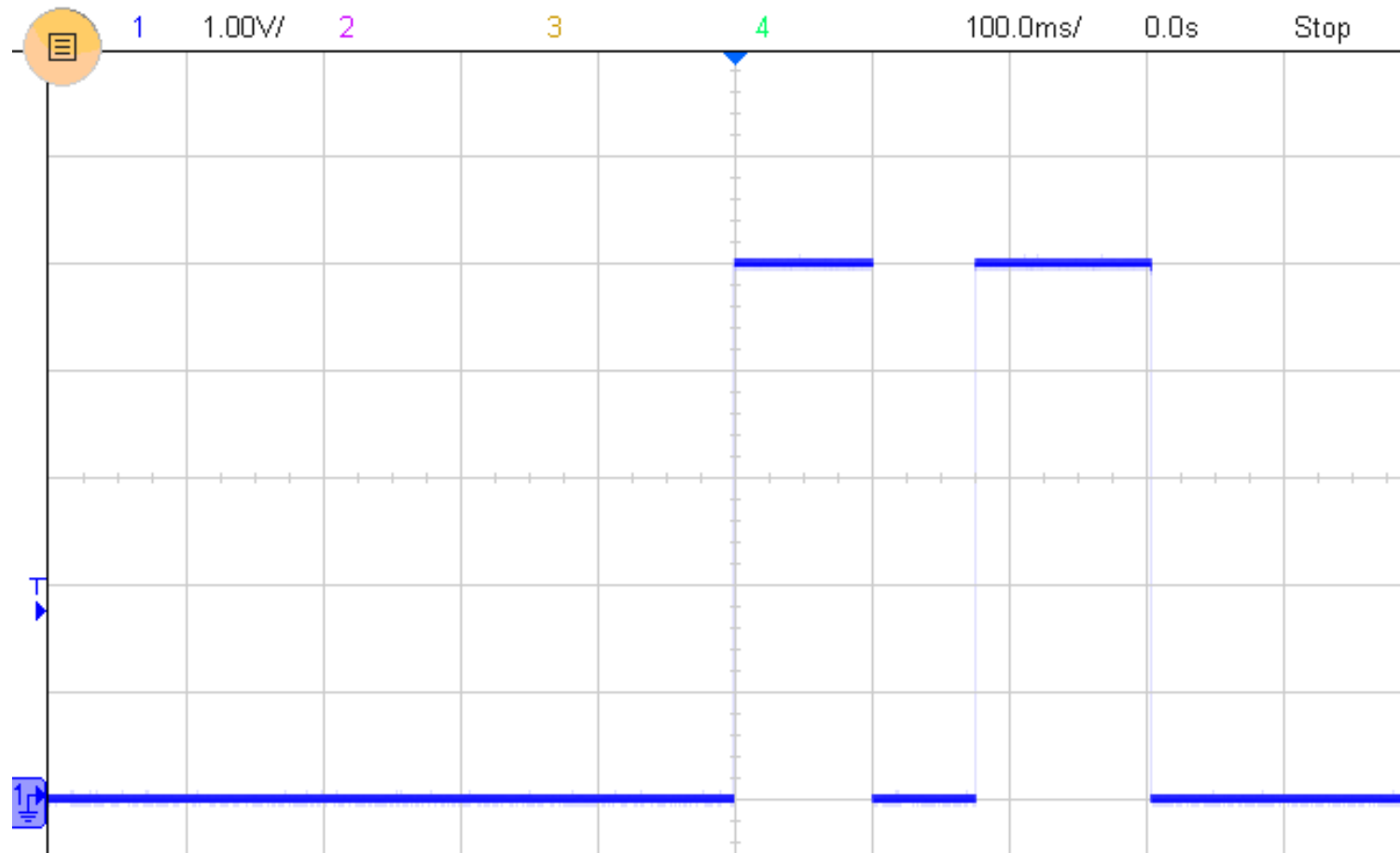
    Serial.print(" C = ");
    Serial.print(C_count);

    Serial.print(" D = ");
    Serial.print(D_count);

    Serial.print(" E = ");
    Serial.println(E_count);
}
```

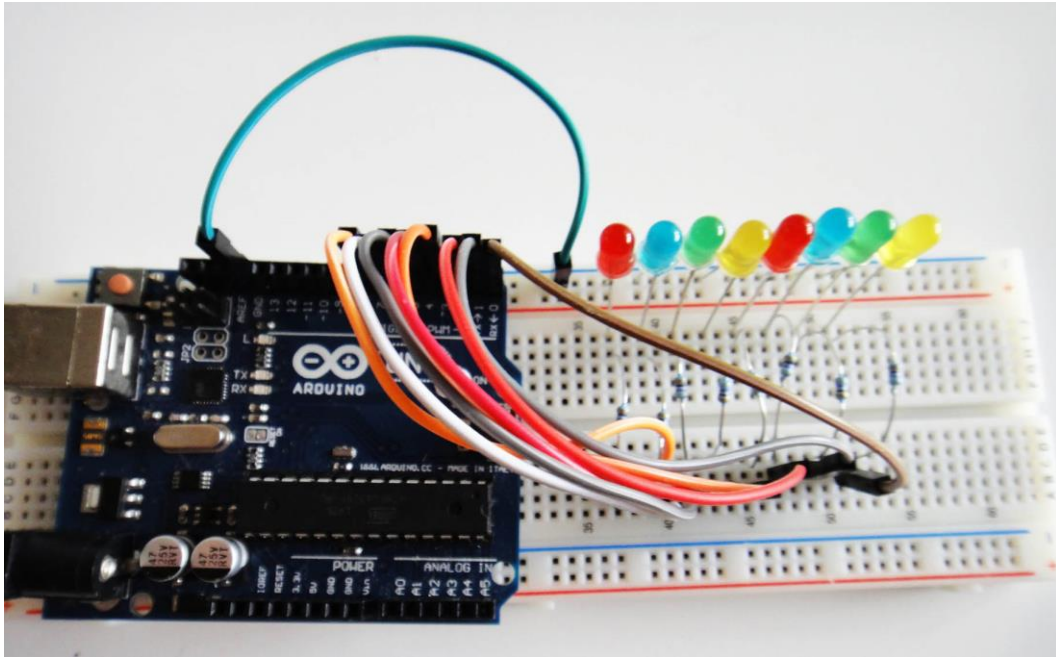


DoublePress Event can also be extended from the Press Event. A double-press event may be defined as two consecutive presses within a time span of, say, 400 ms.



Example 9

Write a program for ATmega328p that blinks 8 LEDs (on PD0 to PD7) with the following on and off times.



Pin	On time (ms)	Off time (ms)
PD0	781	515
PD1	2014	1348
PD2	343	573
PD3	678	1839
PD4	342	534
PD5	1478	326
PD6	1859	351
PD7	777	888

Blinky leds[8];

```
int main()
{
    leds[0].Initialize('D', 0, 781, 515);
    leds[1].Initialize('D', 1, 2014, 1348);
    leds[2].Initialize('D', 2, 343, 573);
    leds[3].Initialize('D', 3, 678, 1839);
    leds[4].Initialize('D', 4, 342, 534);
    leds[5].Initialize('D', 5, 1478, 326);
    leds[6].Initialize('D', 6, 1859, 351);
    leds[7].Initialize('D', 7, 777, 888);

    for (;;)
    {
        for (int i=0; i<8; i++)
        {
            leds[i].Refresh();
        }
    }
}
```

