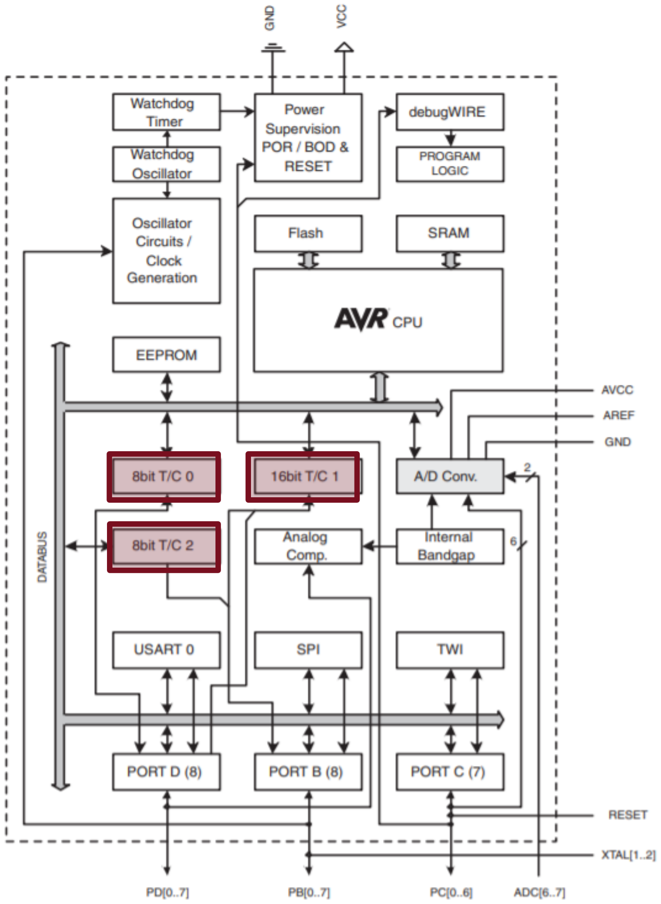
Timers

ELEC3042 EMBEDDED SYSTEMS

Lecture 3

Timer

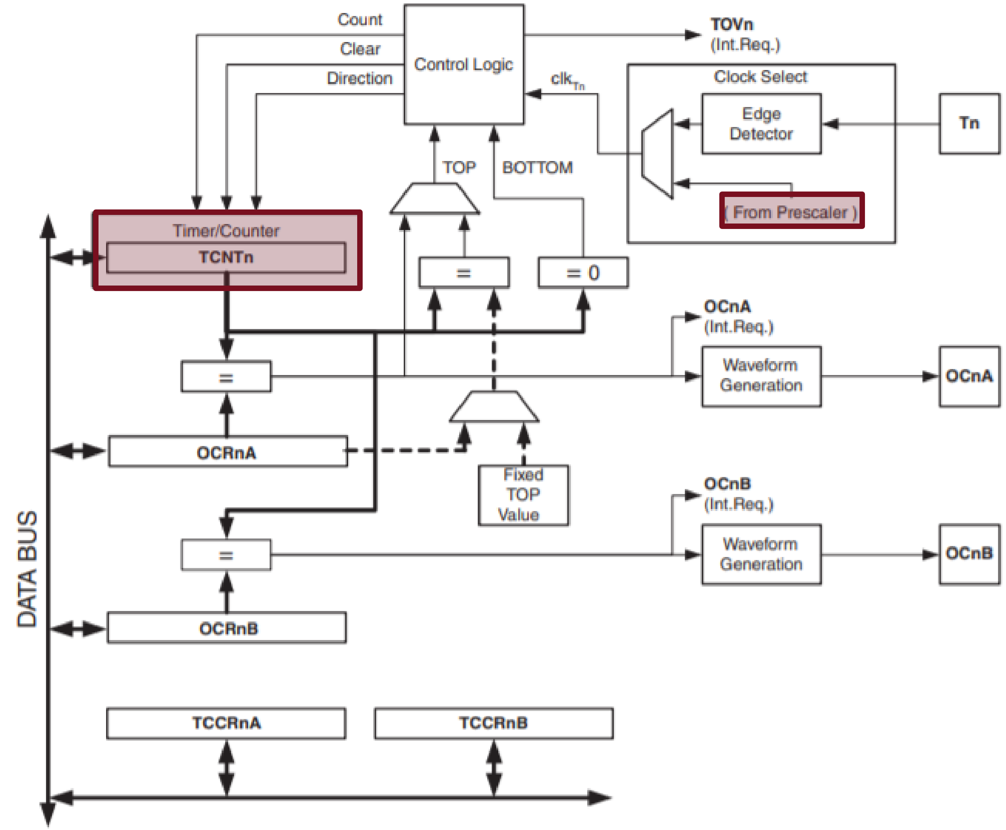
Figure 2-1. Block Diagram



- Timing of events is important in many digital system applications
- ATmega328P has 3 in-built timers
  - Timer0 & Timer2 – 8-bit
  - Timer1 – 16-bit
- Each timer has various modes of operation and 3 interrupt generators
- Each timer has different features for different purposes

How does the timers work?

Figure 15-1. **8-bit Timer/Counter Block Diagram**

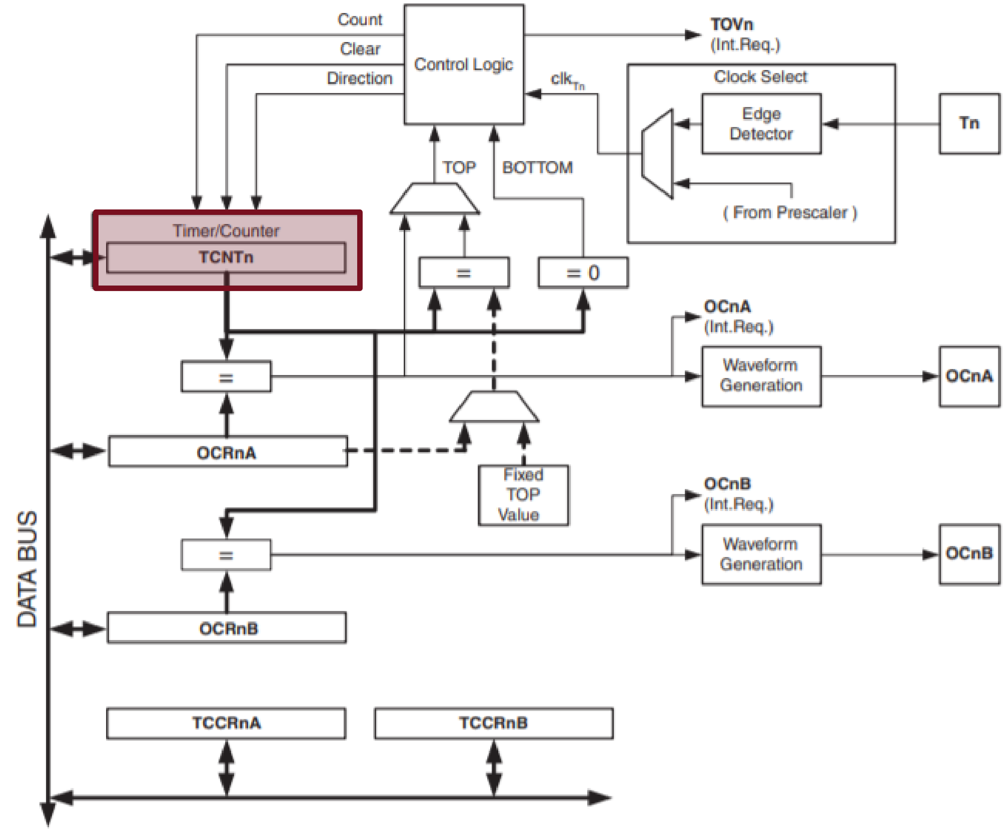


- Counts clock pulses
- Clock can be the I/O clock or an external clock
- If I/O clock is used, a prescaler can be applied to divide the clock frequency by 8, 64, 256 or 1024

Timer Modes of Operation

1. NORMAL MODE

Figure 15-1. 8-bit Timer/Counter Block Diagram



- TCNT counts up from 0 until it overflows (generates interrupt), and then starts from 0 again.

TCNT0

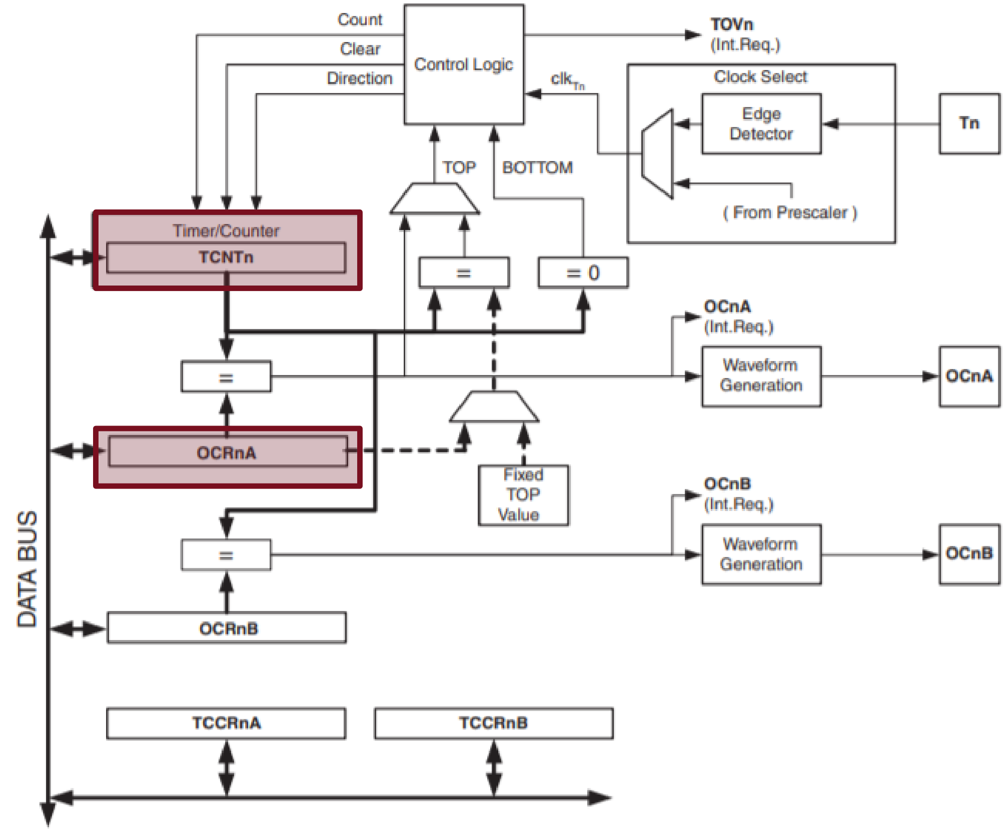
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
			⋮				
1	1	1	1	1	1	1	1

Interrupt

Timer Modes of Operation

2. CTC MODE

Figure 15-1. 8-bit Timer/Counter Block Diagram



- CTC = clear timer on compare match
- When count matches value in OCRnA, the counter will reset (and generate an interrupt)
- Used to have timer count to numbers smaller than maximum count

TCNT0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
⋮							
1	1	1	1	1	0	0	1

OCR0A

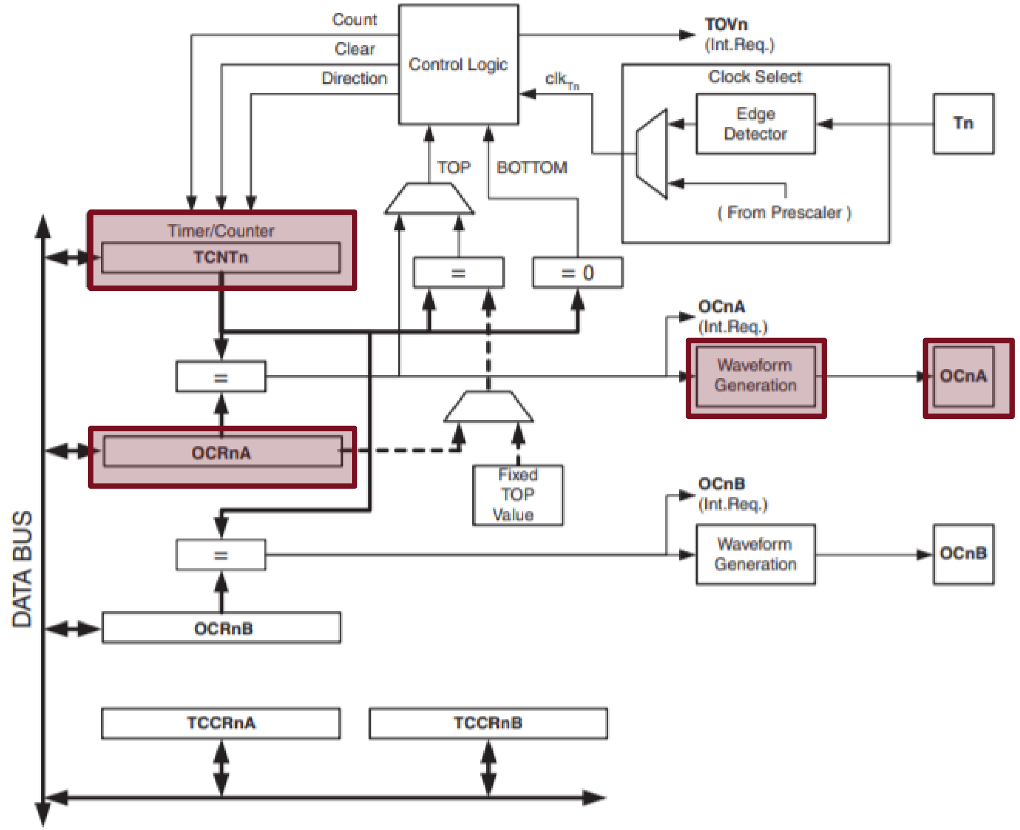
1	1	1	1	1	0	0	1

Interrupt

Timer Modes of Operation

3. PWM MODE

Figure 15-1. **8-bit Timer/Counter Block Diagram**

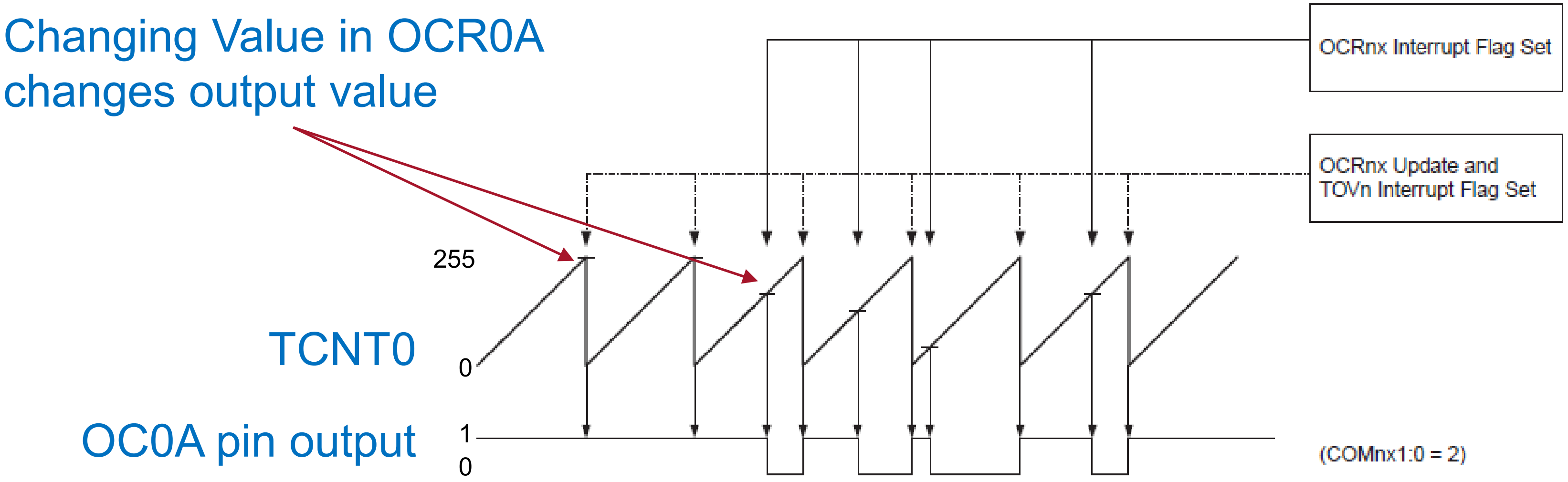


- PWM = pulse width modulation
- Can also be used to generate different output waveforms
- Example uses:
  - change the brightness of LEDs
  - drive motor at different speeds

Timer Modes of Operation

3. PWM MODE

Figure 15-6. Fast PWM Mode, Timing Diagram



OCIE0A – compare match interrupt

- Set OC0A output to 0

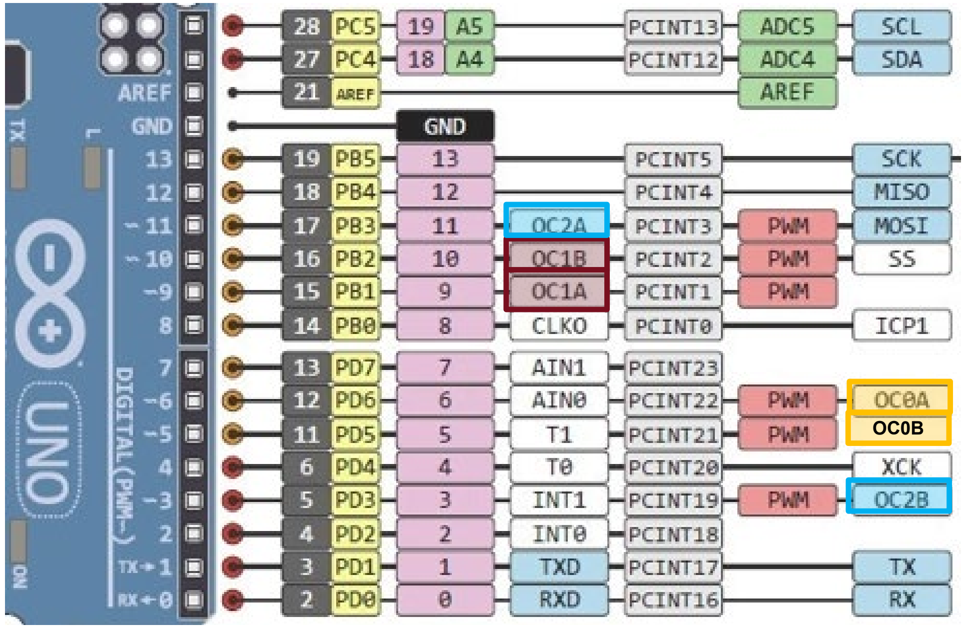
TOIE0 – overflow interrupt

- Set OC0A output to 1
- Change OCR0A

Choosing a timer

Depends on several factors:

- The size of the divisor
  - value put into compare register
  - 8 vs 16 bit
- Whether an external output is required
  - which pins are available
- Accuracy of the PWM output
  - 8 vs 16 bit

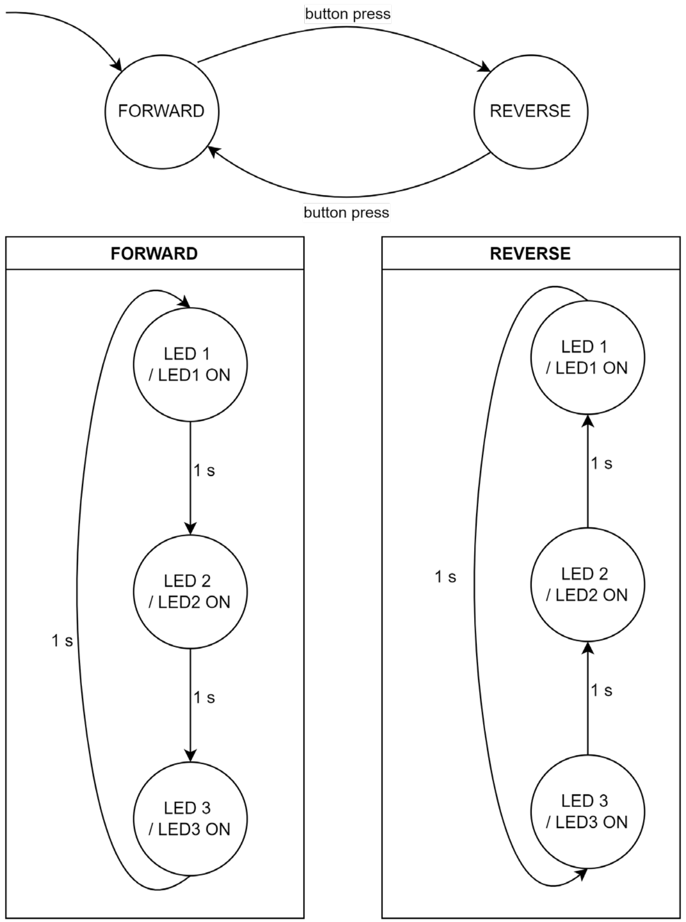


Toy Problem

Create a program that cycles and lights up one of three LEDs for one second each. A push button is used to change the direction of the cycling whenever it is pressed.

DESIGN:

- LEDs are connected to PB5 (LED1), PB4 (LED2), PB3 (LED3)
- Pushbutton is connected to PD2



Code

```c
#include <avr/io.h>

enum STATE {FORWARD, REVERSE}; // define our own data type

void delay_ms(long num) {
    while (num--) {
        for (volatile long x = 0; x < 468; x++) {
            ;
        }
    }
}

void setup(void) {
    DDRB  |= 0b00111000;
    PORTB |= 0b00100000;
    DDRD  &= 0b11111011;
    PORTD |= 0b00000100;
}
```

```c
int main(void) {
    setup();
    enum STATE cur_state = FORWARD;

    while (1) {
        switch (cur_state) {
            case FORWARD:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = REVERSE;    // change state
                } else {
                    delay_ms(1000);
                    PORTB = (PORTB==0b00001000)?0b00100000:PORTB>>1; // shift LED
                }
                break;
            case REVERSE:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = FORWARD;    // change state
                } else {
                    delay_ms(1000);
                    PORTB = (PORTB==0b00100000)?0b00001000:PORTB<<1; // shift LED
                }
                break;
        }
    }
}
```

Busy loops

- Undesirable because:
  - Does no meaningful work
  - Holds up the CPU
  - Wastes energy
- Replace delay_ms(1000) with timer
  - Count enough clock ticks for 1 s and generate interrupt

```
void delay_ms(long num) {
    while (num--) {
        for (volatile long x = 0; x < 468; x++) {
            ;
        }
    }
}
```

1. Choose a timer

- Arduino has 16 MHz clock
  - Choose an appropriate prescaler
    - No Prescaler - need 24 bits to count to 16 million (2^{24} = 16,777,216) – don't have a 24-bit timer
    - Prescaler of 8 => 16,000,000 / 8 = 2,000,000 clock ticks per second
      - To represent 2,000,000, need 21 bits (2^{21} = 2,097,152) – don't have a 21-bit timer
    - Prescaler of 64 => 16,000,000 / 64 = 250,000 clock ticks per second
      - To represent 250,000, need 18 bits
    - Prescaler of 256 => 16,000,000 / 256 = 62,500 clock ticks per second
      - To represent 62,500, need 16 bits
    - Prescaler of 1024 => 16,000,000 / 1024 = 15,625 clock ticks per second
      - To represent 15,625, need 14 bits
- Use Timer1
- Which prescaler?
  - Choose largest prescaler that will count to a whole number
  - If whole number is not possible, a lower prescaler gives higher precision (reduces error)

2. Choose Timer mode of operation?

- Modes of operation:
  1. Normal – counts up from 0 until it overflows (generates interrupt), and then starts from 0 again ✗
  2. CTC (clear timer on compare match) – when count matches value in OCRnA, the counter will reset (and generate an interrupt) ✓
  3. PWM (pulse width modulation) – used to generate different output waveforms ✗

Overflow exceeds 1 s

- 16 MHz/1024 = 15,625 clock ticks per second
- 2^{16} = 65536 clock ticks before overflow
- => 65536/15625 = 4.1943 seconds before overflow

Set OCR1A to 15,625

Not generating an output waveform on one pin

3. Setup Control Registers

SET TIMER MODE OF OPERATION

16.9 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the *Waveform Generation mode* (WGM13:0) and *Compare Output mode* (COM1x1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM1x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM1x1:0 bits control whether the output should be set, cleared or toggle at a compare match (See "Compare Match Output Unit" on page 130.)

TCCR1A = 0b00000000;

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 16-1. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on Compare Match.
1	0	Clear OC1A/OC1B on Compare Match (Set output to low level).
1	1	Set OC1A/OC1B on Compare Match (Set output to high level).

Connects OC1A/OC1B to pins

3. Setup Control Registers

SET TIMER MODE OF OPERATION

16.9 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the *Waveform Generation mode* (WGM13:0) and *Compare Output mode* (COM1x1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COM1x1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COM1x1:0 bits control whether the output should be set, cleared or toggle at a compare match (See "Compare Match Output Unit" on page 130.)

TCCR1A = 0b00000000;
TCCR1B = 0b00001000;

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 16-4. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

3. Setup Control Registers

SET CLOCK PRESCALER

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

TCCR1A = 0b00000000;
TCCR1B = 0b00001101;

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

For input, not using – set to 0

3. Setup Control Registers

TCCR1A = 0b00000000;
TCCR1B = 0b00001101;
TCCR1C = 0;

16.11.3 TCCR1C – Timer/Counter1 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	–	–	–	–	–	–	TCCR1C
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC1A: Force Output Compare for Channel A**
- **Bit 6 – FOC1B: Force Output Compare for Channel B**

The FOC1A/FOC1B bits are only active when the WGM13:0 bits specifies a non-PWM mode. When writing a logical one to the FOC1A/FOC1B bit, an immediate compare match is forced on the Waveform Generation unit. The OC1A/OC1B output is changed according to its COM1x1:0 bits setting. Note that the FOC1A/FOC1B bits are implemented as strobes. Therefore it is the value present in the COM1x1:0 bits that determine the effect of the forced compare.

4. Setup Compare Registers

- Prescaler of 1024 => 15625 clock ticks per second

16.11.4 TCNT1H and TCNT1L – Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.5 OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.6 OCR1BH and OCR1BL – Output Compare Register 1 B

Bit	7	6	5	4	3	2	1	0	
(0x8B)	OCR1B[15:8]								OCR1BH
(0x8A)	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.7 ICR1H and ICR1L – Input Capture Register 1

Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
TCCR1A = 0b00000000;
TCCR1B = 0b00001101;
TCCR1C = 0;
TCNT1 = 0;
OCR1A = 15624;
OCR1B = 0;
ICR1 = 0;
```

5. Setup Interrupt

16.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – ICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 66) is executed when the ICF1 Flag, located in TIFR1, is set.

- **Bit 2 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 66) is executed when the OCF1B Flag, located in TIFR1, is set.

- **Bit 1 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 66) is executed when the OCF1A Flag, located in TIFR1, is set.

- **Bit 0 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Overflow interrupt is enabled. The corresponding Interrupt Vector (See "Interrupts" on page 66) is executed when the TOV1 Flag, located in TIFR1, is set.

```
TCCR1A = 0b00000000;
TCCR1B = 0b00001101;
TCCR1C = 0;
TCNT1 = 0;
OCR1A = 15624;
OCR1B = 0;
ICR1 = 0;
TIMSK1 = 0b00000010;
```

5. Setup Interrupt

MACQUARIE University SYDNEY·AUSTRALIA

16.11.9 TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – ICF1: Timer/Counter1, Input Capture Flag**

This flag is set when a capture event occurs on the ICP1 pin. When the Input Capture Register (ICR1) is set by the WGM13:0 to be used as the TOP value, the ICF1 Flag is set when the counter reaches the TOP value.

ICF1 is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

- **Bit 2 – OCF1B: Timer/Counter1, Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register B (OCR1B).

Note that a Forced Output Compare (FOC1B) strobe will not set the OCF1B Flag.

OCF1B is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCF1B can be cleared by writing a logic one to its bit location.

- **Bit 1 – OCF1A: Timer/Counter1, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register A (OCR1A).

Note that a Forced Output Compare (FOC1A) strobe will not set the OCF1A Flag.

OCF1A is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

- **Bit 0 – TOV1: Timer/Counter1, Overflow Flag**

The setting of this flag is dependent of the WGM13:0 bits setting. In Normal and CTC modes, the TOV1 Flag is set when the timer overflows. Refer to Table 16-4 on page 141 for the TOV1 Flag behavior when using another WGM13:0 bit setting.

TOV1 is automatically cleared when the Timer/Counter1 Overflow Interrupt Vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location.

```
TCCR1A = 0b00000000;
TCCR1B = 0b00001101;
TCCR1C = 0;
TCNT1 = 0;
OCR1A = 15624;
OCR1B = 0;
ICR1 = 0;
TIMSK1 = 0b00000010;
TIFR1 = 0;
```

Code changes

```c
#include <avr/io.h>
#include <avr/interrupt.h>

enum STATE {FORWARD, REVERSE};
volatile uint8_t time1s = 0;

ISR(TIMER1_COMPA_vect) {
    time1s = 1;
}

void setup(void) {
    DDRB  |= 0b00111000;
    PORTB  |= 0b00100000;
    DDRD &= 0b11111011;
    PORTD  |= 0b00000100;
}

void timersetup(void) {
    TCCR1A = 0b00000000;
    TCCR1B = 0b00001101;
    TCCR1C = 0;
    TCNT1  = 0;
    OCR1A = 15624;
    OCR1B = 0;
    ICR1  = 0;
    TIMSK1 = 0b00000010;
    TIFR1 = 0;
}
```

```c
int main(void) {
    setup();
    timersetup();
    sei();

    enum STATE cur_state = FORWARD;

    while (1) {
        switch (cur_state) {
            case FORWARD:
                if ((PIND & 0b00000100) == 0) {
                        cur_state = REVERSE;     // change state
                } else {
                    if (time1s == 1) {
                        time1s = 0;
                        PORTB = (PORTB==0b00001000)?0b00100000:PORTB>>1; // shift LED
                    }
                }
                break;
            case REVERSE:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = FORWARD;     // change state
                } else {
                    if (time1s == 1) {
                        time1s = 0;
                        PORTB = (PORTB==0b00100000)?0b00001000:PORTB<<1; // shift LED
                    }
                }
                break;
        }
    }
}
```

One timer for multiple events

- Example
  - LED changes once per second
  - Button debounce time is ~10 ms
- Implementation
  - Set up timer to interrupt every 10 ms
  - LED should change after 100 interrupts
  - Button is debounced on next interrupt

Exercise

MACQUARIE University SYDNEY·AUSTRALIA

Set up Timer 1 to generate an interrupt every 10 ms

- Choose prescaler
- Set OCR1A value

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

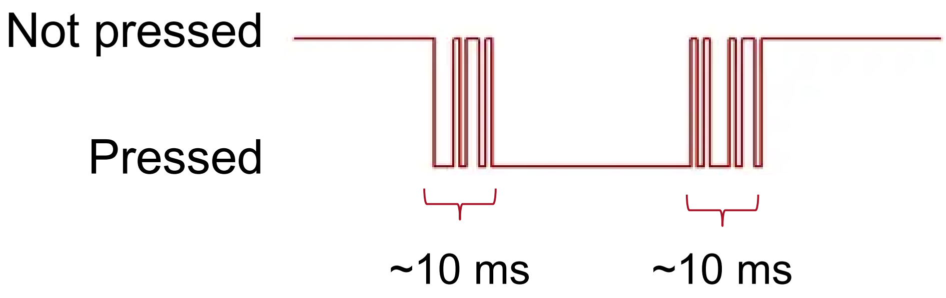
```
TCCR1A = 0b00000000;
TCCR1B = 0b00001???;
TCCR1C = 0;
TCNT1 = 0;
OCR1A = ???;
OCR1B = 0;
ICR1 = 0;
TIMSK1 = 0b00000010;
TIFR1 = 0;
```

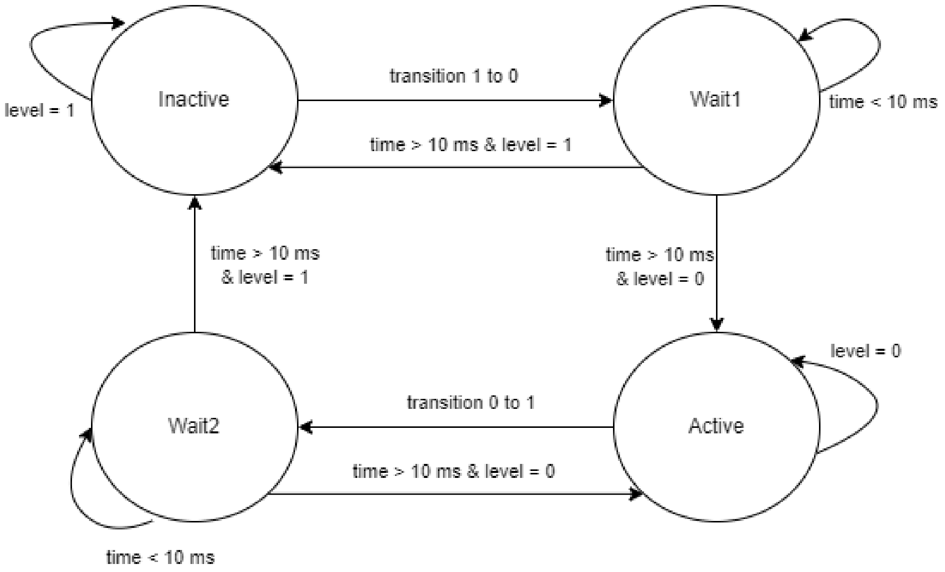
Code changes

```c
#include <avr/io.h>
#include <avr/interrupt.h>

enum STATE {FORWARD, REVERSE};
volatile uint32_t count_ms = 0;

ISR(TIMER1_COMPA_vect) {
    count_ms = count_ms + 10;
}

void setup(void) {
    DDRB |= 0b00111000;
    PORTB |= 0b00100000;
    DDRD &= 0b11111011;
    PORTD |= 0b00000100;
}

void timersetup(void) {
    TCCR1A = 0b00000000;
    TCCR1B = 0b00001100;
    TCCR1C = 0;
    TCNT1 = 0;
    OCR1A = 624;
    OCR1B = 0;
    ICR1 = 0;
    TIMSK1 = 0b00000010;
    TIFR1 = 0;
}
```

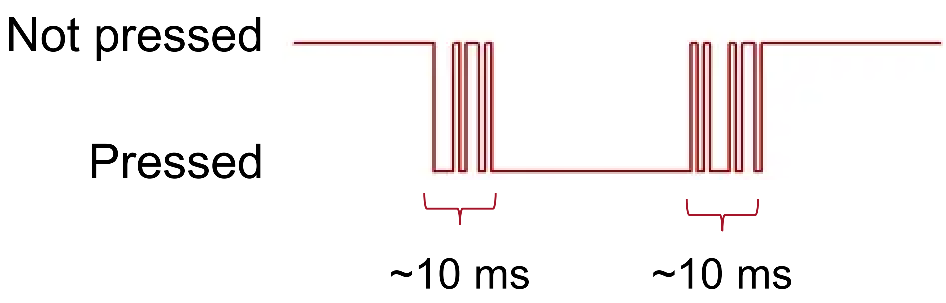
```c
int main(void) {
    setup();
    timersetup();
    sei();

    enum STATE cur_state = FORWARD;
    uint32_t now = 0;
    uint32_t last_ms = 0;

    while (1) {
        now = count_ms;                    // save current time (in case we get interrupted)
        switch (cur_state) {
            case FORWARD:
                if ((PIND & 0b00000100) == 0) {
                        cur_state = REVERSE;    // change state
                } else {
                    if ((now != last_ms) && (now % 1000) == 0) {
                        last_ms = now;
                        PORTB = (PORTB==0b00001000)?0b00100000:PORTB>>1; // shift LED
                    }
                }
                break;
            case REVERSE:
                if ((PIND & 0b00000100) == 0) {
                    cur_state = FORWARD;    // change state
                } else {
                    if ((now != last_ms) && (now % 1000) == 0) {
                        last_ms = now;
                        PORTB = (PORTB==0b00100000)?0b00001000:PORTB<<1; // shift LED
                    }
                }
                break;
        }
    }
}
```

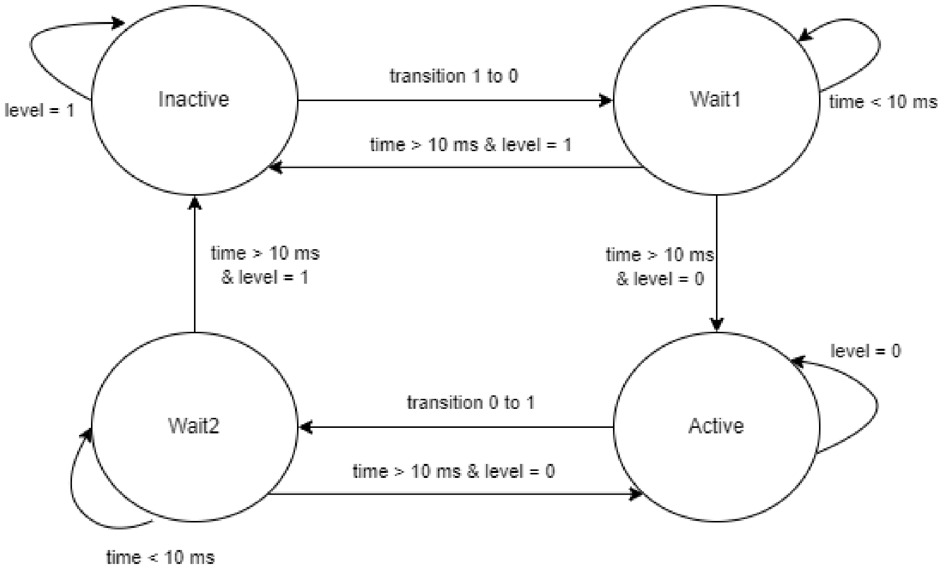
Button Debouncing





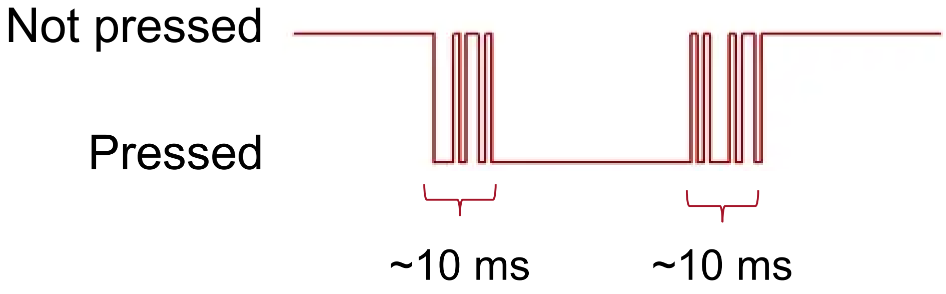
Button Debouncing

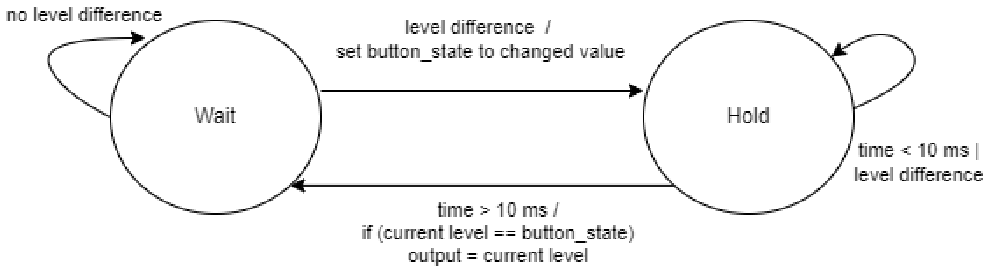




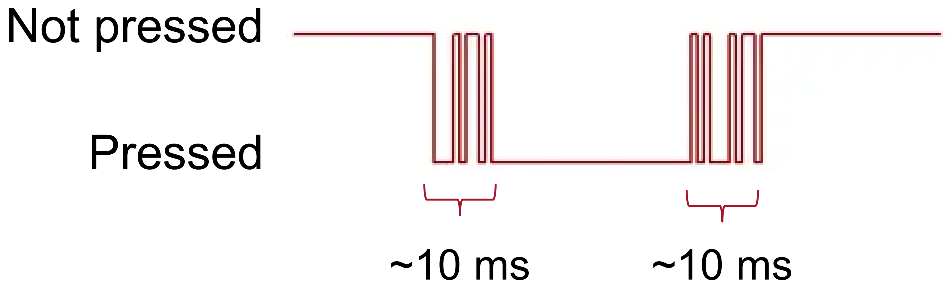

```
int main(void) {
    setup();    // set up the physical hardware
    while (1) {
        for(;;) {
            while (button_state == Inactive) {
                ;   // wait until the push button is pressed
            }
            delay_ms(10);
            if (button_state == Inactive) {
                // still pressed
                continue;
            }
            break;
        }
        PORTB = ~((++count)<<2);     // show count on LEDs
        for (;;) {
            while (button_state == Active) {
                ;   // wait until the push button is released
            }
            delay_ms(10);
            if (button_state == Active) {
                // still active
                continue;
            }
            break;
        }
    }
}
```

Button Debouncing

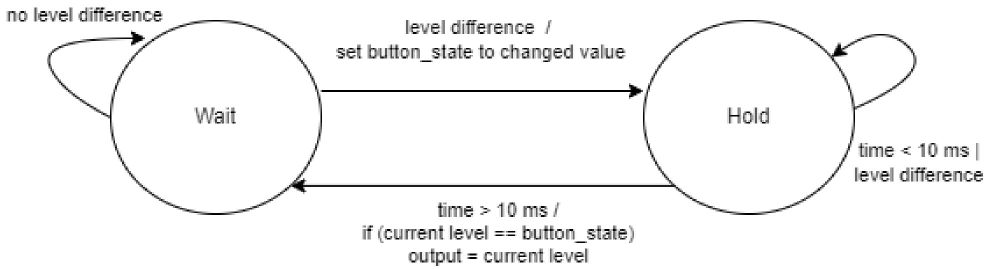




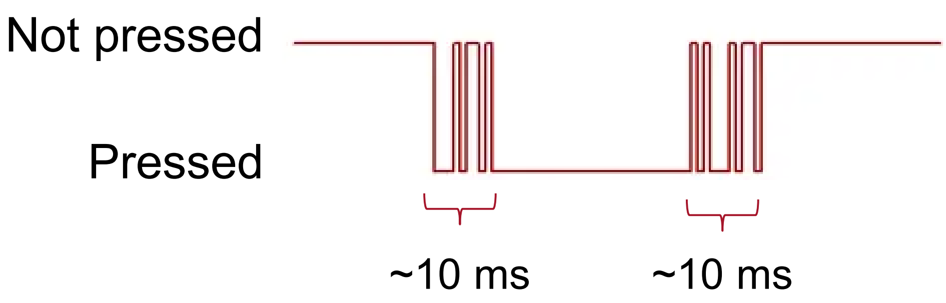
Button Debouncing



Input = 1

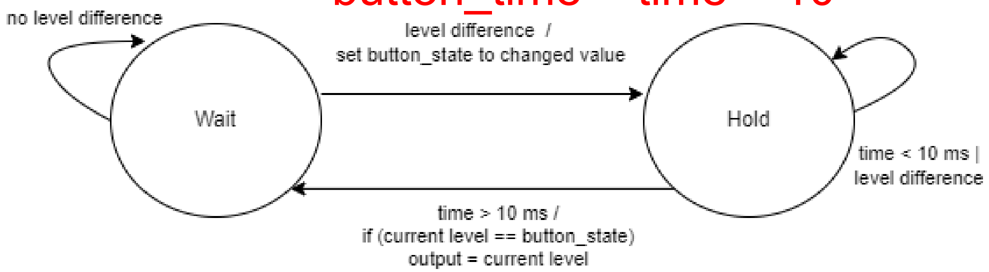


Button Debouncing

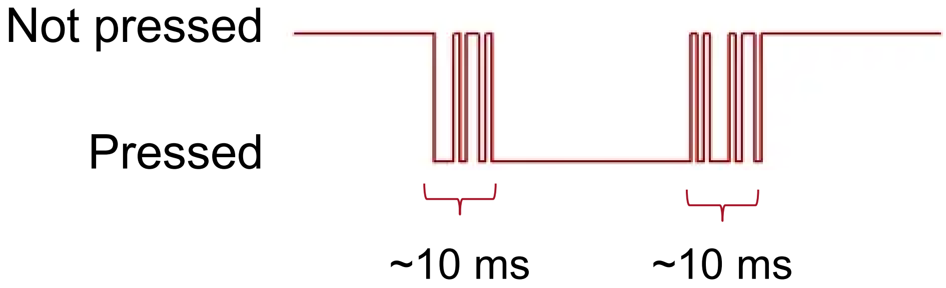


Input = 1

Input = 1 -> 0
button_state = 0
button_time = time + 10

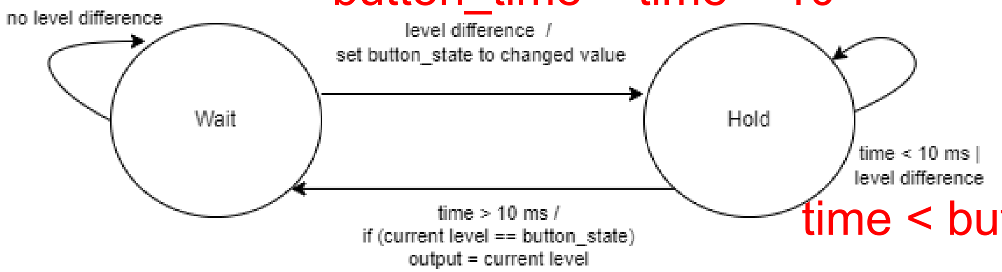


Button Debouncing



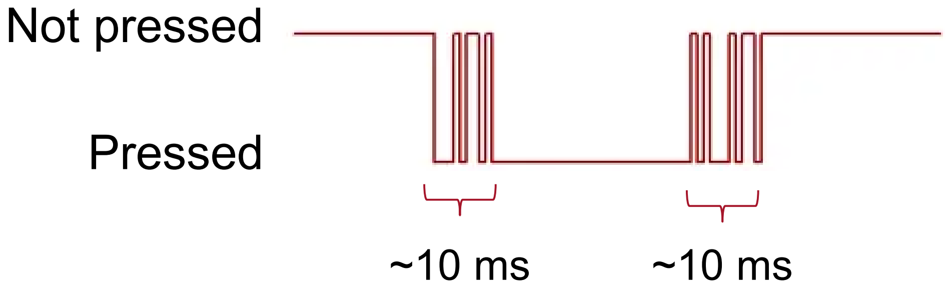
Input = 1

Input = 1 -> 0
button_state = 0
button_time = time + 10



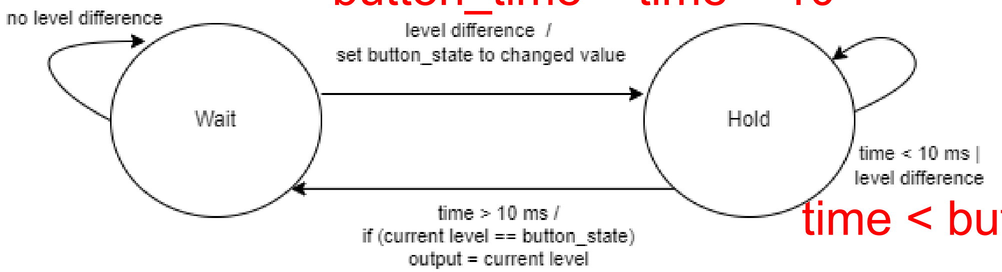
time < button_time

Button Debouncing



Input = 1

Input = 1 -> 0
button_state = 0
button_time = time + 10

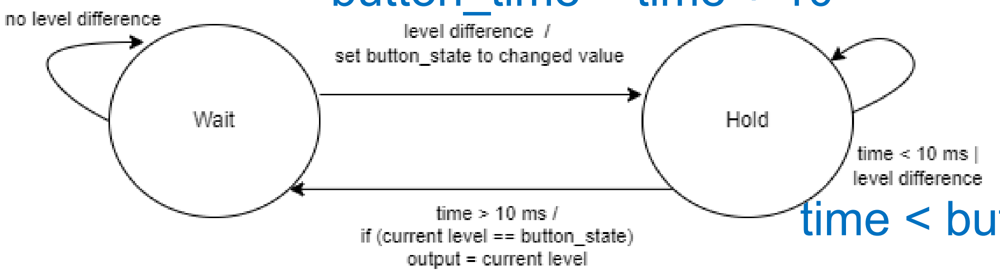


time < button_time

time > button_time
If Input == button_state => output = 0
Otherwise output = 1

Input = 0

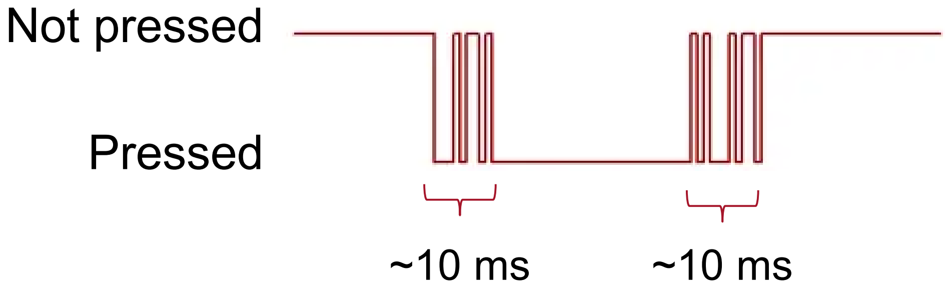
Input = 0 -> 1
button_state = 1
button_time = time + 10



time < button_time

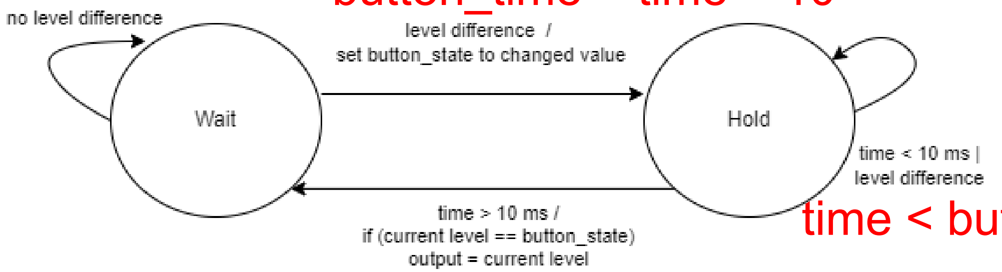
time > button_time
If Input == button_state => output = 1
Otherwise output = 0

Code



Input = 1

Input = 1 -> 0
button_state = 0
button_time = time + 10



time < button_time

time > button_time
If Input == button_state => output = 0
Otherwise output = 1

```c
int main(void) {
    setup();
    timersetup();
    sei();

    enum STATE cur_state = FORWARD;
    uint32_t now = 0;
    uint32_t last_ms = 0;
    uint32_t button_time = 0;
    uint8_t button_value = 0b00000100;
    uint8_t button_state = 0b00000100;
    uint8_t button = 0;

    while (1) {
        if (count_ms > button_time) {
            button_value = (PIND & 0b00000100);          // save current input state (in case we get interrupted)
            if ((button_value ^ button_state) != 0) {    // check if state has changed
                button_state = button_value;             // store changed state of button
                button_time = count_ms + 10;             // add debounce time
                button = (button_value==0)?1:0;          // set whether button is pressed or released
            }
        }

        now = count_ms;                                  // save current time (in case we get interrupted)
        switch (cur_state) {
            case FORWARD:
                if ((now > button_time) & (button == 1)) {
                    cur_state = REVERSE;     // change state
                    button = 0;
                } else {
                    if ((now != last_ms) && (now % 1000) == 0) {
                        last_ms = now;
                        PORTB = (PORTB==0b00001000)?0b00100000:PORTB>>1; // shift LED
                    }
                }
                break;
            case REVERSE:
                if ((now > button_time) & (button == 1)) {
                    cur_state = FORWARD;     // change state
                    button = 0;
                } else {
                    if ((now != last_ms) && (now % 1000) == 0) {
                        last_ms = now;
                        PORTB = (PORTB==0b00100000)?0b00001000:PORTB<<1; // shift LED
                    }
                }
                break;
        }
    }
}
```

Timer Summary

- Using a Timer
  - Choose an appropriate timer
  - Configure mode of operation
  - Setup prescaler
  - If using CTC mode, set compare register value
  - Setup interrupt and ISR, if desired
- One timer can be used to time multiple events in a system

Quiz 1

- Must be done on **Wednesday 15th March, 2023** before 11:55 pm
- Eight Questions: 4 MC, 4 short answer
- Duration: 30 minutes
- Make sure you have the datasheet readily available
- Pre-practical quizzes are practice questions for Quiz 1