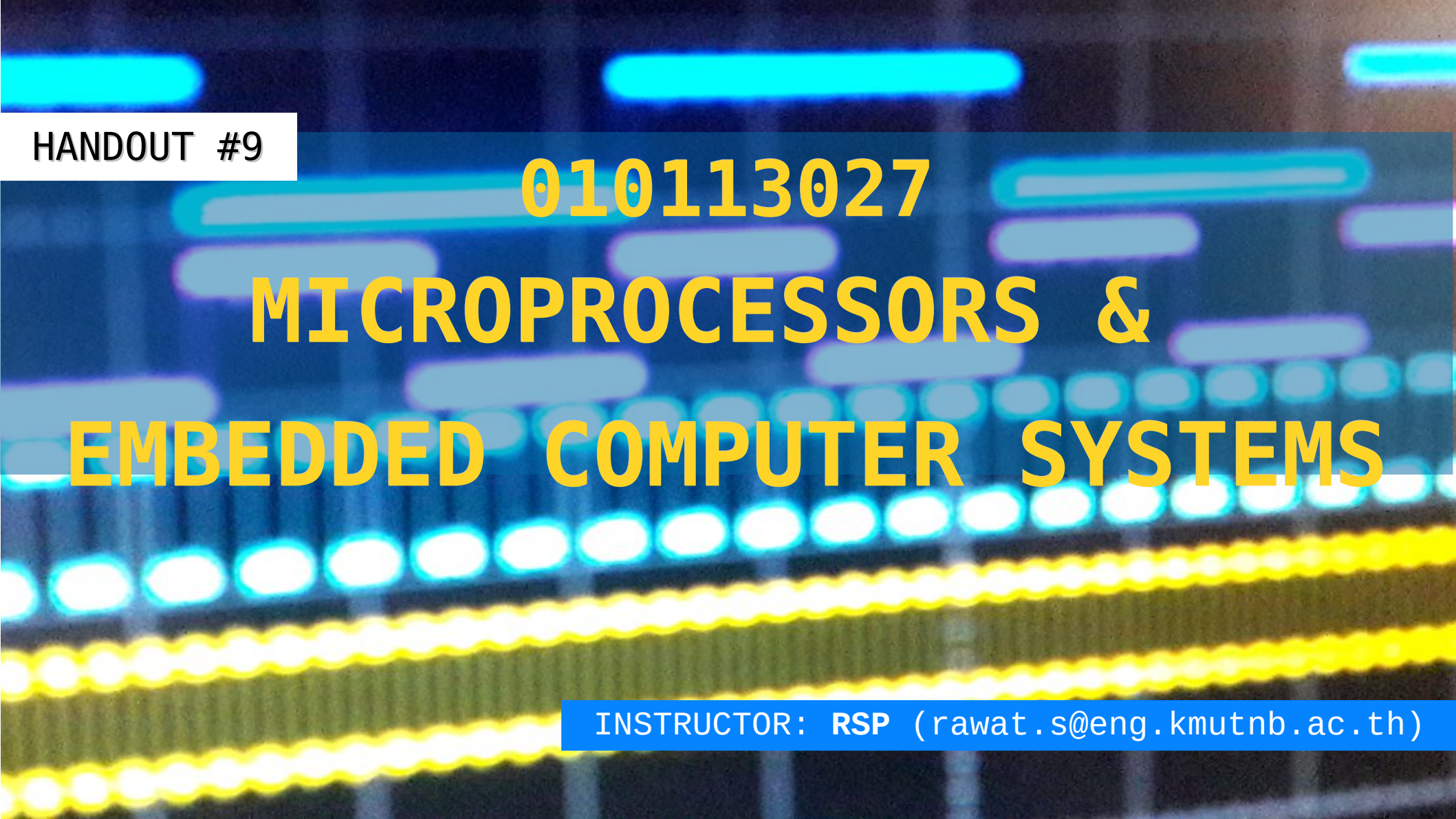


HANDOUT #9

010113027
MICROPROCESSORS & EMBEDDED COMPUTER SYSTEMS

INSTRUCTOR: **RSP** (rawat.s@eng.kmutnb.ac.th)

What We Will Learn...

- AVR Timer/Counter Modules
- Operation Modes of AVR Timers
- Interrupt-driven Timer Operations
- PWM Signal Generation
- Pulse Counting
- Bare-Metal C Code Examples
- Arduino API and AVR Timers

Timers vs. Counters

Timer/Counter

- In AVR microcontrollers, there are peripherals called **Timer/Counter modules**.
- The same hardware module can operate in two different ways: as a **timer** or as a **counter**, depending on the selected clock source.

Timer Mode

- A timer counts **internal** clock pulses at a constant rate.
- The clock source is derived from the CPU clock, typically through a **prescaler**.

Counter Mode

- A counter counts **external** events or pulses.
- The clock (trigger) source is provided by an external signal applied to the **T0** (for **Timer0**) or **T1** (for **Timer1**) pin.
- The external signal can be a periodic clock signal or any digital pulse signal.

Timer/Counter Units in ATmega328P

- The **ATmega328P** has three Timer/Counter units:
  - **Timer0** (8-bit), **Timer1** (16-bit) and **Timer2** (8-bit)
  - **ATmega2560**: two 8-bit timers (**Timer0**, **Timer2**) and four 16-bit timers (**Timer1**, **Timer3**, **Timer4**, **Timer5**)
- **Timer1** supports **Input Capture mode**.
  - Used for precise timing measurement of external events via the ICP1 pin.
- **Timer2** supports **asynchronous mode operation**.
  - In asynchronous mode, **Timer2** can use an external 32.768 kHz crystal oscillator connected to the **TOSC1** and **TOSC2** pins.
  - **Timer2** continues running independently of the CPU clock in Power-save mode. This allows periodic wake-up from sleep using **Timer2** overflow interrupts.

AVR Timer/Counter units used by **Arduino API**:

- **Timer0** (8-bit): The **millis() function** of the Arduino API uses Timer0 for timekeeping, **4 usec resolution**.
- **Timer1** (16-bit): Used by the **Arduino Servo library**.
- **Timer2** (8-bit): Used by the **tone() function** of the Arduino API.

Timer/Counter Units in ATmega328P

- **Clock selection block and prescaler**
- **Counter registers (TCNTn), n=0,1,2**
  - 8-bit (**Timer0** & **Timer2**) and 16-bit (**Timer1** only)
- **Control registers (TCCRnA, TCCRnB, TCCRnC)**
  - Used to configure the operation of a timer.
- **Output Compare registers (OCRnA, OCRnB)**
  - Compare Match event: The current timer value (**TCNTn**) becomes equal to the value stored in an **OCRnA** or **OCRnB** register.
  - If this event occurs, the CPU can
    - set a flag (**OCFnA** or **OCFnB**),
    - generate an interrupt,
    - toggle / clear / set an output pin (**OCnA** / **OcnB**), and
    - reset the counter (in CTC mode).
- **Interrupt Mask registers (TIMSKn)**
- **Interrupt Flag registers (TIFRn)**

Operation Modes of AVR Timers

- **Normal Mode**
  - In this mode, the counter register increments by one at each clock cycle according to the selected clock frequency.
  - If the timer is 8-bit wide, the counter counts from 0 to 255 and then rolls over back to 0.
  - This event is called a **Timer Overflow**, and an interrupt can be generated when this event occurs.
- **CTC Mode (Clear Timer on Compare Match)**
  - In this mode, the counter does not count up to its maximum value. Instead, it counts up to a predefined value stored in the **Output Compare Register** (**OCRn**).
  - When the counter value matches the compare value, it resets to 0.
  - An interrupt can also be generated on this compare match event.
- **PWM Modes: Fast-PWM Mode and Phase-Correct PWM**
  - This mode is used to generate **PWM** signals with adjustable duty cycles.

Clock Prescaler of AVR Timers

- Each timer module includes a **selectable prescaler circuit** used to divide the CPU clock frequency.
- Available prescaler values are: **1**, **8**, **64**, **256**, and **1024**
- The prescaler allows the timer to increment at a slower rate than the CPU clock frequency (e.g. 16MHz).
- What is the **longest** time interval for timer overflow ?
  - T_overflow = 2^n x prescaler / F_CPU, n=8 or 16
  - Choose prescaler = 1024:
    - 8-bit: T_overflow = 16.384 ms
    - 16-bit: T_overflow = 4.194304 sec

Types of Timer Interrupts

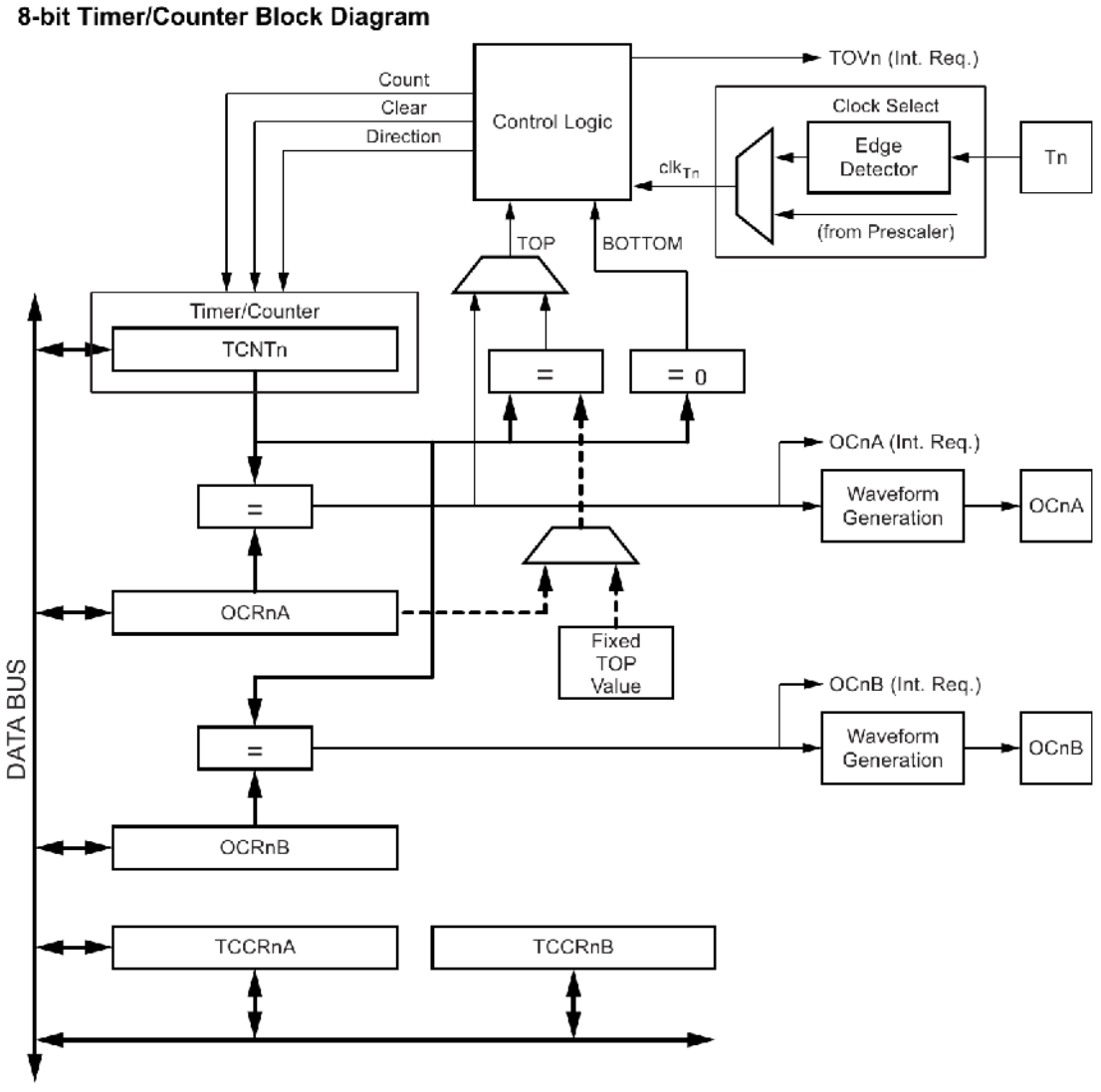
There are **different types of interrupts** from Timer/Counter units.

- **Timer Overflow interrupt**
  - Triggered when **TCNTn** rolls over from **TOP** (maximum value) to 0.
  - For a 8-bit timer: overflow from 0xFF to 0x00
  - For a 16-bit timer: overflow from 0xFFFF to 0x0000
- **Compare Match A interrupt**: **TCNTn** == **OCRnA**
- **Compare Match B interrupt**: **TCNTn** == **OCRnB**
- **Input Capture (Timer1 only)**:
  - Triggered when a selected edge (rising or falling) is detected on the **ICP1** pin.
  - The current value of **TCNT1** is copied into the **ICR1** register.

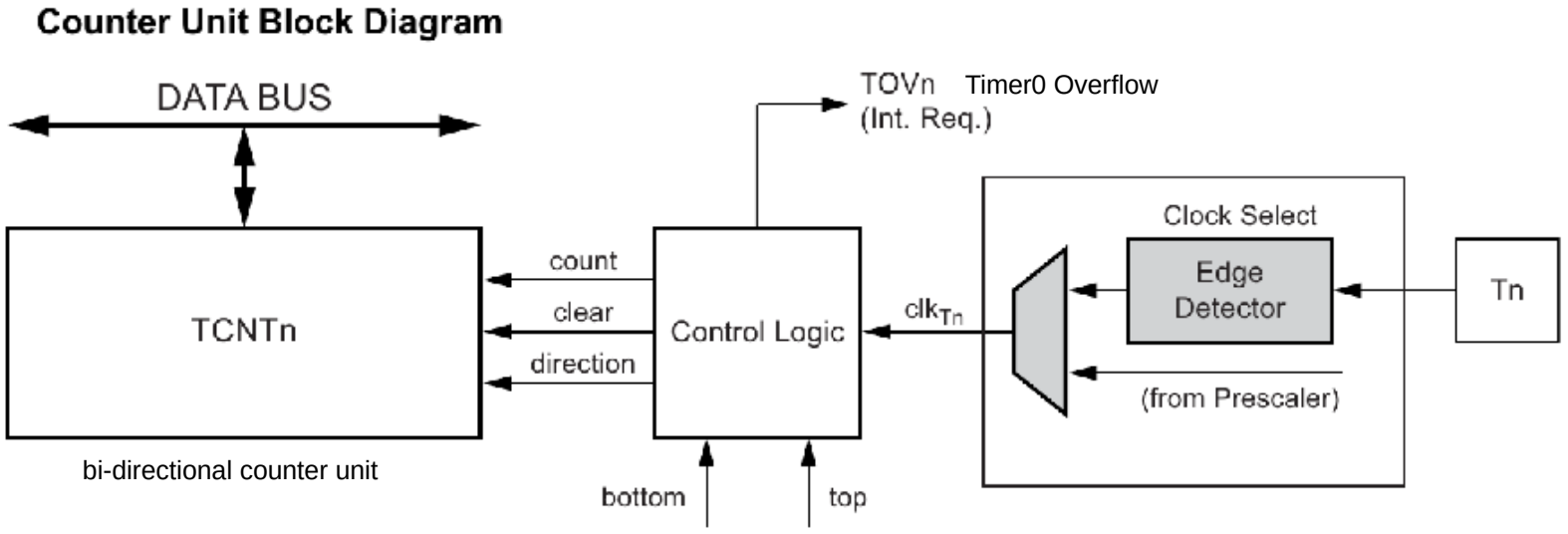
Timer0 Block Diagram

8-bit Timer/Counter 0

- **T0 pin**: used as an external clock pin or pulse source for **Timer0**
- **OC0A pin**: "Output Compare Match A"
- **OC0B pin**: "Output Compare Match B"
- **TCNT0**: 8-bit count register
- **TCCR0A & TCCR0B**: 8-bit configuration registers for **Timer0**

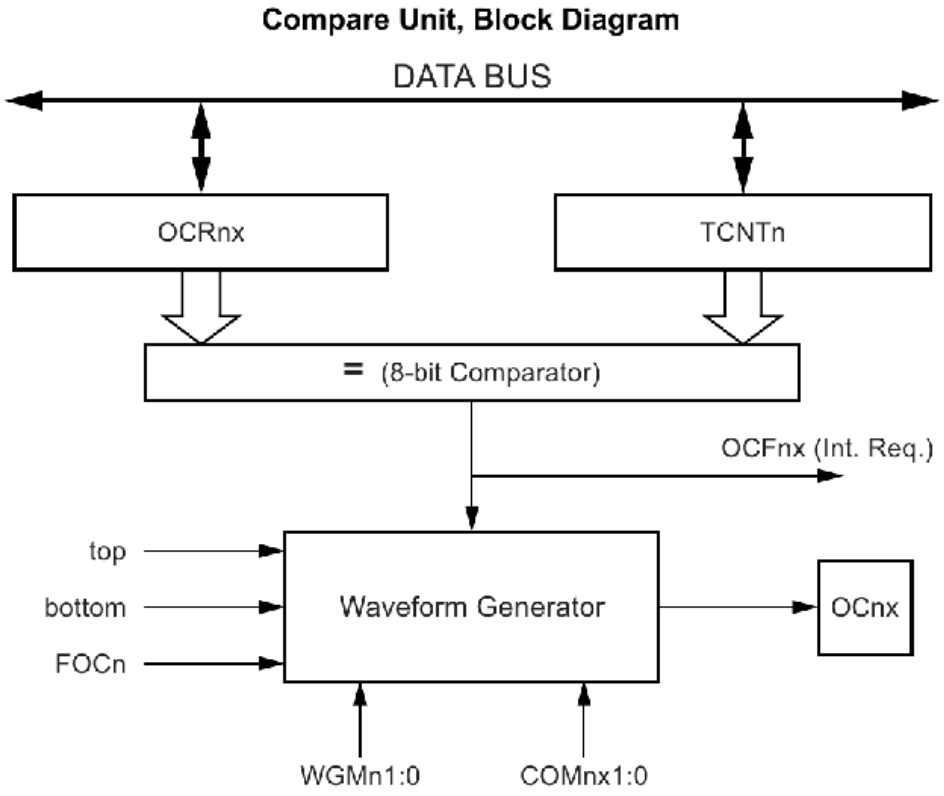


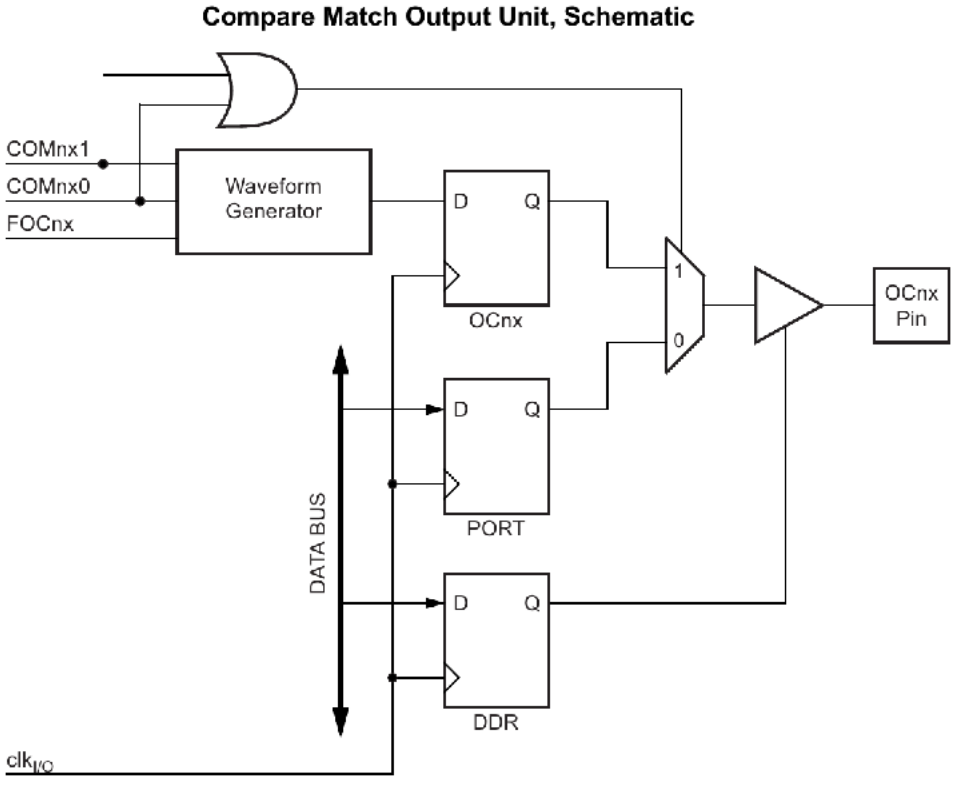
Timer0 Block Diagram



- The counter reaches the **BOTTOM** when it becomes **0x00**.
- The counter reaches its **MAX** (maximum) when it becomes **0xFF** (decimal 255).
- The counter reaches the **TOP** when it becomes equal to the **highest value** in the count sequence.
  - **TOP** can be assigned to be the fixed value **0xFF (MAX)** or the value stored in the **OCR0A** register.
  - The assignment is dependent on the mode of operation.

Timer0 Block Diagram





Timer0 Registers

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
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	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCNT0 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	TCNT0[7:0]								TCNT0
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	

- Waveform generator mode (**WGM02:0**) bits
- Clock select (**CS02:0**) bits

- Compare output mode **(COM0A1:0**) bits
- Compare output mode (**COM0B1:0**) bits
- Force output compare (**FOC0A and FOC0B**) bits

Timer0 Registers

OCR0A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	OCR0A[7:0]								OCR0A
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	

OCR0B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	OCR0B[7:0]								OCR0B
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	

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TIFR0 – Timer/Counter 0 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer0 Clock Selection Bits

Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(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 T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Timer0 Waveform Generation Mode Bits

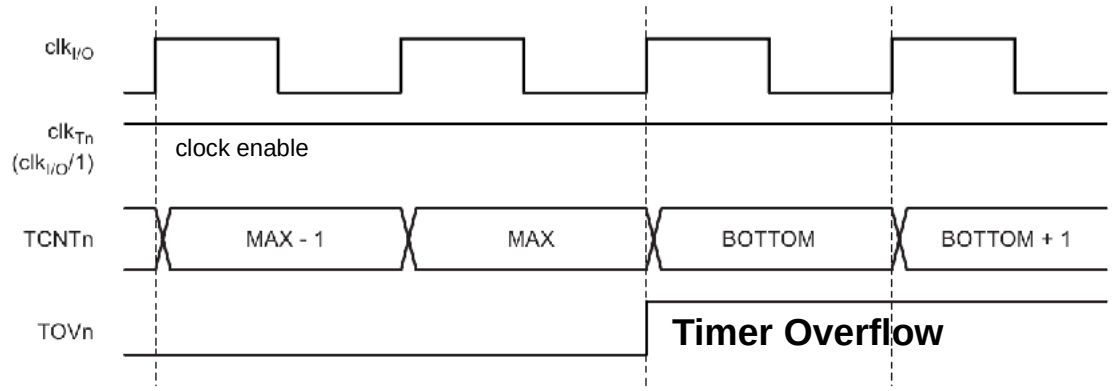
Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[(1)(2)]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

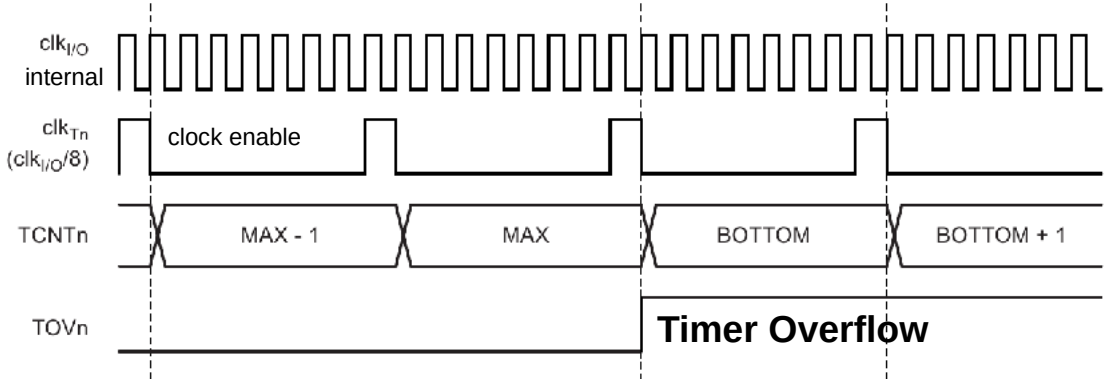
Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

Timer0 Overflow

Timer/Counter Timing Diagram, no Prescaling

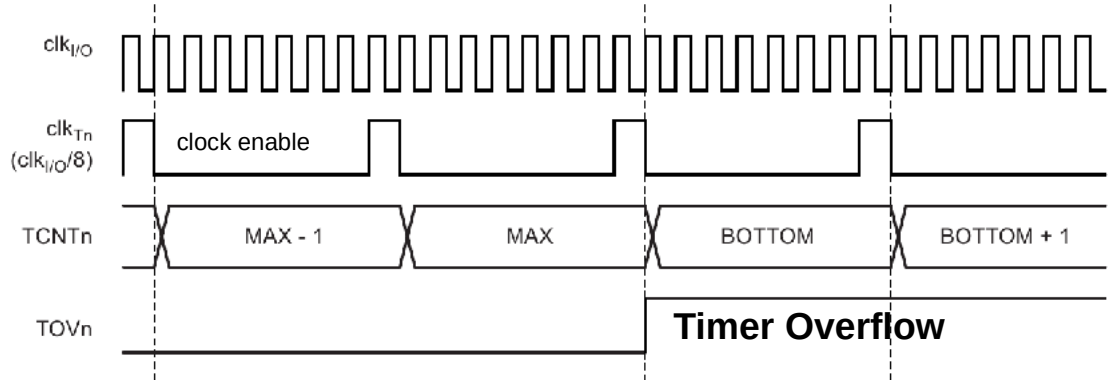


Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)

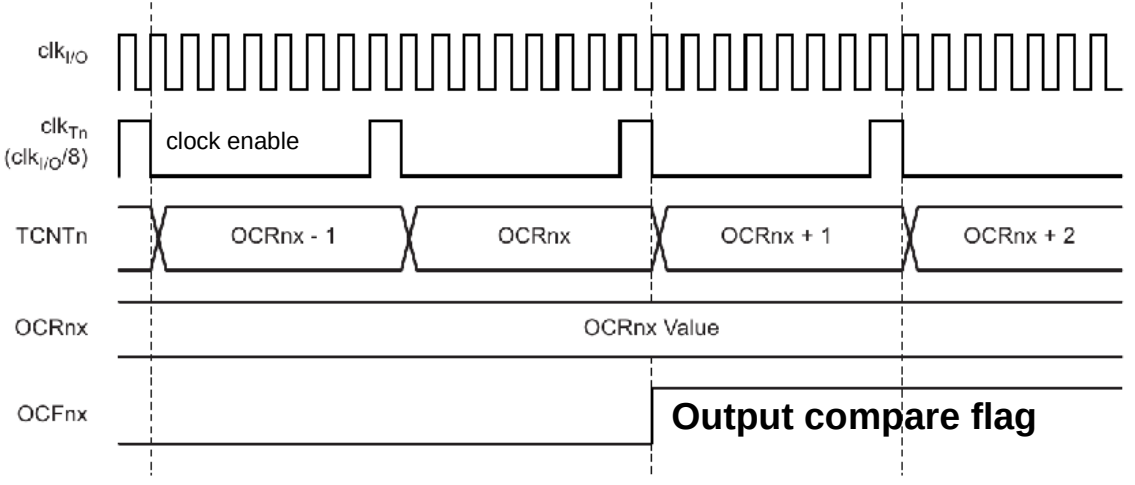


Timer0 Overflow vs. Compare Match

Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)

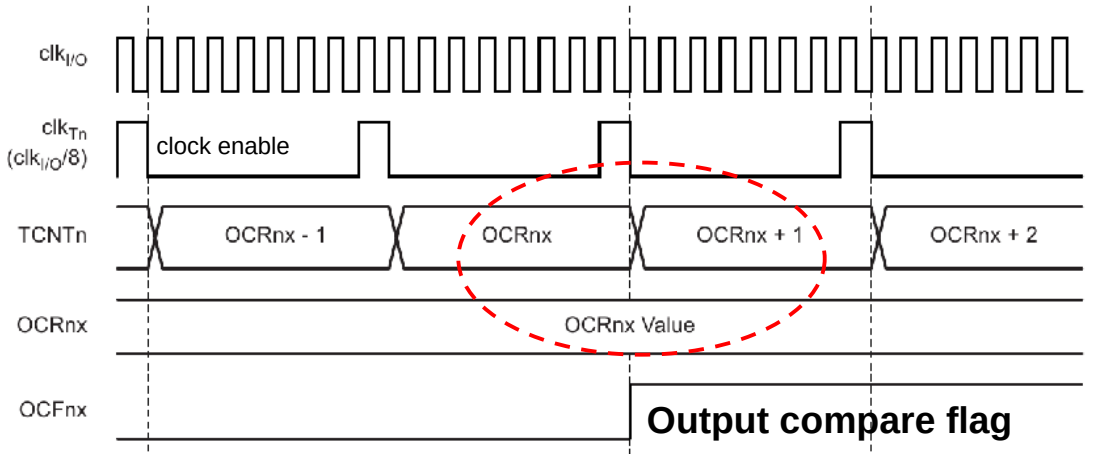


Timer/Counter Timing Diagram, Setting of OCF0x, with Prescaler ($f_{clk\_I/O}/8$)

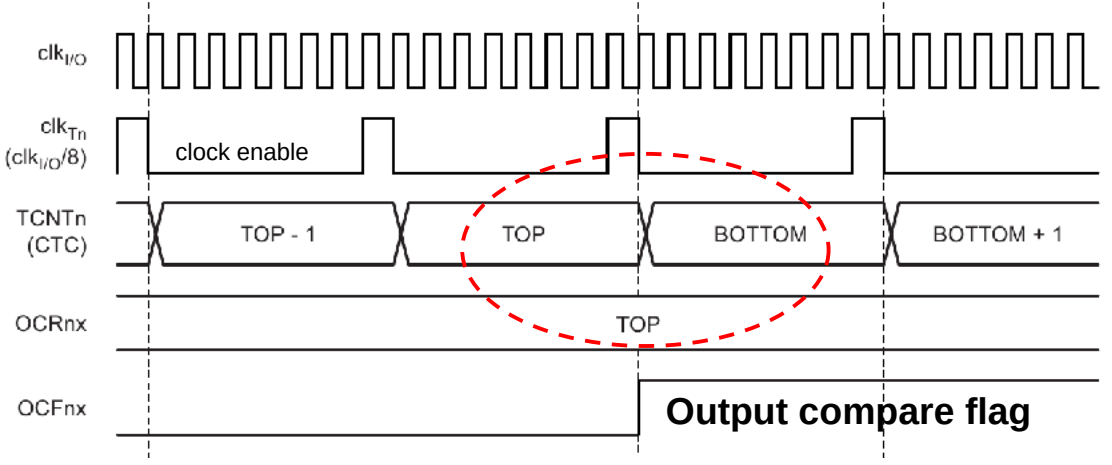


Timer0 Overflow vs. Compare Match

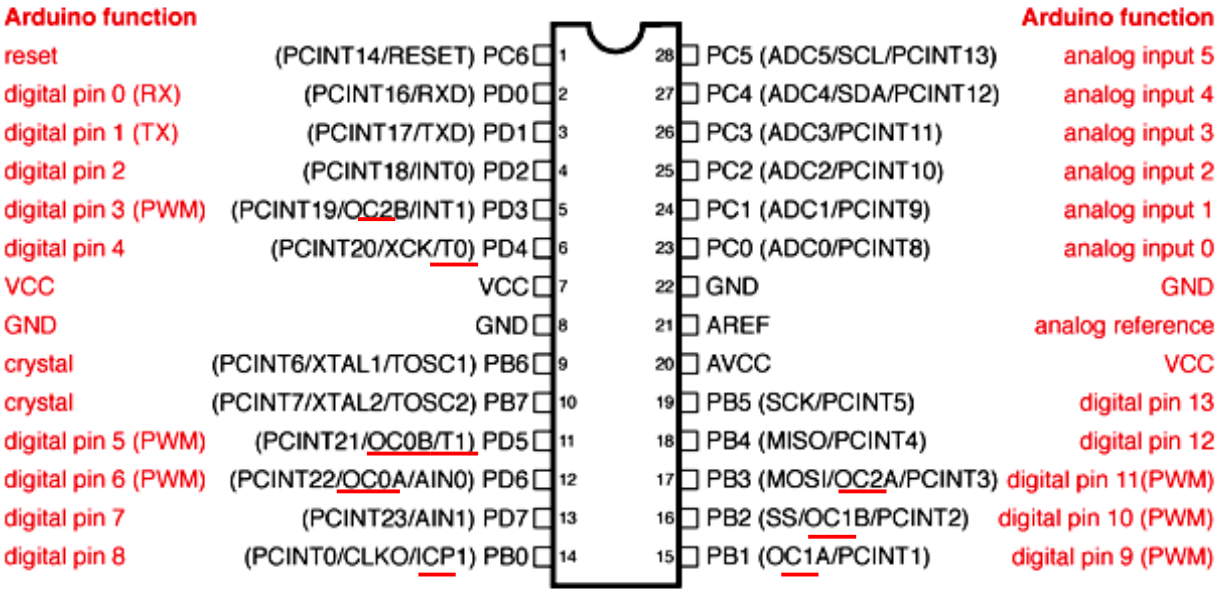
Timer/Counter Timing Diagram, Setting of OCF0x, with Prescaler ($f_{clk\_I/O}/8$)



Timer/Counter Timing Diagram, Clear Timer on Compare Match mode, with Prescaler ($f_{clk\_I/O}/8$)



Arduino Uno / Nano Pins



Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

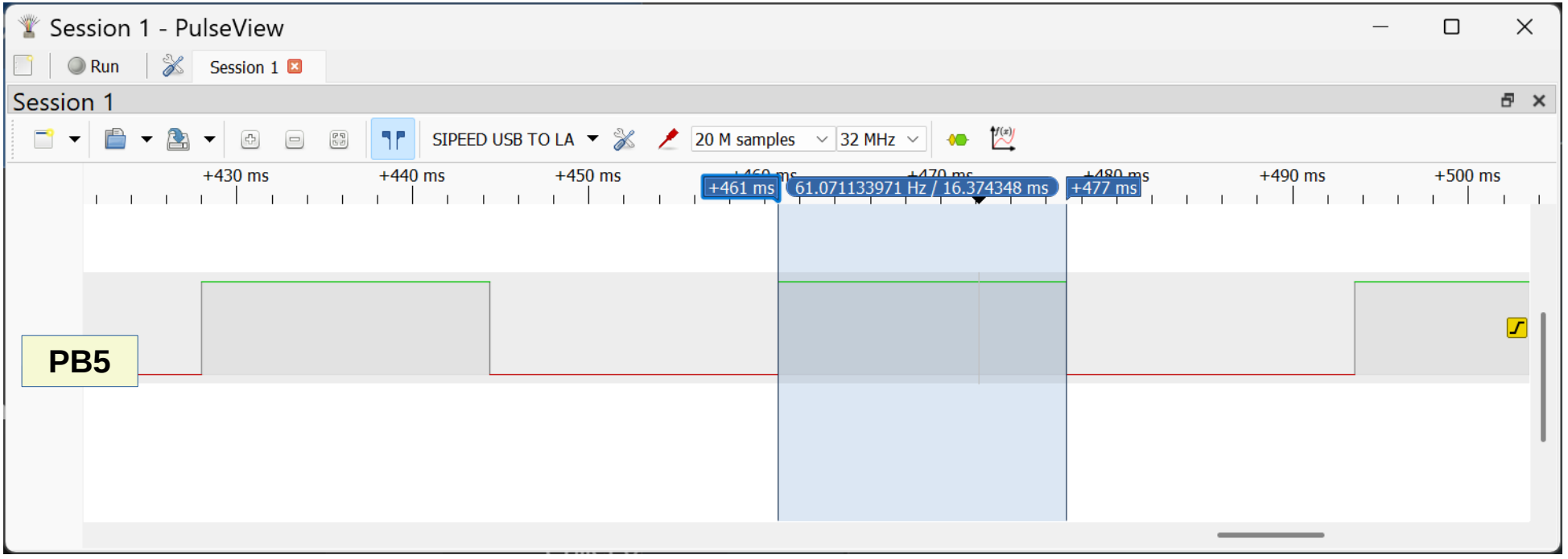
C Code: Timer0 in Normal Mode + Overflow Interrupt

```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>

void gpio_init(void) {
  // PB5 (Arduino D13) as output
  DDRB |= (1<<DDB5);
}

void timer0_init(void) {
  TCCR0A = TCCR0B = 0x00; // Normal mode
  TCNT0 = 0;              // Clear counter
  TIMSK0 |= (1<<TOIE0);   // Enable overflow interrupt
  // Set prescaler = 1024, start Timer0
  TCCR0B |= (1<<CS02) | (1<<CS00);
}
```

```c
ISR(TIMER0_OVF_vect) {
  PINB = (1<<PINB5); // Toggle PB5 pin
}

int main(void) {
  gpio_init();
  timer0_init();
  sei(); // Enable global interrupts
  while (1) { }
}
```

Toggle Rate = 16MHz / 1024 / 256
= 61.035 Hz
T = 16.384 msec

USB Logic Analyzer: Output Waveform



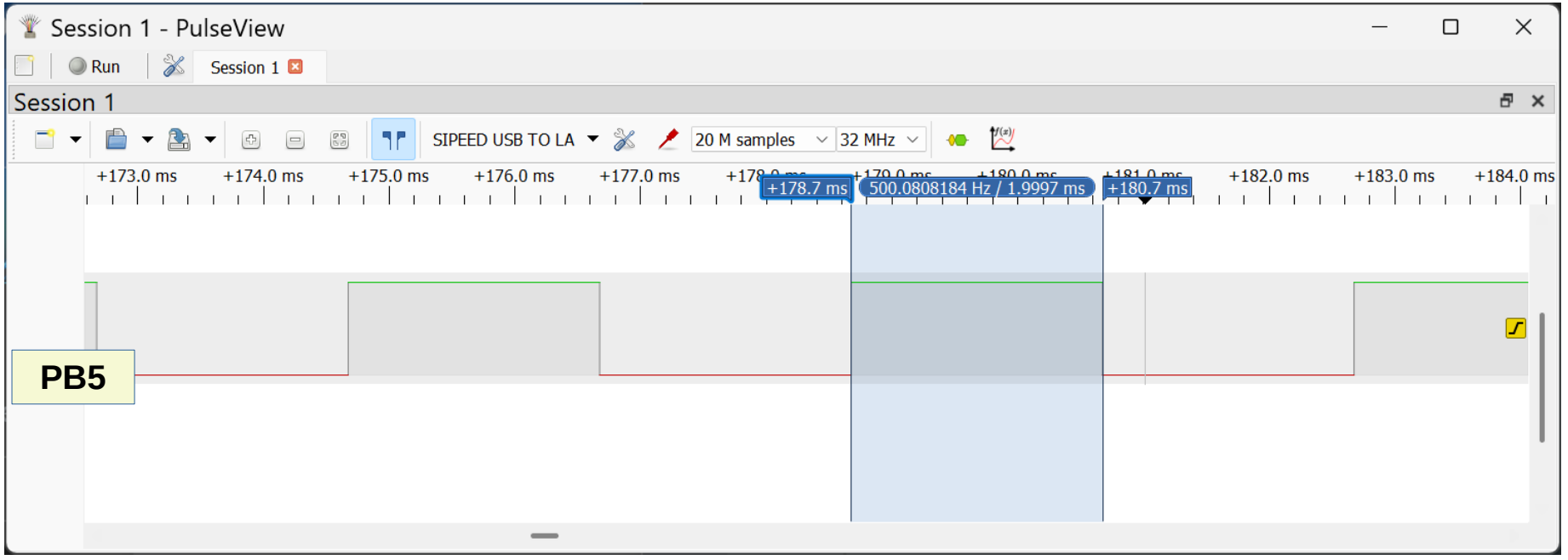
C Code: Timer0 in CTC Mode + Compare Match A Interrupt

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>

void gpio_init(void) {
  // PB5 (Arduino D13) as output
  DDRB |= (1<<DDB5);
}

// 16MHz/256/125 = 62500Hz/125 = 500Hz
void timer0_init(void) {
  TCCR0A = TCCR0B = 0;
  // CTC mode (WGM01=1)
  TCCR0A |= (1<<WGM01);
  // Set compare match A value (TOP)
  OCR0A = (125-1);
  // Enable Compare Match A interrupt
  TIMSK0 |= (1<<OCIE0A);
  // Prescaler = 256
  TCCR0B |= (1<<CS02);
}
```

```
ISR(TIMER0_COMPA_vect) {
  PINB = (1<<PINB5); // Toggle PB5
}

int main(void) {
  gpio_init();
  timer0_init();
  sei();  // Enable global interrupts
  while (1) {}
}
```

Toggle Rate = 16MHz / 256 / 125
= 500 Hz
T = 2 msec

USB Logic Analyzer: Output Waveform



C Code: Timer0 in CTC Mode + Compare Match B Interrupt

```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>

void gpio_init(void) {
  // PB5 (Arduino D13) as output
  DDRB |= (1<<DDB5);
}

void timer0_init(void) {
  TCCR0A = TCCR0B = 0;
  // CTC mode (WGM01 = 1, TOP = OCR0A)
  TCCR0A |= (1<<WGM01);
  // TOP value (set the period)
  OCR0A = 125-1;
  // Set Compare Match B value
  OCR0B = 0; // Can be any value between 0..255
  // Enable Compare Match B interrupt
  TIMSK0 |= (1<<OCIE0B);
  // Prescaler = 256
  TCCR0B |= (1<<CS02);
}
```

```c
ISR(TIMER0_COMPB_vect) {
  PINB = (1<<PINB5); // Toggle PB5 pin
}

int main(void) {
  gpio_init();
  timer0_init();
  sei();
  while (1) {}
}
```

Toggle Rate = 16MHz / 256 / 125
= 500 Hz
T = 2 msec

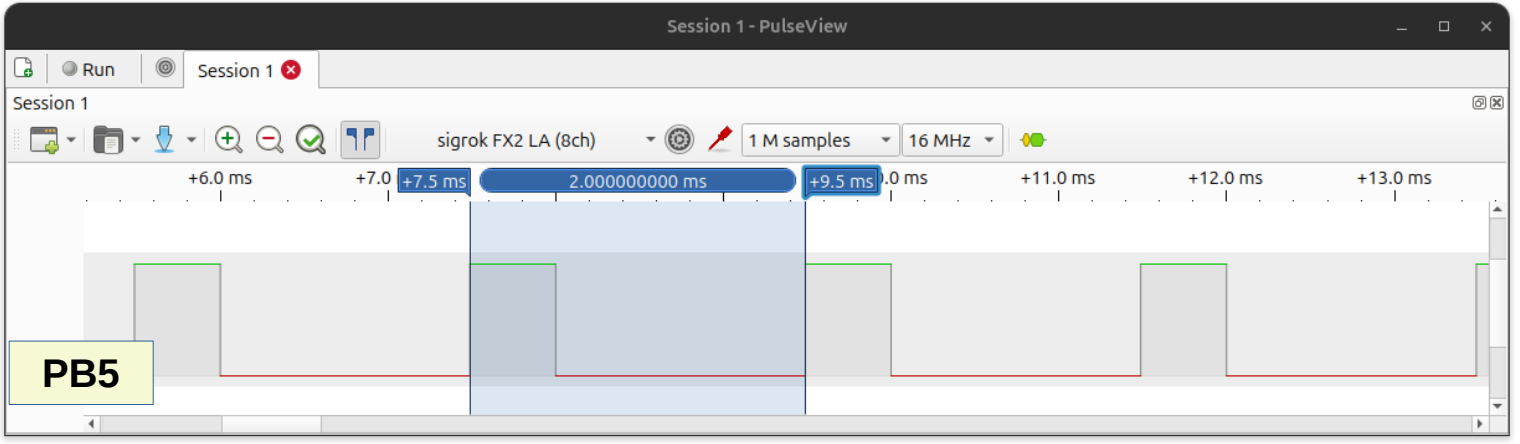
C Code: Timer0 in CTC Mode + Compare Match A & B Interrupt

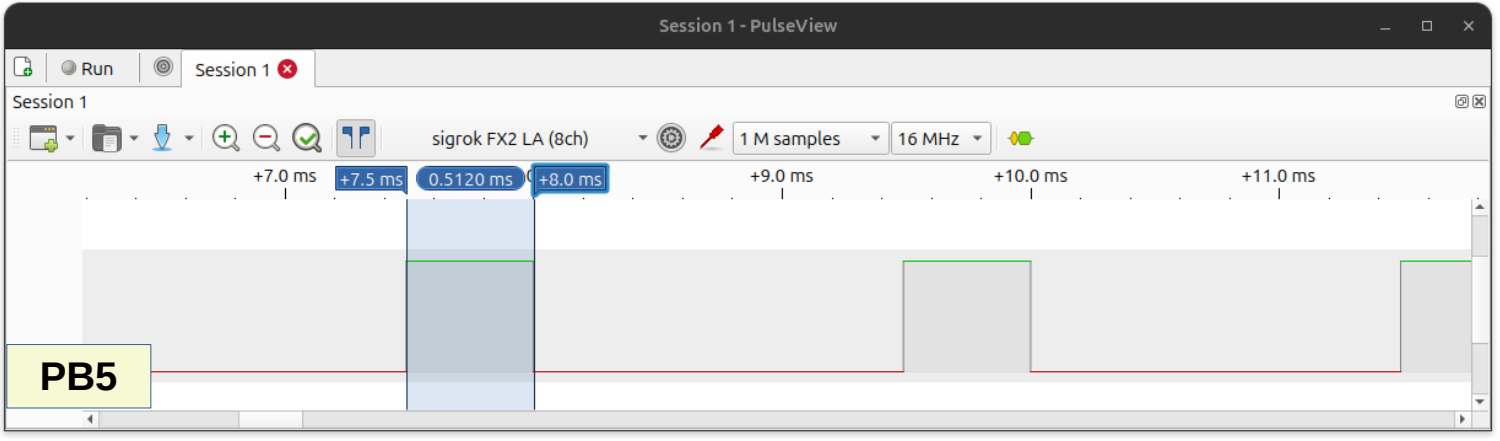
```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>

void gpio_init(void) {
  DDRB |= (1<<DDB5); // PB5/D13 output
}

#define PERIOD (125)
void timer0_init(void) {
  TCCR0A = TCCR0B = 0;
  // CTC mode (WGM01 = 1, TOP = OCR0A)
  TCCR0A |= (1<<WGM01);
  // Set TOP value => 500 Hz timer cycle
  OCR0A = PERIOD - 1;
  // Set Compare Match B value
  OCR0B = PERIOD/4; // 25% duty cycle
  // Enable Compare Match A AND B interrupts
  TIMSK0 |= (1<<OCIE0A) | (1<<OCIE0B);
  // Prescaler = 256
  TCCR0B |= (1<<CS02);
}
```

```c
ISR(TIMER0_COMPA_vect) {
  PORTB |= (1<<PINB5); // PB5 High
}

ISR(TIMER0_COMPB_vect) {
  PORTB &= ~(1<<PINB5); // PB5 Low
}
int main(void) {
  gpio_init();
  timer0_init();
  sei();
  while (1) {}
}
```

Toggle Rate = 16MHz / 256 / 125
= 500 Hz
T = 2 msec

USB Logic Analyzer: Output Waveform





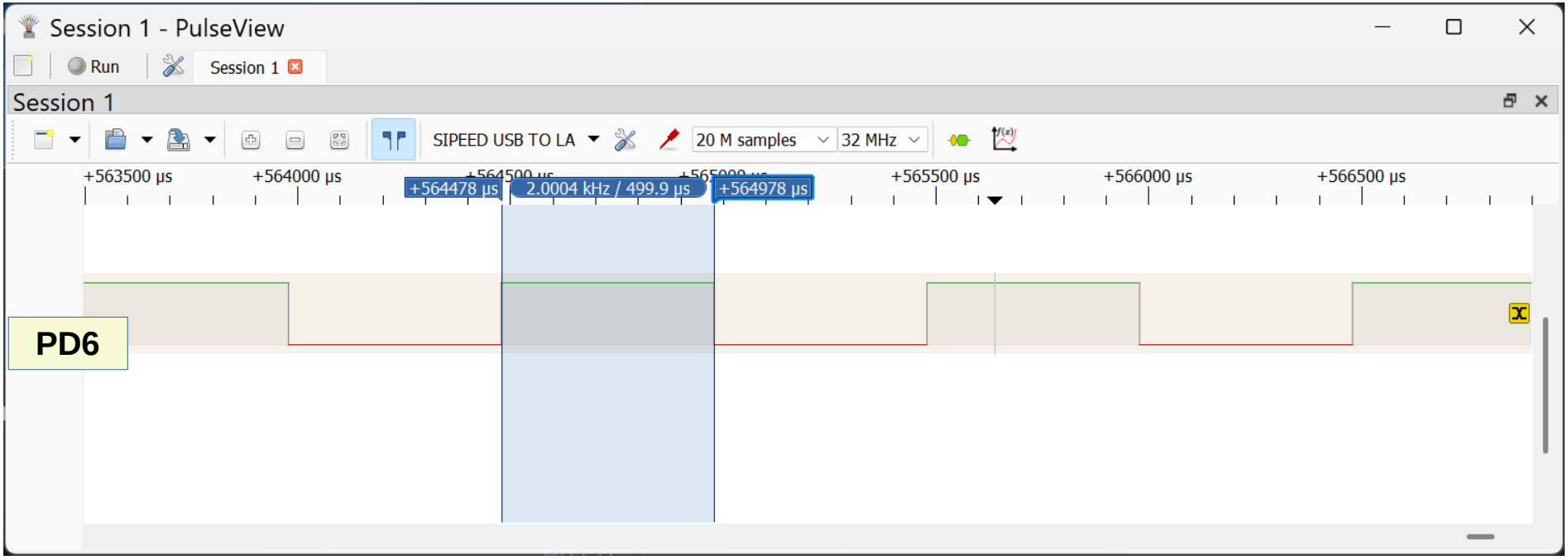
C Code: Timer0 in CTC Mode with Compare Match A + OC0A Toggle

```c
#define F_CPU 16000000UL
#include <avr/io.h>

void gpio_init(void) {
  // Must use OC0A = PD6 (Arduino D6) pin as output
  DDRD |= (1<<DDD6);
}

void timer0_init(void) {
  TCCR0A = TCCR0B = 0;
  // CTC mode (WGM01 = 1)
  TCCR0A |= (1<<WGM01);
  // Toggle OC0A pin on Compare Match A (TOP)
  TCCR0A |= (1<<COM0A0);
  // Set Compare Match A value (TOP)
  OCR0A = (125-1);
  // Prescaler = 64
  TCCR0B |= (1<<CS01) | (1<<CS00);
}
```

```c
int main(void) {
  gpio_init();
  timer0_init();
  while (1) { }
}
```

Toggle Rate = 16MHz / 64 / 125
= 2000 Hz
T = 500 usec

USB Logic Analyzer: Output Waveform



C Code: Timer0 in CTC Mode with Compare Match + OC0B Toggle

```c
#define F_CPU 16000000UL
#include <avr/io.h>

void gpio_init(void) {
  // Must use OC0B = PD5 (Arduino D5) as output
  DDRD |= (1<<DDD5);
}

void timer0_init(void) {
  TCCR0A = TCCR0B = 0;
  // CTC mode (WGM01 = 1)
  TCCR0A |= (1<<WGM01);
  // Toggle OC0B pin on Compare Match B
  TCCR0A |= (1<<COM0B0);
  // Set Compare Match A (TOP) value
  OCR0A = (125-1);
  // Set Compare Match B value
  OCR0B = OCR0A;
  // Prescaler = 64
  TCCR0B |= (1<<CS01) | (1<<CS00);
}
```

```c
int main(void) {
  gpio_init();
  timer0_init();
  while (1) { }
}
```

Toggle Rate = 16MHz / 64 / 125
= 2000 Hz
T = 500 usec

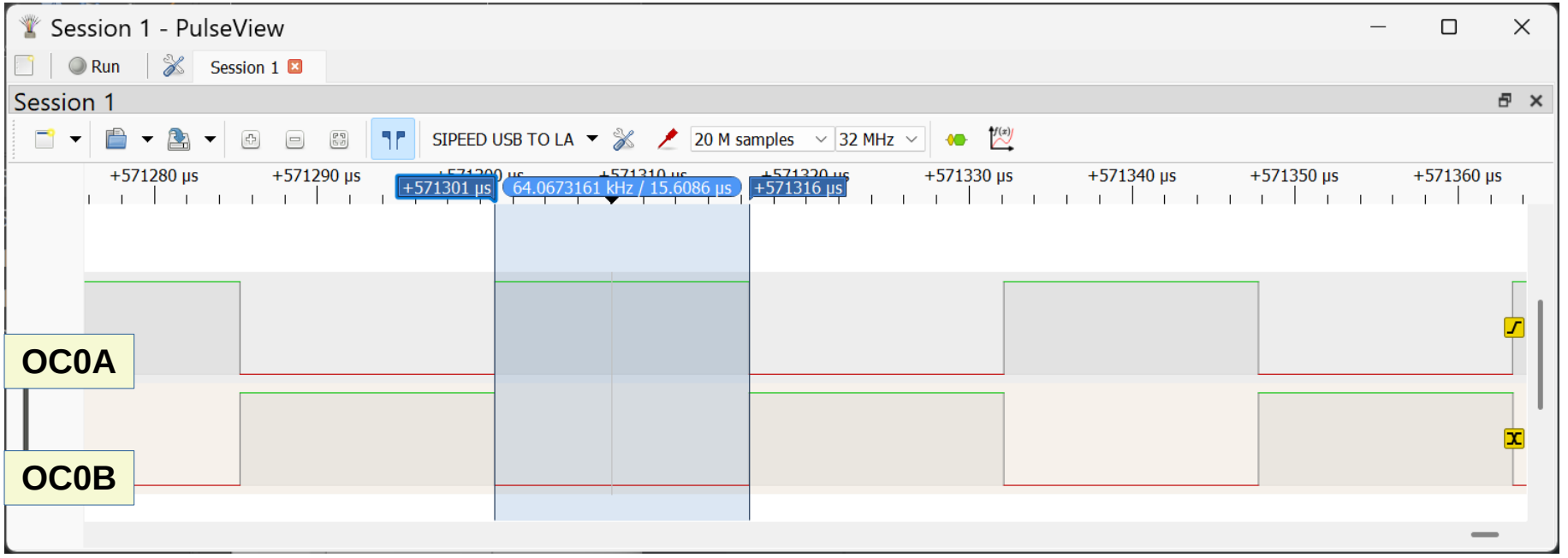
C Code: Timer0 in CTC Mode, Compare Match + OC0A & OC0B Toggle

```c
#define F_CPU 16000000UL
#include <avr/io.h>

void gpio_init(void) {
  // Use OC0A and OC0B pins as output
  DDRD |= (1<<DDD5) | (1<<DDD6);
}
void timer0_init(void) {
  TCCR0A = TCCR0B = 0;
  // CTC mode (WGM01 = 1)
  TCCR0A |= (1 << WGM01);
  // Enable toggle on OC0A and OC0B pins
  TCCR0A |= (1<<COM0A0) | (1<<COM0B0);
  // Set Compare Match A (TOP)
  OCR0A = OCR0B = 250-1;
  // Prescaler=1, Force Output Compare on OC0A
  // Immediately toggles OC0A once
  TCCR0B |= (1<<CS00) | (1<<FOC0A);
}
```

```c
int main(void) {
  gpio_init();
  timer0_init();
  while (1) {}
}
```

Toggle Rate = 16MHz / 1 / 250
= 64 000 Hz
T = 15.625 usec

USB Logic Analyzer: Output Waveform



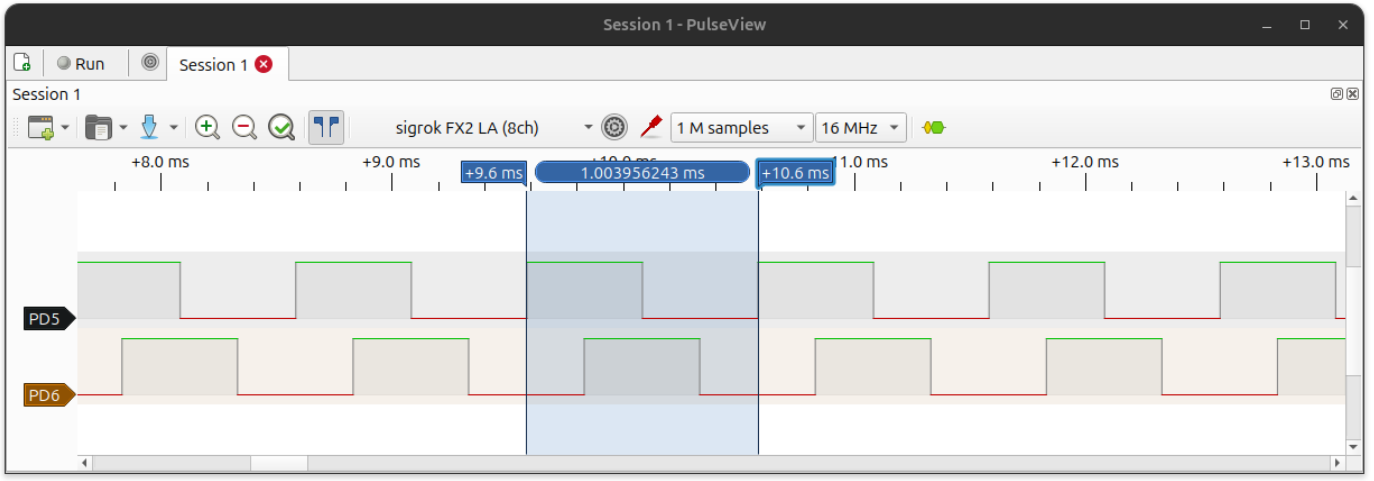
C Code: Timer0 in CTC Mode, Compare Match + OC0A & OC0B Toggle

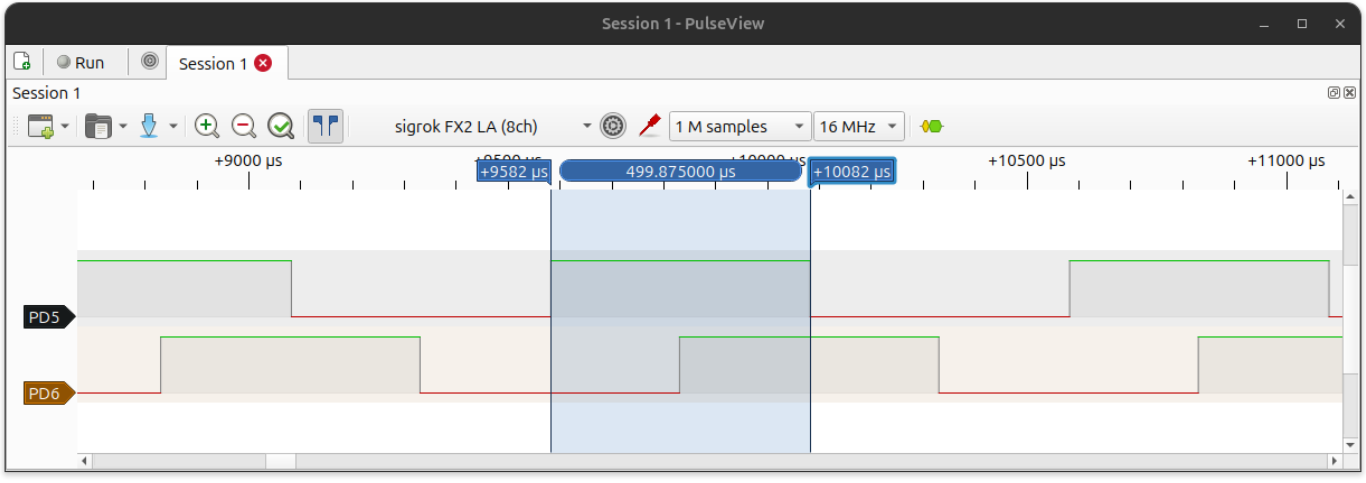
```c
#define F_CPU 16000000UL
#include <avr/io.h>

#define TOP (125-1)

void timer0_init(void) {
  // OC0A/PD6 and OC0B/PD5 output
  DDRD |= (1<<DDD6) | (1<<DDD5);
  // Stop timer first
  TCCR0A = TCCR0B = 0;
  // CTC mode (WGM01 = 1)
  TCCR0A |= (1<<WGM01);
  // Toggle OC0A and OC0B on Compare Match
  TCCR0A |= (1<<COM0A0) | (1<<COM0B0);
  // Set Compare Match A and B values
  OCR0A = TOP;
  OCR0B = TOP / 2;
  // Prescaler = 64 (start timer)
  TCCR0B |= (1<<CS01) | (1<<CS00);
}
```

```c
int main(void) {
  timer0_init();
  while (1) {}
}
```

Toggle Rate = 16MHz / 64 / 125
= 2000 Hz
T = 500 usec

USB Logic Analyzer: Output Waveform



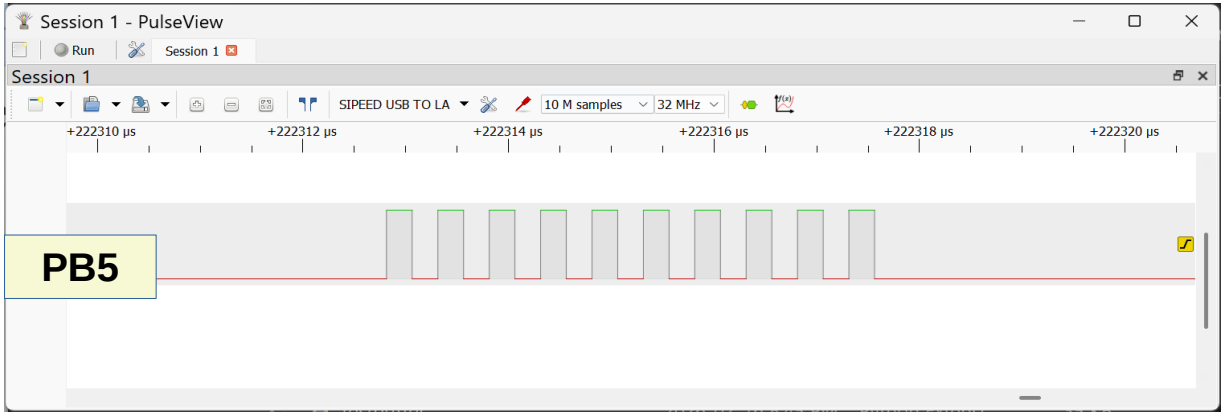


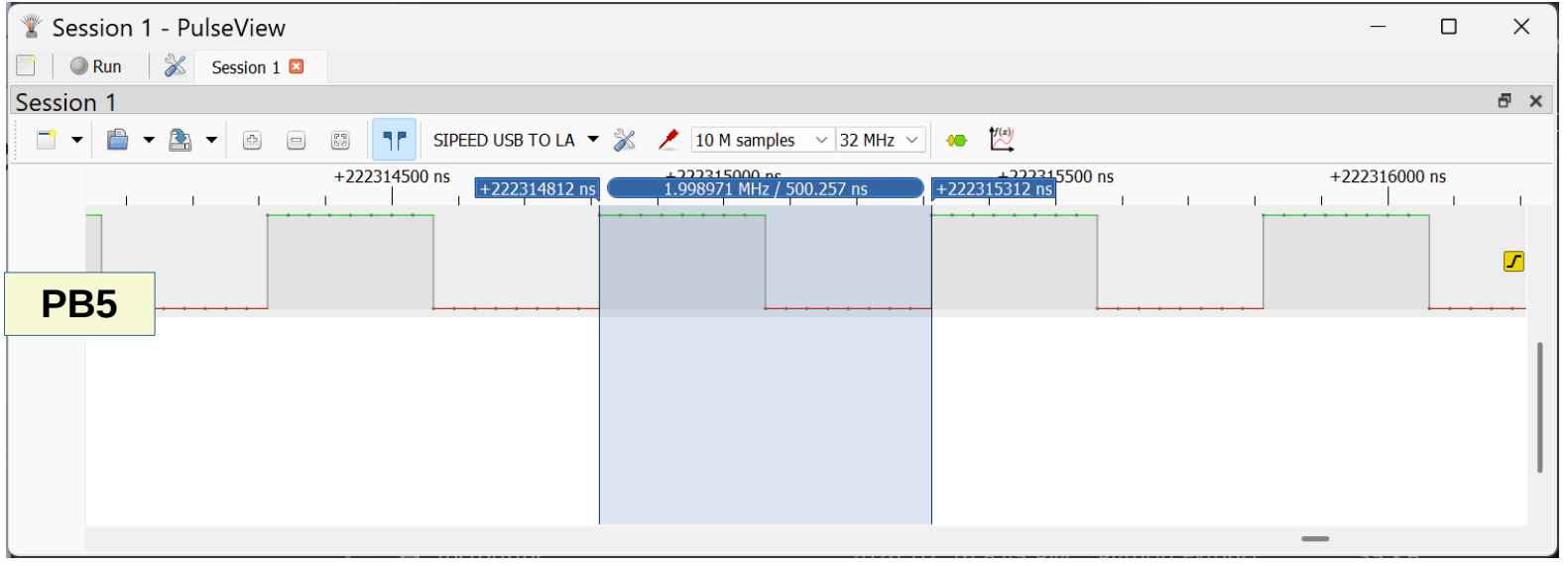
C Code: Timer0 Pulse Counting

```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>

extern void usart_init(void);
extern void usart_print(const char *s);

#define NUM_PULSES 10

// PB5 (D13) --- wire jumper --- PD4 (T0/D4)
void gpio_init() {
  // PB5 as output
  DDRB |= (1<<DDB5);
  // PD4 (T0) as input
  DDRD &= ~(1<<DDD4);
}

// TIMER0 as External Counter
void timer0_init(void) {
  TCCR0A = 0x00; // Normal mode
  // External Clock on T0 pin, Rising Edge
  TCCR0B = (1<<CS02)|(1<<CS01)|(1<<CS00);
}
```

```c
void generate_pulses(uint8_t num_pulses) {
  uint8_t n = 2*num_pulses;
  // output frequency: 2MHz
  for (uint8_t i=0; i < n; i++) {
    PINB = (1<<PB5);
  }
}

int main(void) {
  char sbuf[20];
  gpio_init();
  usart_init();
  timer0_init();
  usart_print("Pulse Counting using Timer 0\r\n");
  _delay_ms(1000);
  while (1) {
    TCNT0 = 0; // Reset counter
    generate_pulses(NUM_PULSES);
    _delay_ms(1);
    uint8_t counted = TCNT0;
    snprintf(sbuf, sizeof(sbuf),
             "Count=%u\r\n", counted);
    usart_print(sbuf);
    _delay_ms(100);
  }
}
```

USB Logic Analyzer: Output Waveform





Timer in Fast PWM Mode

On AVR devices like the ATmega328P, **Timer0 Fast PWM** has two important sub-modes depending on what defines **TOP**.

1) Fast PWM: WGM02:0 = 0b011 (Mode 3), TOP = 0xFF

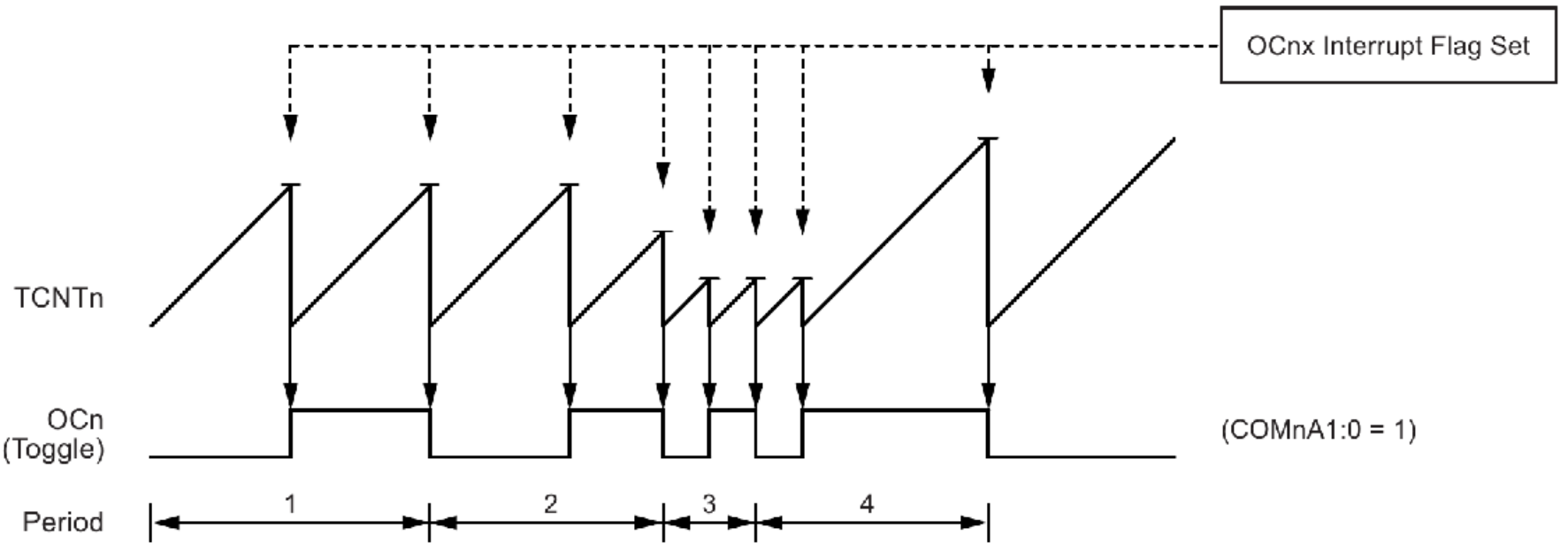
- **OCR0A** and **OCR0B** are used to control the duty cycles of the **OC0A** and **OC0B** pins.
- The PWM frequency is fixed (**TOP = 255**).
- When to use this mode: two PWM channels operating at the same fixed frequency.

2) Fast PWM: WGM02:0 = 0b111 (Mode 7), TOP = OCR0A

- **OCR0A** defines the **TOP** value and therefore controls the PWM frequency.
- **OCR0B** can be used as a normal PWM channel.
- **OC0A** cannot be used as a normal PWM output (except in toggle mode).
- When to use this mode: one PWM channel with an adjustable PWM frequency.

Timer in CTC Mode

CTC Mode, Timing Diagram



Fast PWM Mode vs. Phase-Correct PWM

1) Fast PWM

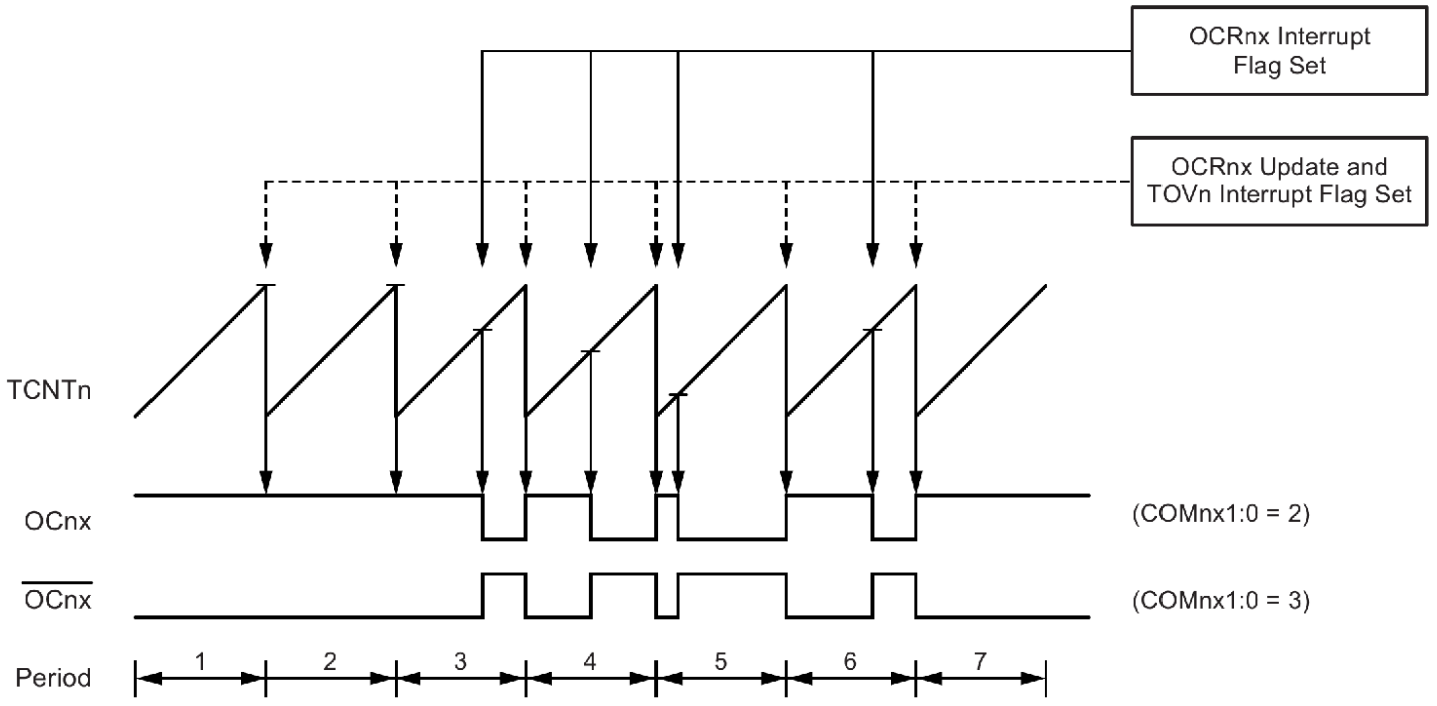
- The timer counts up only (single-slope).
- Pulse always starts at **BOTTOM** (0).

2) Phase-Correct PWM

- The timer counts up and down (dual-slope).
- Phase-Correct is half the frequency of Fast PWM.
- Center-Aligned: Pulse is centered in the period and symmetrical around midpoint.

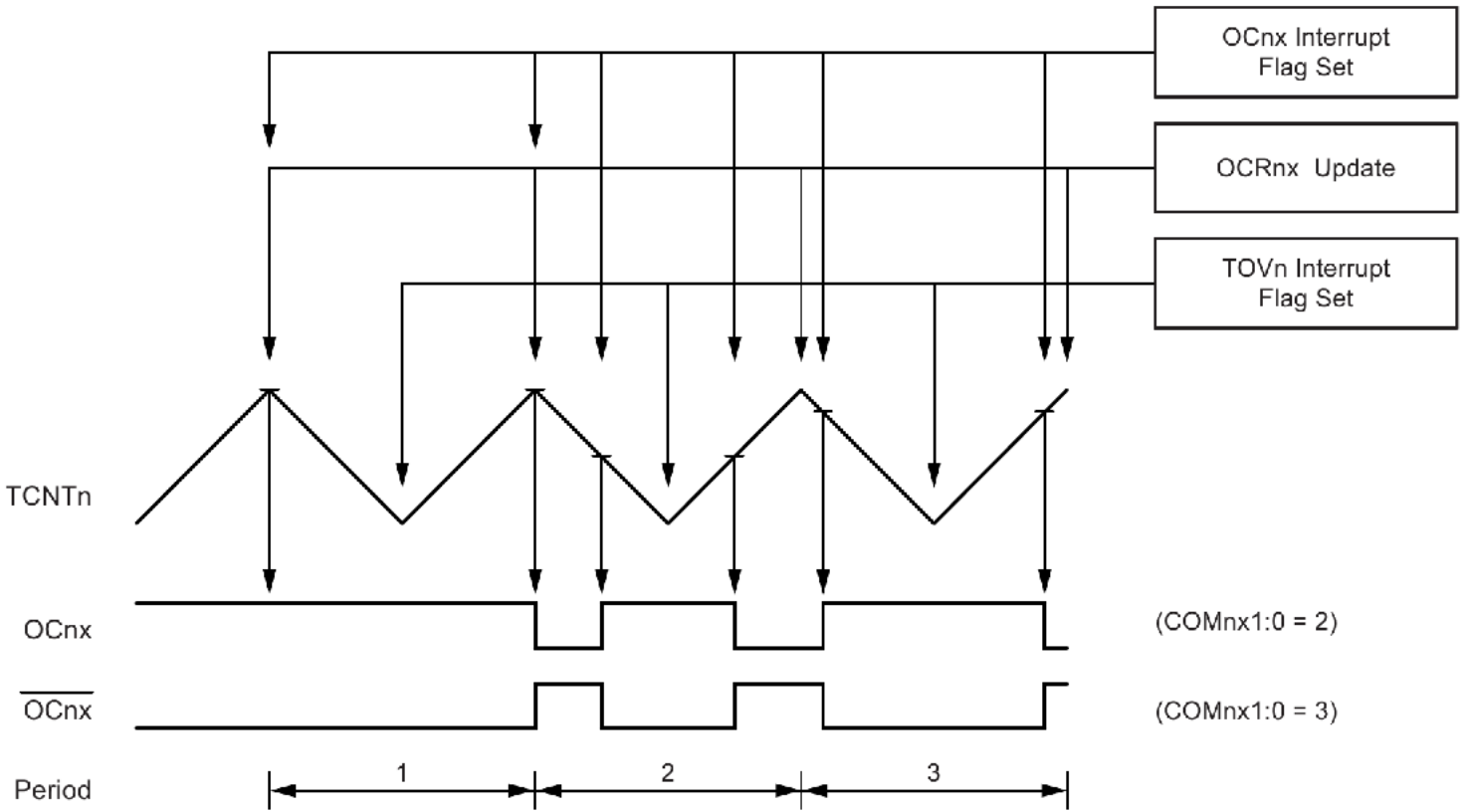
Timer in Fast-PWM Mode

Fast PWM Mode, Timing Diagram



Timer in Phase-Correct PWM Mode

Phase Correct PWM Mode, Timing Diagram



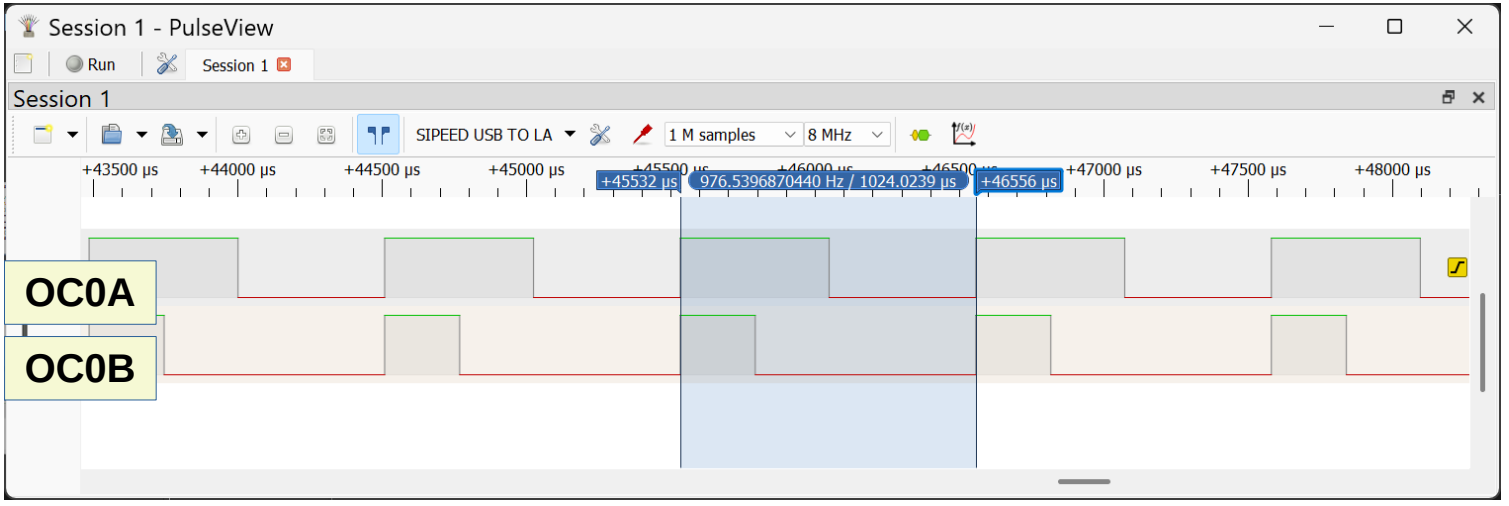
C Code: Timer0 Fast PWM Mode 3 (Two PWM Outputs)

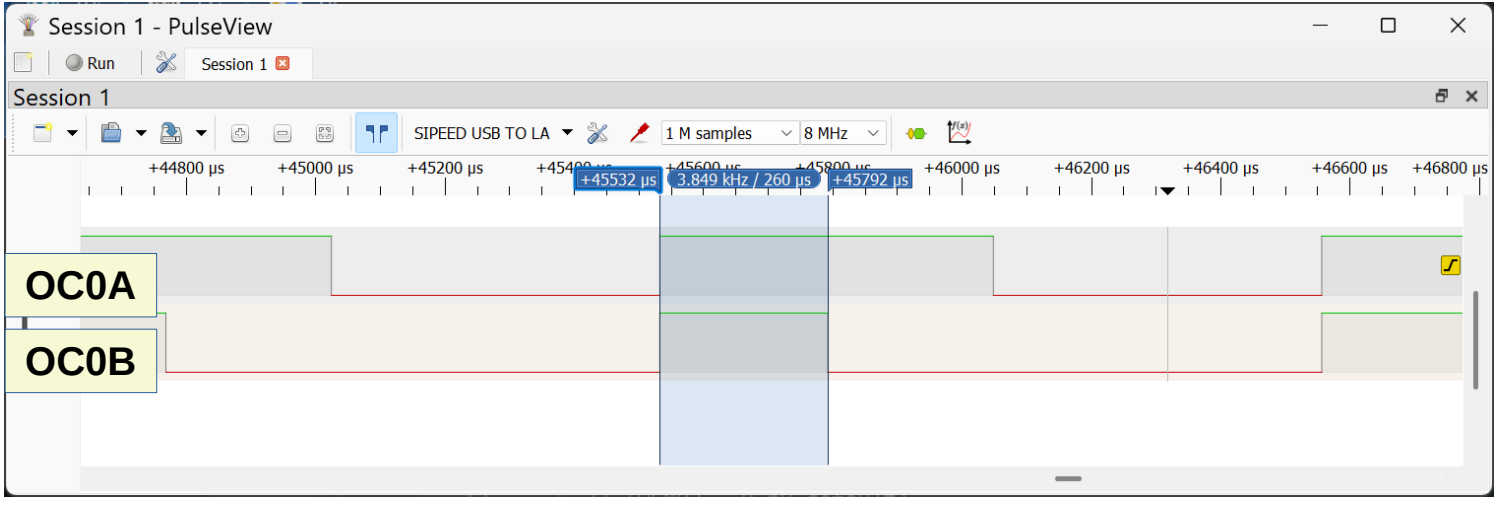
```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

void pwm_init(void) {
    // Set OC0A (PD6) and OC0B (PD5) as outputs
    DDRD |= (1<<DDD6) | (1<<DDD5);
    // Clear control registers
    TCCR0A = TCCR0B = 0;
    // Fast PWM Mode 3: WGM02:0 = 0b011
    // WGM01 = 1, WGM00 = 1
    TCCR0A |= (1<<WGM01) | (1<<WGM00);
    // (WGM02 = 0 by default)
    // Non-inverting mode on OC0A and OC0B
    // Clear on compare match, set at BOTTOM
    TCCR0A |= (1<<COM0A1) | (1<<COM0B1);
    // Set duty cycles (0–255)
    OCR0A = 128;   // 50% duty on OC0A
    OCR0B = 64;    // 25% duty on OC0B
    // Prescaler = 64 (CS02:0 = 0b011)
    TCCR0B |= (1<<CS01) | (1<<CS00);
}
```

```c
int main(void) {
    uint16_t count = 0;
    uint8_t dc;
    pwm_init();
    _delay_ms(1000);
    while (1) {
        count = (count + 4) % 512;
        dc = (count > 255) ? (511-count) : count;
        OCR0A = dc;
        OCR0B = 255-dc;
        _delay_ms(4);
    }
}
```

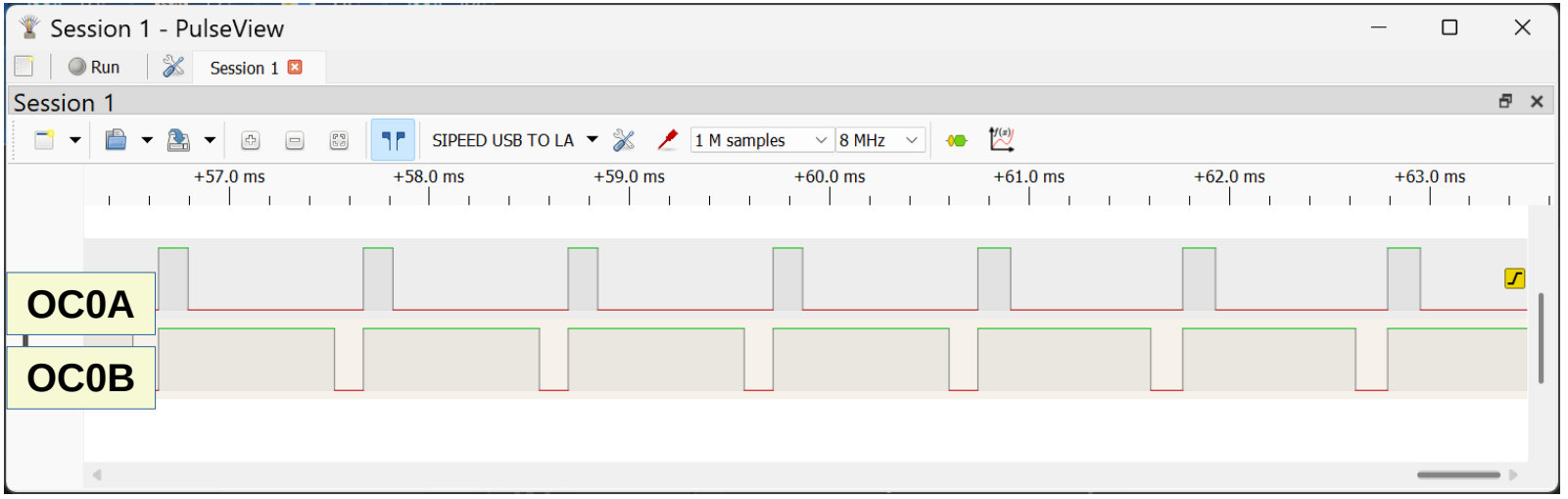
PWM Freq. = 16MHz / 64 / 256
= 976.56 Hz
T = 1024 usec

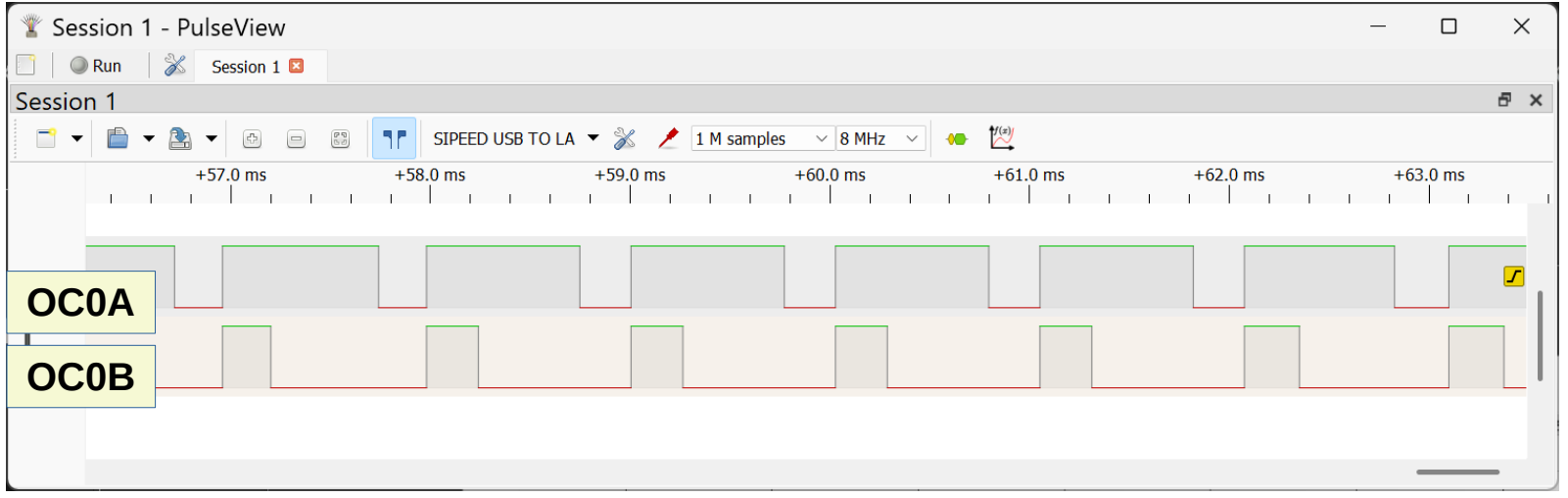
USB Logic Analyzer: Output Waveform





USB Logic Analyzer: Output Waveform





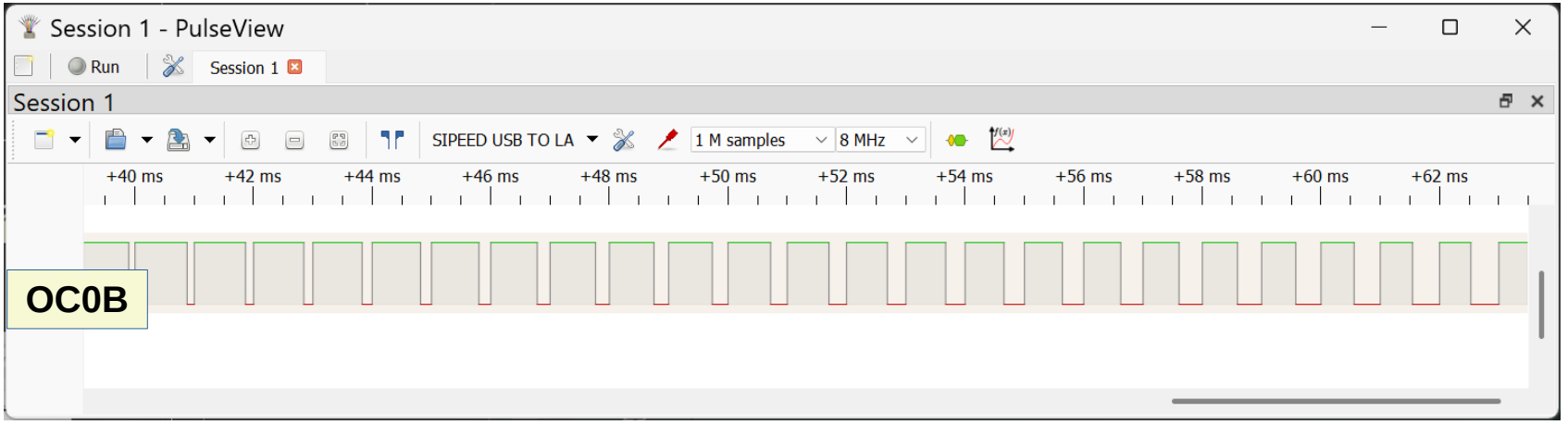
C Code: Timer0 Fast PWM Mode 7 (Single PWM Output)

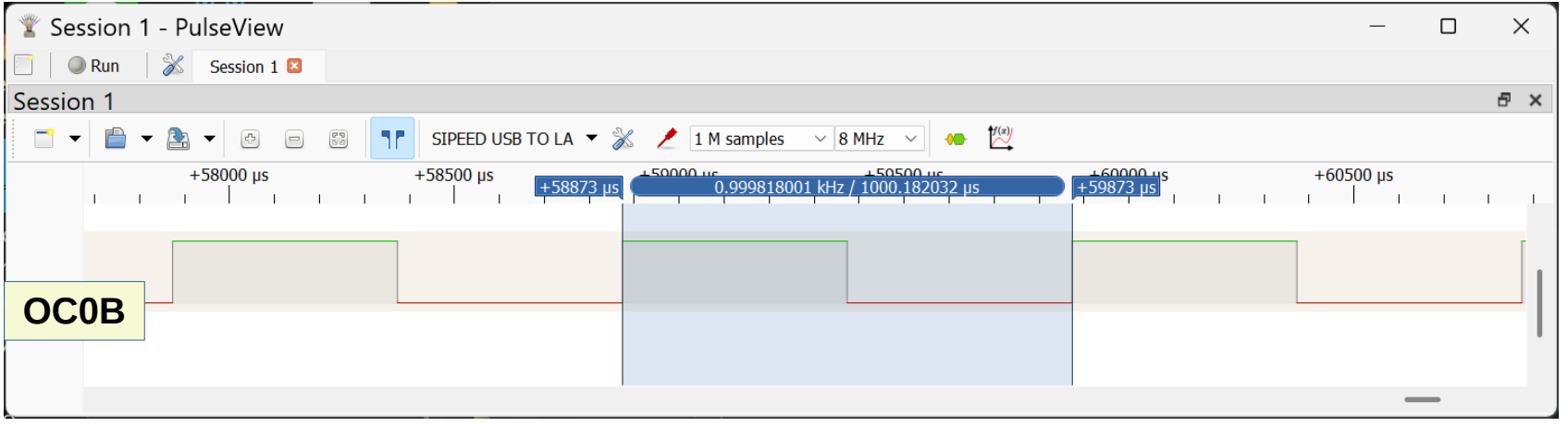
```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

#define PERIOD 250 // 8-bit unsigned

void pwm_init(void) {
  // Set OC0B (PD5) as output
  DDRD |= (1<<DDD5);
  // Clear control registers
  TCCR0A = TCCR0B = 0;
  // Fast PWM Mode 7 (WGM02:0 = 0b111)
  TCCR0A |= (1<<WGM01) | (1<<WGM00);
  TCCR0B |= (1<<WGM02);
  // Non-inverting mode on OC0B
  TCCR0A |= (1<<COM0B1);
  // Set TOP for ~1kHz
  OCR0A = PERIOD - 1; // Define PWM frequency
  // Initially set the duty cycle
  OCR0B = PERIOD / 2;  // ~50%
  // Prescaler = 64
  TCCR0B |= (1<<CS01) | (1<<CS00);
}
```

```c
int main(void) {
  uint16_t MAX = 2 * PERIOD;
  uint16_t count = 0;
  uint8_t dc;
  pwm_init();
  _delay_ms(1000);
  while (1) {
    count = (count + 4) % MAX;
    dc = (count >= PERIOD)
          ? (MAX - 1 - count) : count;
    // Must stay within 0..OCR0A
    OCR0B = dc;
    _delay_ms(1);
  }
}
```

PWM Freq. = 16MHz / 64 / 250
= 1000 Hz
T = 1000 usec

USB Logic Analyzer: Output Waveform





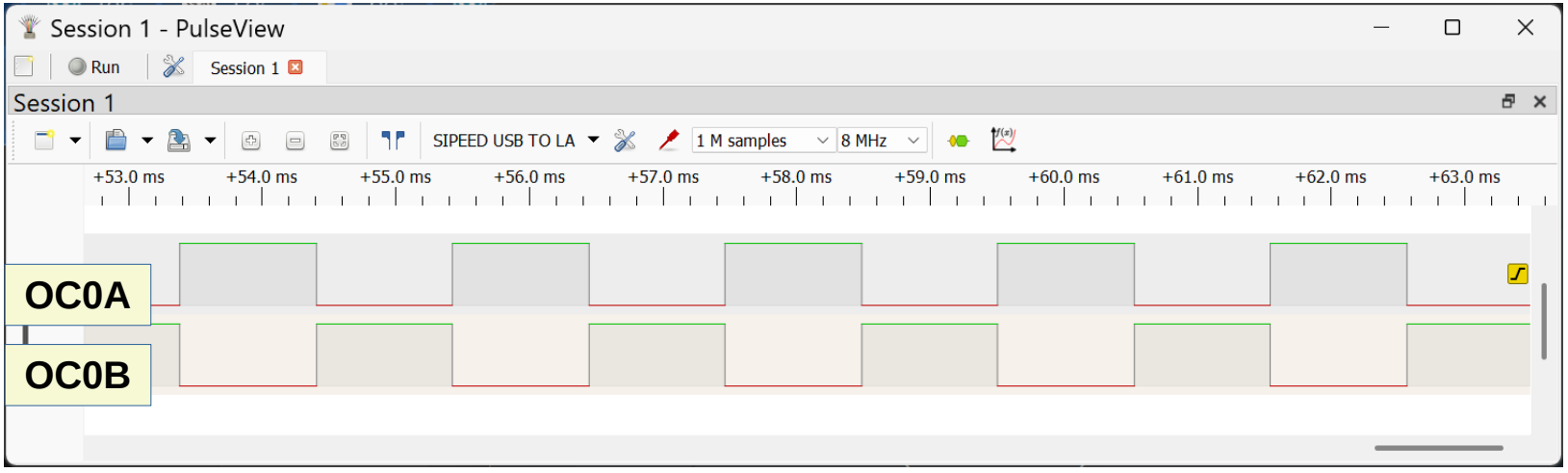
C Code: Timer0 Phase-Correct PWM (Mode 1)

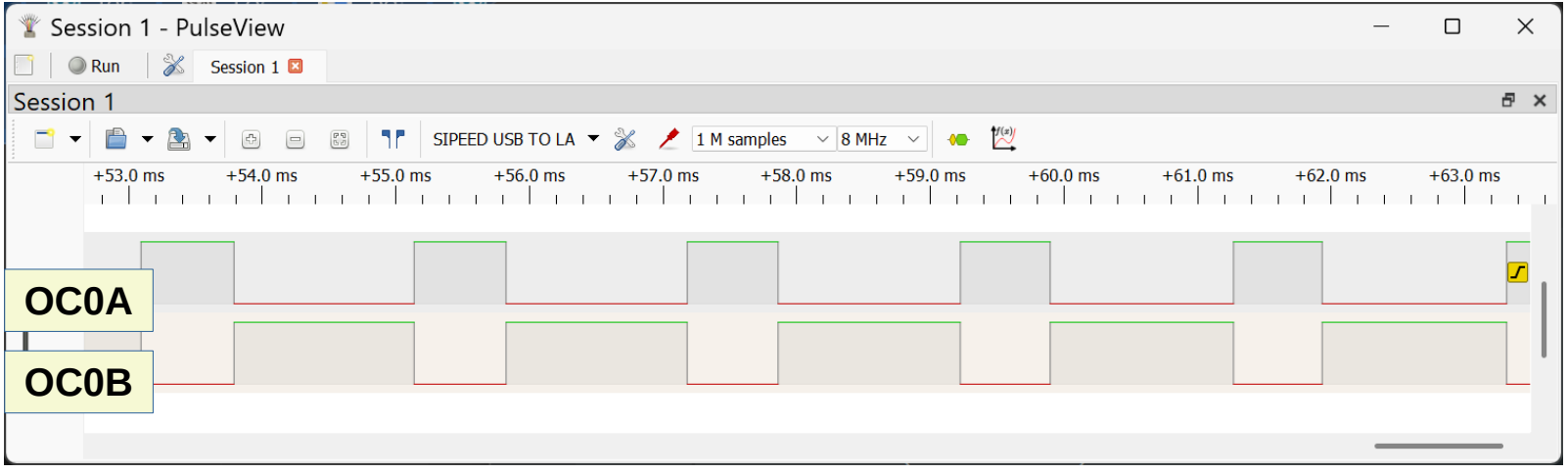
```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

void pwm_init(void) {
  // OC0A (PD6) and OC0B (PD5) as outputs
  DDRD |= (1<<DDD6) | (1<<DDD5);
  TCCR0A = TCCR0B = 0;
  // Phase-Correct PWM Mode 1 (WGM02:0 = 0b001)
  TCCR0A |= (1<<WGM00);
  // OC0A non-inverting
  TCCR0A |= (1<<COM0A1);
  // OC0B inverting (complementary)
  TCCR0A |= (1<<COM0B1) | (1<<COM0B0);
  // Initially set duty cycle (0..255)
  OCR0A = 128;
  OCR0B = 128;
  // Prescaler = 64
  TCCR0B |= (1<<CS01) | (1<<CS00);
}
```

```c
#define DC_MIN (10)
#define DC_MAX (255 - DC_MIN)

int main(void) {
  uint8_t duty = 40;
  int8_t step = 1;
  pwm_init();
  while (1) {
    if (duty < DC_MIN) step = 1;
    if (duty > DC_MAX) step = -1;
    duty += step;
    OCR0A = duty;
    OCR0B = duty;
    _delay_ms(2);
  }
}
```

PWM Freq. = 16MHz / 64 / 510
= 490.19 Hz
T = 2040 usec

USB Logic Analyzer: Output Waveform





C Code: Timer2 Phase-Correct PWM (Mode 1)

```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

void pwm_init(void) {
  // Set OC2A (PB3) and OC2B (PD3) as outputs
  DDRB |= (1<<DDB3);  // OC2A pin
  DDRD |= (1<<DDD3);  // OC2B pin
  TCCR2A = TCCR2B = 0;
  // Phase-Correct PWM Mode 1 (WGM22:0 = 0b001)
  TCCR2A |= (1<<WGM20);
  // OC2A non-inverting
  TCCR2A |= (1<<COM2A1);
  // OC2B inverting (complementary output)
  TCCR2A |= (1<<COM2B1) | (1<<COM2B0);
  // Initially set duty (0..255)
  OCR2A = 128;
  OCR2B = 128;
  // Prescaler = 64
  TCCR2B |= (1<<CS22);
}
```

```c
#define DC_MIN (10)
#define DC_MAX (255 - DC_MIN)

int main(void) {
  uint8_t duty = 40;
  int8_t step = 1;
  pwm_init();
  while (1) {
    if (duty < DC_MIN) step = 1;
    if (duty > DC_MAX) step = -1;
    duty += step;
    OCR2A = duty;
    OCR2B = duty;
    _delay_ms(2);
  }
}
```