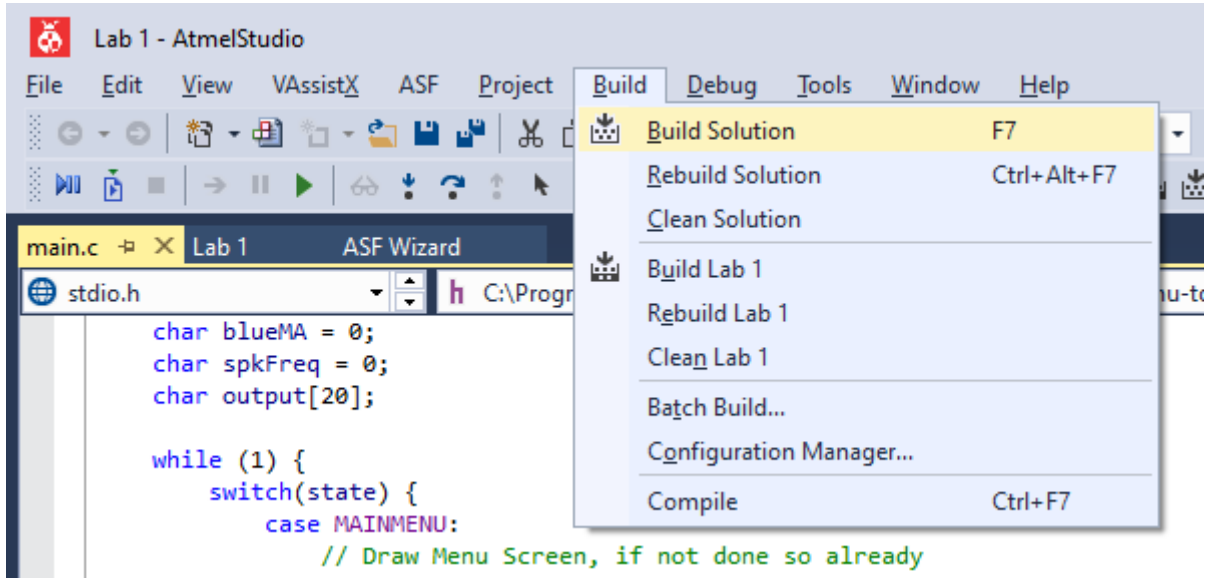
# Checking your Code

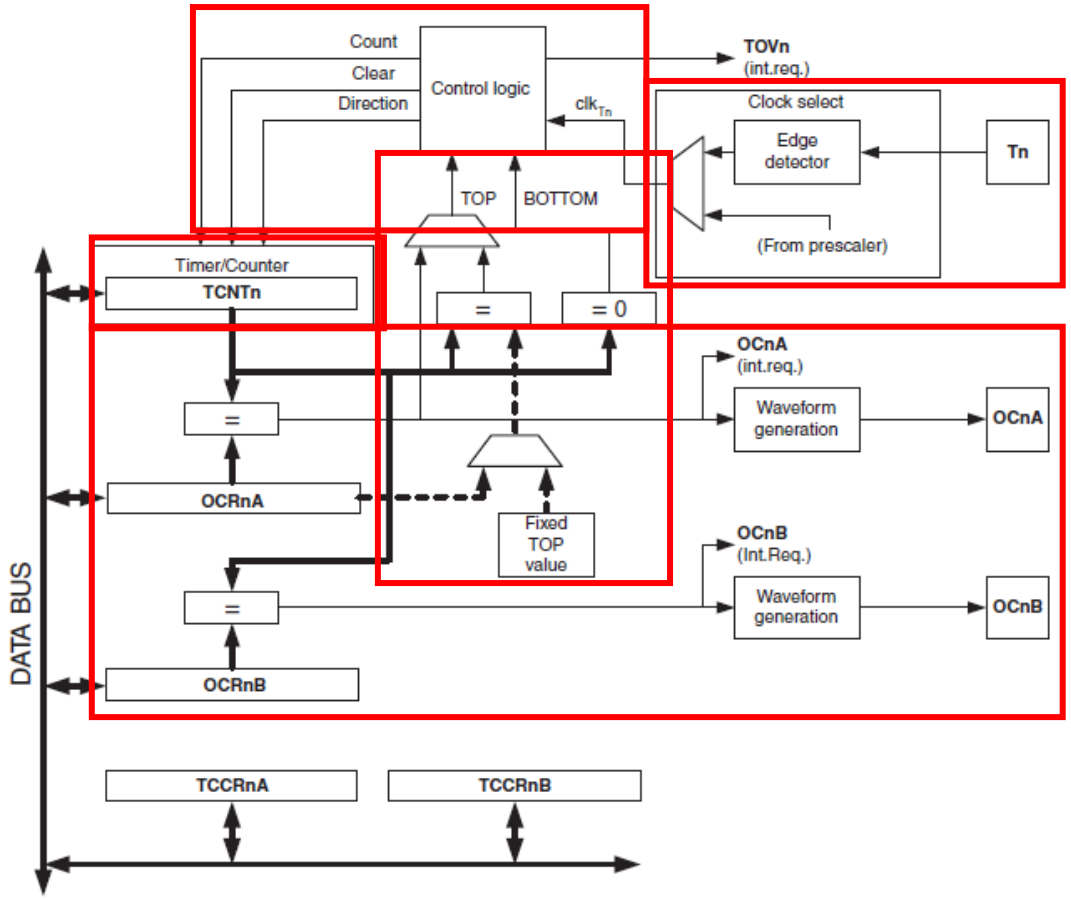



Use the Build function (F7) in ATMEL studio to check for simple errors:

- Syntax Errors
- Undeclared Variables / Variables not in scope
- Incorrect use of functions

# Timers

- Clock Select
  - Based off $f_{osc}$ (and Prescaler) <u>or</u> from external source (via Tn Pin)
- Clock Logic
  - This instructs the timer/counter what to do next
  - Considers parameters from other inputs (clock pulse, TOP & BOTTOM values, setup registers)
- Timer/Counter
  - Register TCNTn
  - Will either count up, down, or reset to TOP/BOTTOM value
- Output Compare Registers
  - OCRnA/OCRnB
  - Control the outputs on OCnA/OCnB pins on the µC
  - OCRnA can also influence the Control Logic

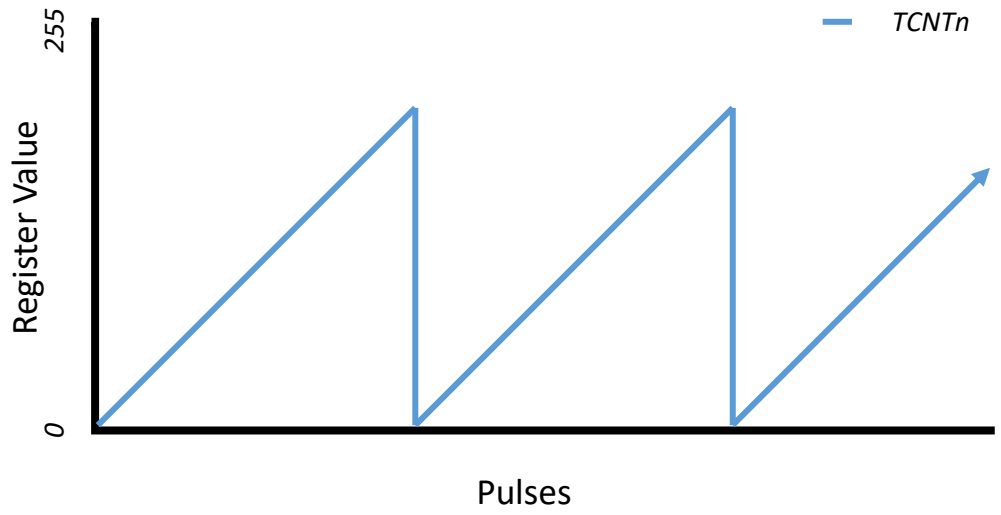


# Timer Modes

- There are four different modes
  - Normal
  - Clear Timer on Compare Match
  - Fast PWM
  - Phase-Correct PWM

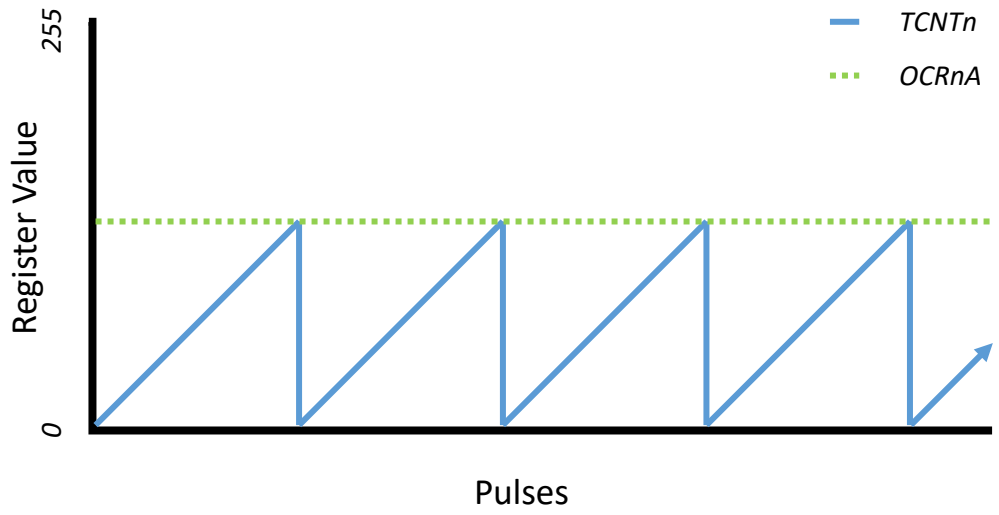
# Modes - Normal

- Timer counts up from 0 to 0xFF (25)
- Events can be set on the timer to occur when the values in Output Compare Registers match
- Four Compare Match Modes
  - Normal
  - Toggle
  - Clear
  - Set

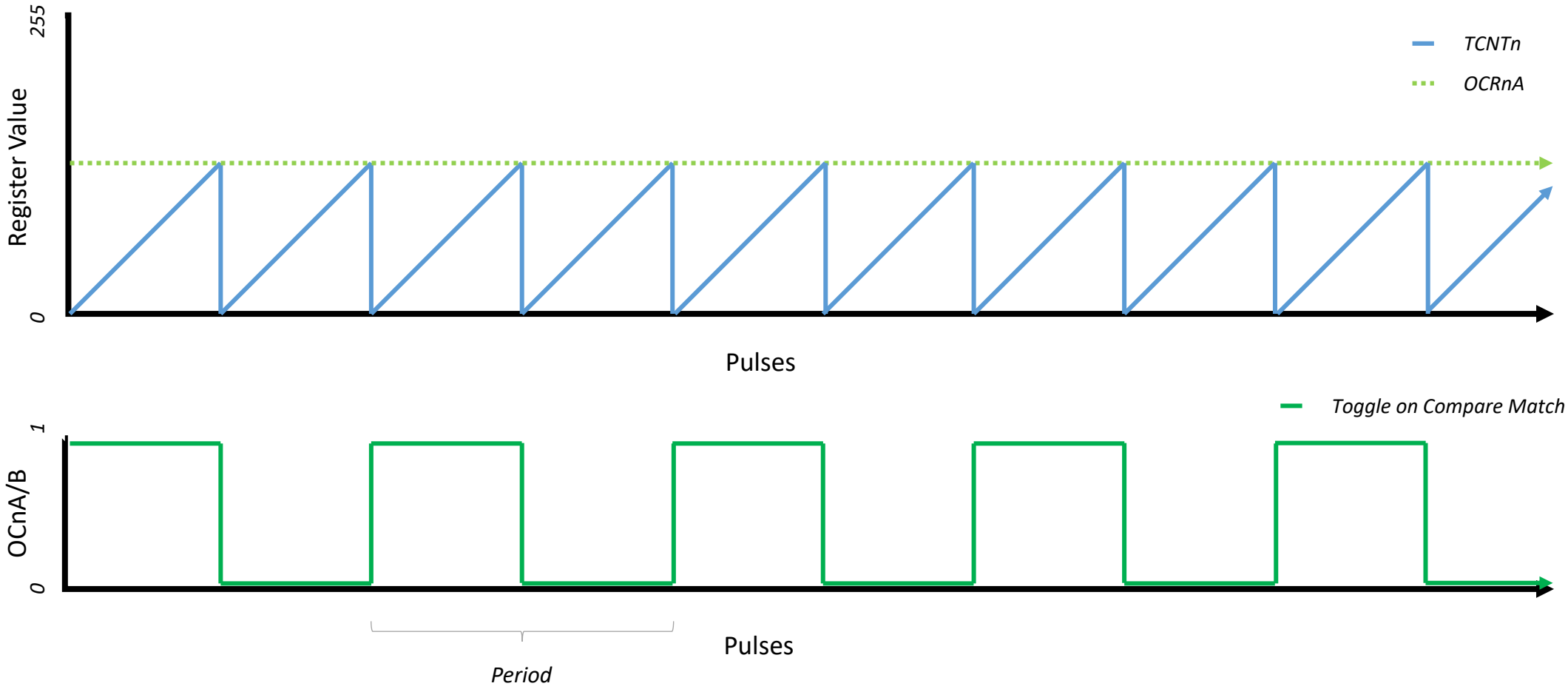


# Modes - Clear on Compare Match (CTC)

- Timer counts up from 0 to the TOP value stored in OCRnA
- When reached, the timer clears to 0, and an Interrupt is triggered (if enabled)
- Four Compare Match Modes
  - Normal
  - Toggle
  - Clear
  - Set

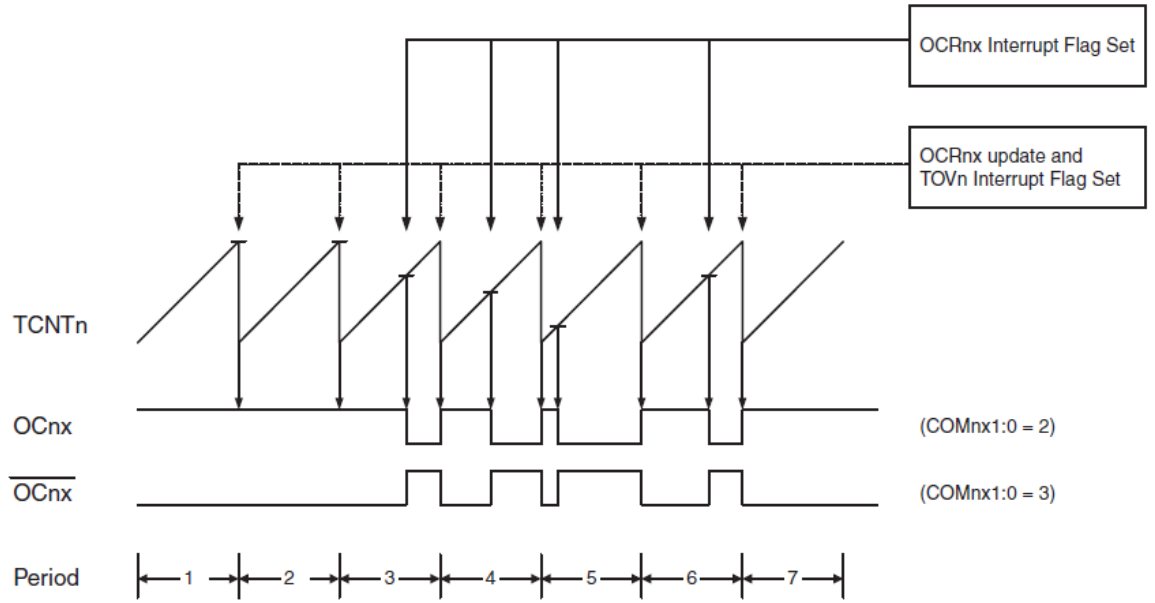


# CTC Output Modes



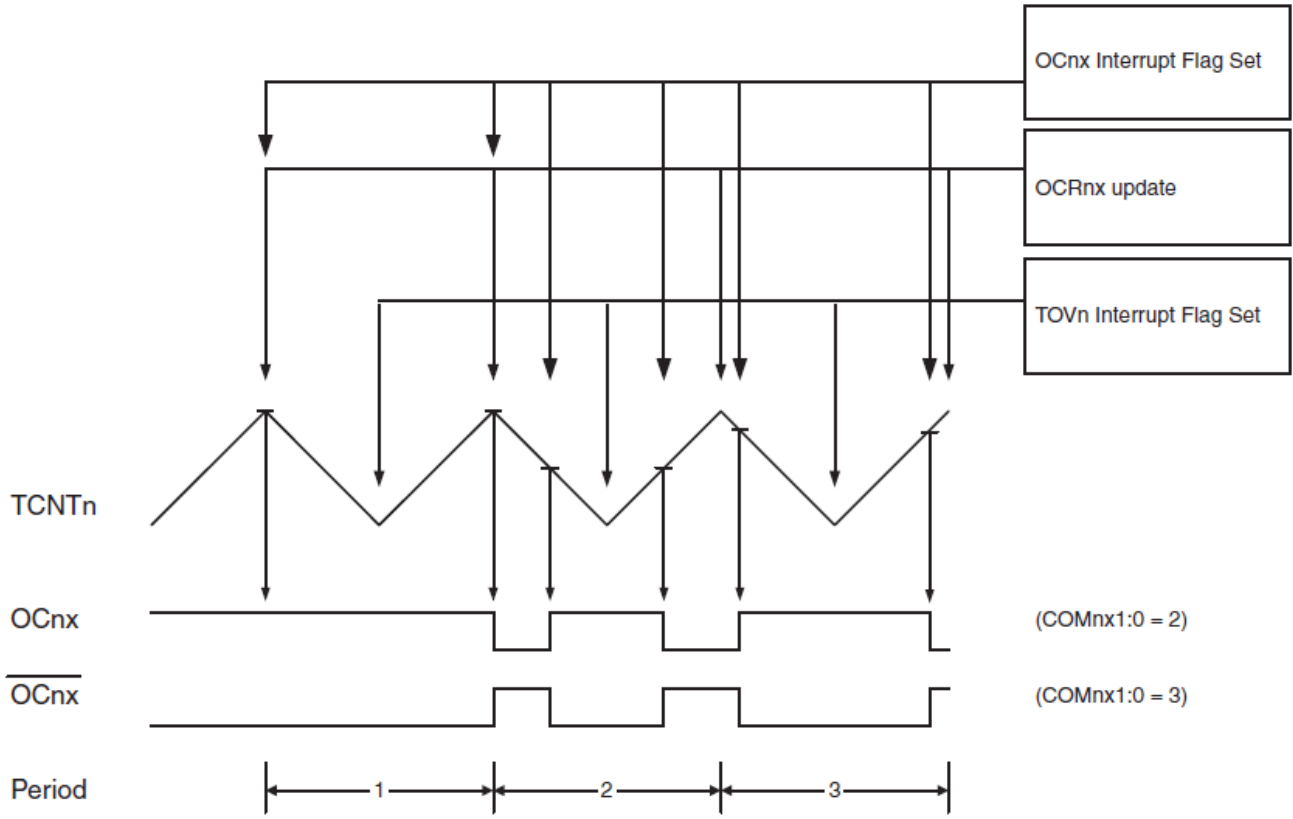
# Modes – Fast PWM

- Timer counts from 0 to TOP value (0xFF or OCRnA Value)
- Outputs on OC2A and OC2B can be set for a non-inverted or inverted output signal.



# Modes – Phase-correct PWM

- Operates similarly to Fast PWM
  - Notice how the timer counts UP then back DOWN (two slopes)
  - This reduces the effective frequency of PWM (hence not “Fast”)



# Calculating Top Values for Frequencies

$$f_{PWM} = \frac{f_{osc}}{2 \times Prescaler \times (Top + 1)} \quad \leftrightarrow \quad Top = \frac{f_{osc}}{2 \times Prescaler \times f_{PWM}} - 1$$

# Interrupt Vectors Table

Use this table to identify the vectors you need to use:

- Find the subsystem you need (ex. USART)
- Find the trigger you want to use (ex. Rx Complete)

Ensure you enable the appropriate interrupt registers for the device (ex. RXCIE Flag in UCSR1B)

| Vector No | Program Address | Source | Interrupt Definition |
|---|---|---|---|
| 1 | $0000 | RESET_vect | Reset |
| 2 | $0002 | INT0_vect | External Interrupt Request 0 |
| 3 | $0004 | INT1_vect | External Interrupt Request 1 |
| 4 | $0006 | INT2_vect | External Interrupt Request 2 |
| 5 | $0008 | INT3_vect | External Interrupt Request 3 |
| 6 | $000A | INT4_vect | External Interrupt Request 4 |
| 7 | $000C | INT5_vect | External Interrupt Request 5 |
| 8 | $000E | INT6_vect | External Interrupt Request 6 |
| 9 | $0010 | INT7_vect | External Interrupt Request 7 |
| 10 | $0012 | PCINT0_vect | Pin Change Interrupt Request 0 |
| 11 | $0014 | USB_General_vect | USB General Interrupt request |
| 12 | $0016 | USB_Pipe__vect | USB Endpoint/Pipe Interrupt request |
| 13 | $0018 | WDT_vect | Watchdog Time-out Interrupt |
| 14 | $001A | TIMER2_COMPA_vect | Timer/Counter2 Compare Match A |
| 15 | $001C | TIMER2_COMPB_vect | Timer/Counter2 Compare Match B |
| 16 | $001E | TIMER2_OVF_vect | Timer/Counter2 Overflow |
| 17 | $0020 | TIMER1_CAPT_vect | Timer/Counter1 Capture Event |
| 18 | $0022 | TIMER1_COMPA_vect | Timer/Counter1 Compare Match A |
| 19 | $0024 | TIMER1_COMPB_vect | Timer/Counter1 Compare Match B |
| 20 | $0026 | TIMER1_COMPC__vect | Timer/Counter1 Compare Match C |
| 21 | $0028 | TIMER1_OVF_vect | Timer/Counter1 Overflow |
| 22 | $002A | TIMER0_COMPA_vect | Timer/Counter0 Compare Match A |
| 23 | $002C | TIMER0_COMPB_vect | Timer/Counter0 Compare match B |
| 24 | $002E | TIMER0_OVF_vect | Timer/Counter0 Overflow |
| 25 | $0030 | | |
| 26 | $0032 | USART1_RX_vect | USART1 Rx Complete |
| 27 | $0034 | | |
| 28 | $0036 | USART1_TX_vect | USART1 Tx Complete |
| 2 9 | $0038 | | |
| 30 | $003A | ADC_vect | ADC Conversion Complete |
| 31 | $003C | EE_READY_vect | EEPROM Ready |
| 32 | $003E | TIMER3_CAPT_vect | Timer/Counter3 Capture Event |
| 33 | $0040 | TIMER3_COMPA_vect | Timer/Counter3 Compare Match A |
| 34 | $0042 | TIMER3_COMPB_vect | Timer/Counter3 Compare Match B |
| 35 | $0044 | TIMER3_COMPC_vect | Timer/Counter3 Compare Match C |
| 36 | $0046 | TIMER3_OVF_vect | Timer/Counter3 Overflow |
| 37 | $0048 | TWI_vect | 2-wire Serial Interface |
| 38 | $004A | SPM_ READY_vect | Store Program Memory Ready |

# Using Interrupts in your Program

- Include the <avr/interrupt.h> library
- Create Interrupt Service Routines (ISRs) using the function ISR()
  - Make sure you also declare the function definition
  - Ensure that you pass in the name of the interrupt vector used
  - Write the code inside the function that you want to use
- Enable Interrupts using the sei() function, and setting the appropriate registers.

```
// Include the Interrupt Library
#include <avr/interrupt.h>

// Declare the Function Definition
   for the ISR
ISR(Your Vector's Name);

// Write the code to run within
   your interrupt
ISR(Your Vector's Name)
{
    // Your Code Here
}

// Enable Interrupts in your
   setup() function
sei();
```

# Example: Receiving Data over USART

This code takes the 8-bit value received over the USART and displays on PORTC:

*The example is the basic outline to show you how vectors can be used.*

- Use the USART1_RX_vect
- Enable Interrupts in setup()
  - Using the sei() function
- Create Interrupt Service Routines (ISRs) using the function ISR()
  - Stores the value from the USART
  - Writes to PORTC
- Ensure interrupts are enabled on the USART registers
  - UCSR1B: RXCIE

```
ISR(USART1_RX_vect);
void setup(void);

int main (void)
{
    setup();
}

// Every Time USART recieves data, show on PORTC
ISR(USART1_RX_vect)
{
    char valueToDisplay = UDR1;
    PORTC = valueToDisplay;
}

void setup(void)
{
    DDRC = 0xFF;
    UCSR1B = 0b10010000;
    UCSR1C = 0b00000110;
    UBRR1L = 8;

    // Enable Interrupts in your setup() function
    sei();
}
```