Introduction to Microcontrollers

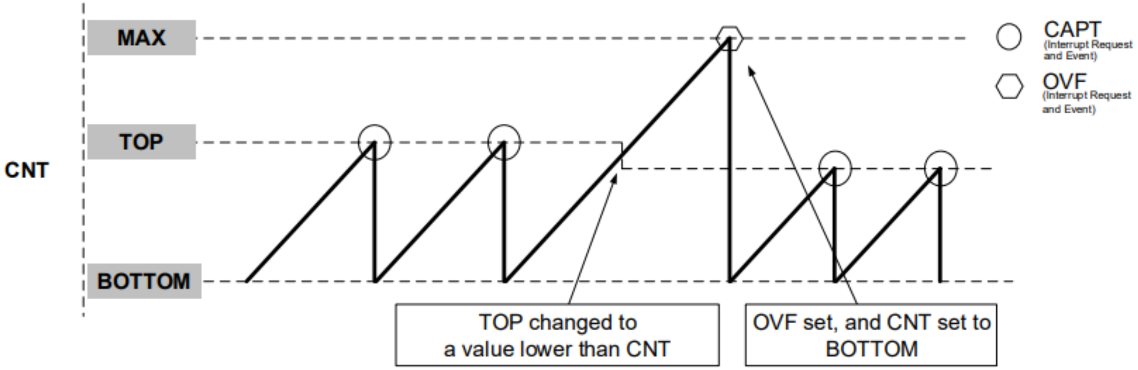
Timers and Interrupts

Goals

- Know what a timer is, and what they are used for
- Know what interrupts are, and what they are used for
- Learn how to use interrupts with AVRs
- Learn how to make PWM signals with AVRs

Timers

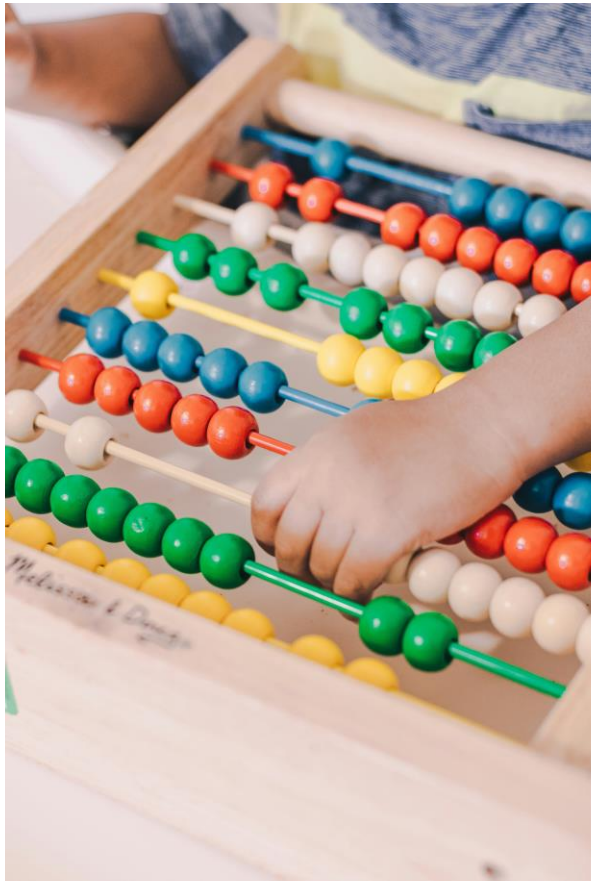
- Also referred to as counters
- Main objective of a timer is to count
  - Count up
  - Count down
  - Count up then down
  - You name it
- So what is so special about counting?
  - Any thoughts?



The Power of Counting

- Periodic interrupts
  - Maybe a piece of code needs to happen periodically
  - Maybe a delay function that is non-blocking via ticks?
- Time-outs
  - Stop a task if it took to long?
- Waveform measurements
  - Frequency
  - Pulse-width
  - Number of events
- Waveform generation
  - Pulse-width
  - Frequency
- Noise cancellation
  - Maybe debounce a button with a single shot counter?
- This is just the possibility of ONE counter
  - By combining multiple we can create very complex circuits in hardware!

Luis Arias

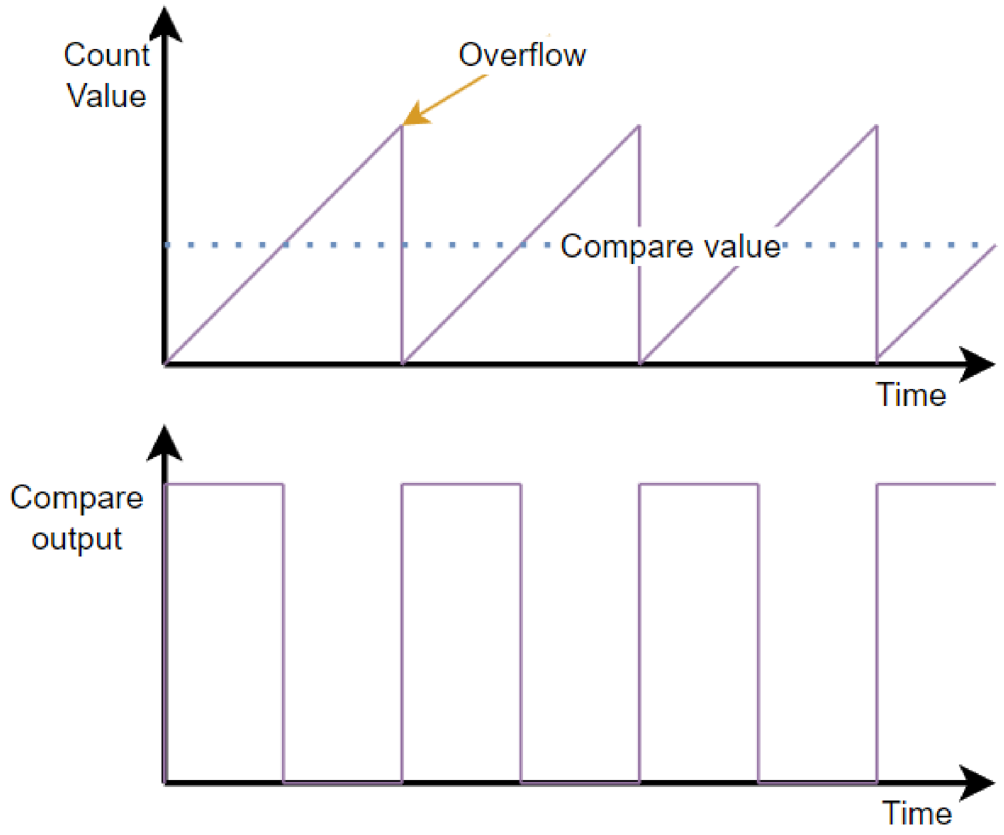


Definitions

- CNT
  - Counter value – Increments every clk_per
- TOP
  - Counter maximum value – Overflow when CNT = TOP
    - Alternatively BTM for underflow, but TOP is more used
  - Some AVR peripherals call this PER for period
- CMP
  - Compare – Toggle/Clear register CNT = CMP
    - Dependent on timer/counter. If Clear is used, register is set on overflow
  - Often used for waveform generation

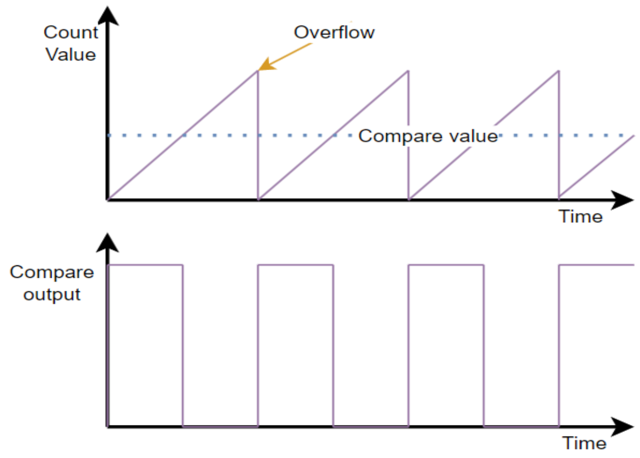
Timer as Waveform Generator

- Typically we count UP
  - Reset CNT value on TOP
    - Overflow
- Typically we have one or more CMP registers
  - CMP0, CMP1, …, CMPn
- Example: PWM
  - Pulse-Width Modulation
  - GPIO is ‘1’ when CNT < CMP
  - GPIO is ‘0’ when CNT > CMP



Timers as Waveform Generator

- PWM
  - Pulse-Width = ON time
  - Duty-Cycle = ON time / OFF time
- Frequency
  - 1/Period
  - Period = ON time + OFF time
- Timer/counter resolution
  - TOP register value determines
    - Frequency
    - Resolution for PWM
  - Not all duty cycles and frequencies can be synthesized
    - Dithering is an option



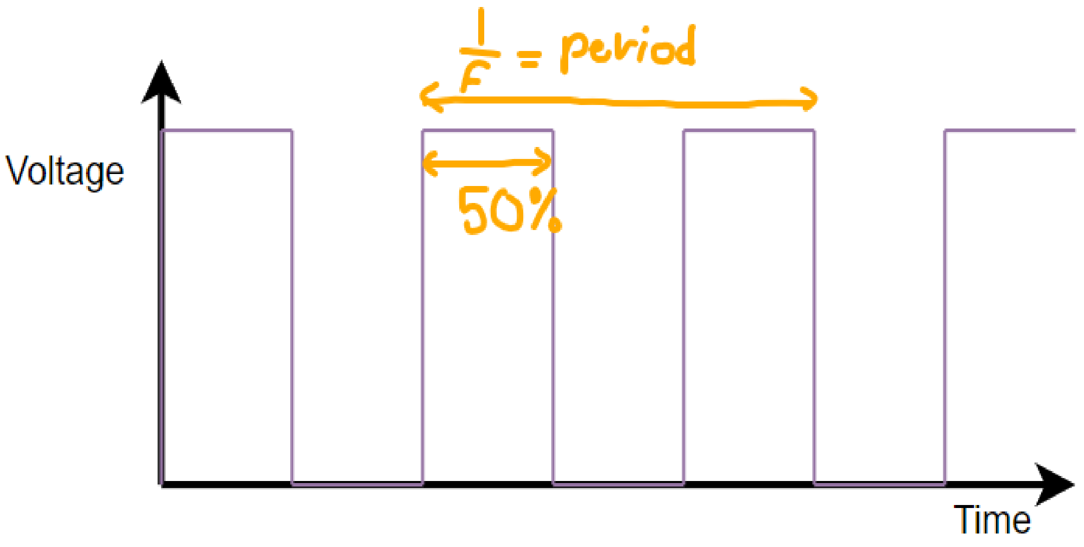
$$f = \frac{f_{counter}}{\mathbf{TOP}}$$

$$D = \frac{\mathbf{CMP}}{\mathbf{TOP}}$$

$$f_{counter} = \frac{f_{CPU}}{\mathbf{PRESCALER}}$$

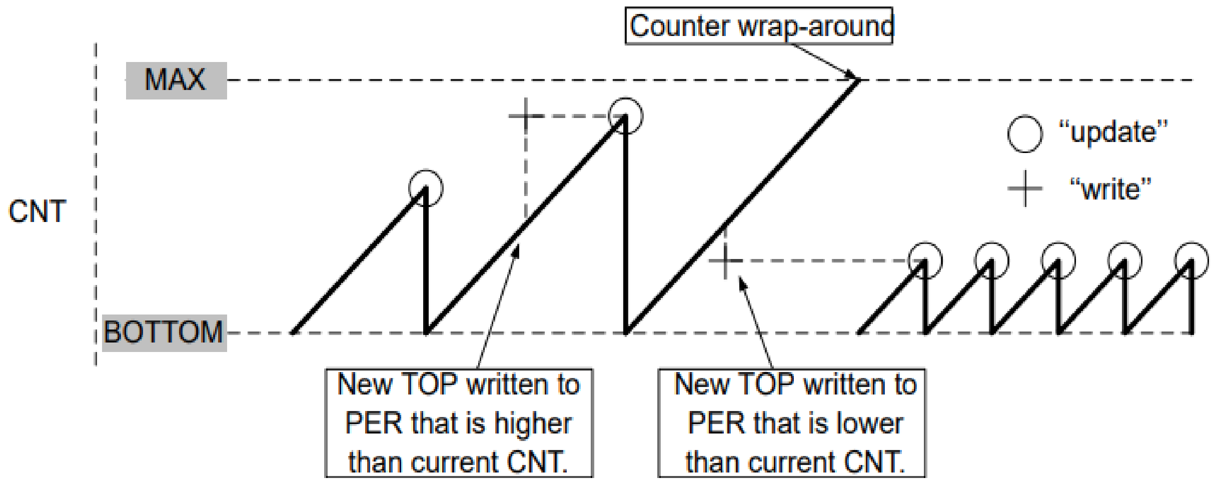
PWM

- Control external circuits
- Typical uses
  - LED dimming
  - DACs
  - Motor control
  - Voltage regulation
- Can be combined with software to create complex signals
  - Example: Sinusoidal PWM



Gotchas

- Some timer counters have double buffering
  - Registers updated on overflow
  - Makes sure timer always works as intended
- Not all timer counters can do everything
  - They are usually designed for some specific tasks
  - This is why we have so many variants



Timers in AVRs

- Many different peripherals

Name	Common use
• TCA (Most newer AVRs)	- Waveform generation
• TCB (All newer AVRs)	- Waveform measurement / Periodic interrupt
• TCD (Some newer AVRs)	- Motor control
• TCE (Few newer AVRs)	- Waveform generation
• TCF (Few newer AVRs)	- Frequency generation
• RTC (All newer AVRs)	- Periodic interrupt
• ++	

TCB

- Simplest timer counter
- AVR128DA48
  - 4 instances!
  - TCB0, TCB1, TCB2, TCB3
- We will use it for periodic interrupts today

Peripheral Overview

The following table shows the peripheral overview of the entire AVR® DA(S) family. Further documentation describes only the AVR128DA28/32/48/64(S) devices.

Table 2. Peripheral Overview

Feature	AVR128DA28(S) AVR64DA28(S) AVR32DA28(S)	AVR128DA32(S) AVR64DA32(S) AVR32DA32(S)	AVR128DA48(S) AVR64DA48(S) AVR32DA48(S)	AVR128DA64(S) AVR64DA64(S)
Pins	28	32	48	64
Max. Frequency (MHz)	24	24	24	24
16-bit Timer/Counter type A (TCA)	1	1	2	2
16-bit Timer/Counter type B (TCB)	3	3	4	5
12-bit Timer/Counter type D (TCD)	1	1	1	1
Real-Time Counter (RTC)	1	1	1	1
USART	3	3	5	6
SPI	2	2	2	2
TWI/I^{2}C	1[(1)]	2[(1)]	2[(1)]	2[(1)]
12-bit Differential ADC (channels)	1 (10)	1 (14)	1 (18)	1 (22)
10-bit DA(S)C (outputs)	1(1)	1(1)	1(1)	1(1)
Analog Comparator (AC)	3	3	3	3
Zero-Cross Detectors (ZCD)	1	1	2	3

22. TCB - 16-Bit Timer/Counter Type B

22.1 Features

- 16-bit Counter Operation Modes:
  - Periodic interrupt
  - Time-out check
  - Input capture
    - On event
    - Frequency measurement
    - Pulse-width measurement
    - Frequency and pulse-width measurement
    - 32-bit capture
  - Single-shot
  - 8-bit Pulse-Width Modulation (PWM)
- Noise Canceler on Event Input
- Synchronize Operation with TCAn

Interrupts

- Hardware can stop CPU from executing instructions and jump to a new place in memory and execute from there
  - New piece of code is called an Interrupt Service Routine (ISR)
  - Once ISR is complete, CPU jumps back to where it was previously and continues where it left off
- Perks
  - We can execute code on specific conditions
  - Especially useful for time sensitive operations

Interrupts

- Typical use cases
  - Communication
    - Example: Received data or ready for new data
  - Periodic interrupts
    - Example: Periodic measurements or actions
  - Safety critical hardware
    - Example: Brownout or Watchdog timeout
  - Measurements
    - Example: ADC conversion complete or ADC result above threshold
  - IO
    - Example: Change in IO logic
- AVR Event system
  - Some interrupts can be replaced by hardware in AVRs
  - Example: Start ADC on timer interrupt

Interrupts

- Pro
  - Perfect for time sensitive operations
  - Perfect for system timers
  - Perfect for rare occurrences
    - Do not need to check in main loop
  - Can prevent blocking code
- Con
  - Easy to abuse
  - Jumping from main.c at the wrong time can cause problems
    - Non-atomic instructions
    - Maybe a signal is left on too long?
  - ISR nesting
    - Timing issues
    - Race conditions
    - Deadlock
    - Stack depth

Interrupts

- Rules of thumb
  - Only use interrupts if it makes sense
    - Avoid using them to make things easy
  - Make ISRs short
    - Don’t have many instructions in an ISR
    - Typical case: Read some register value, set a flag, clear interrupt, then back to main
  - Don’t use interrupts for events that occur too frequently
    - SysTick is an exception
    - May become locked inside ISR
      - Round Robin
- If some interrupts are more time sensitive than others
  - Make them a higher priority!
  - Higher priority ISRs can interrupt other ISRs
- If some parts of your code should not be interrupted disable interrupts before doing these instructions and enable them after
- AVR
  - Global interrupt enable: sei()
  - Global interrupt disable: cli()
  - avr/interrupt.h must be included

Interrupts in practice

Small ISR = OK

```
    while (1)
    {
        // Do nothing
    }
}

ISR(TCB0_INT_vect)
{
    // Clear interrupt flag
    TCB0.INTFLAGS = TCB_CAPT_bm;

    // Toggle LED
    PORTC.OUTTGL = PIN6_bm;
}
```

Software flag = GOOD

```
volatile uint8_t tcb0_flag = 0;

int main(void)
{

    while (1)
    {
        if (tcb0_flag)
        {
            tcb0_flag = 0;

            // Do something
        }
    }
}

ISR(TCB0_INT_vect)
{
    // Clear interrupt flag
    TCB0.INTFLAGS = TCB_CAPT_bm;

    // Set software flag
    tcb0_flag = 1;
}
```

Long ISR and use of functions = BAD

```
ISR(TCB0_INT_vect)
{
    // Clear interrupt flag
    TCB0.INTFLAGS = TCB_CAPT_bm;

    check_adc();
    compute_pid_values();
    check_buttons();
}
```

Task: TCB Periodic Interrupt

- LED toggle task
  - No _delay_ms()
  - Instead we use software
- Same as Arduino function millis()
- Non blocking!
- Many tasks can use the same system timer

- Of course we need to configure TCB0 to have an interrupt every 1 ms

```c
volatile uint32_t millis = 0;

int main(void)
{
    uint32_t led_millis_last = 0, led_millis_delay = 500;
    while (1)
    {
        // Check if 500 ms has passed
        if ((millis - led_millis_last) >= led_millis_delay)
        {
            // Update last millisecond value
            led_millis_last = millis;

            PORTC.OUTTGL = PIN6_bm;
        }
    }
}

ISR(TCB0_INT_vect)
{
    // Clear interrupt flag
    TCB0.INTFLAGS = TCB_CAPT_bm;

    millis++;
}
```

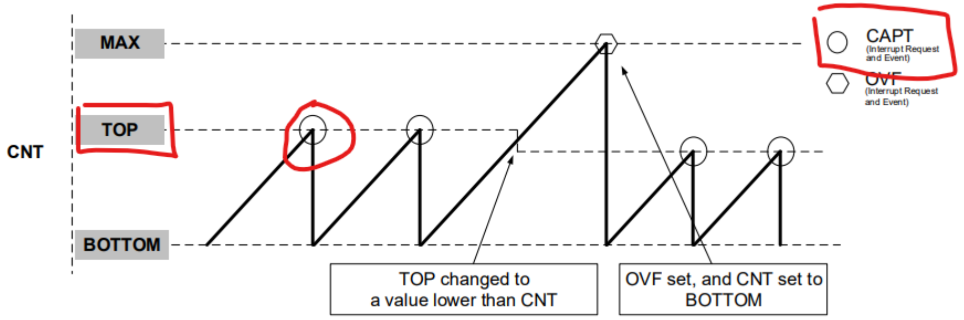
Task: TCB Periodic Interrupt

- To do
  - Select clk_tcb0 = F_CPU/1
  - Select CNTMODE = INT
  - Enable CAPT interrupt
  - Set TOP value
    - TOP = CCMP

22.3.3.1.1 Periodic Interrupt Mode

In the Periodic Interrupt mode, the counter counts to the capture value and restarts from BOTTOM. A CAPT interrupt and event is generated when the CNT is equal to TOP. If TOP is updated to a value lower than CNT, upon reaching MAX, an OVF interrupt and event is generated, and the counter restarts from BOTTOM.

Figure 22-3. Periodic Interrupt Mode



22.4 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0		RUNSTDBY	CASCADE	SYNCUPD		CLKSEL[2:0]		ENABLE
0x01	CTRLB	7:0		ASYNC	CCMPINIT	CCMPEN			CNTMODE[2:0]	
0x02 ... 0x03	Reserved									
0x04	EVCTRL	7:0		FILTER		EDGE				CAPTEI
0x05	INTCTRL	7:0							OVF	CAPT
0x06	INTFLAGS	7:0							OVF	CAPT
0x07	STATUS	7:0								RUN
0x08	DBGCTRL	7:0								DBGRUN
0x09	TEMP	7:0	TEMP[7:0]							
0x0A	CNT	7:0	CNT[7:0]							
		15:8	CNT[15:8]							
0x0C	CCMP	7:0	CCMP[7:0]							
		15:8	CCMP[15:8]							

Task: TCB Periodic Interrupt

- f = (clk_cpu / DIV) / TOP
- TOP = clk_cpu / (DIV * f)
- f = 1000 [Hz] = 1/1000 [s]
- TOP = 4e6 / (1 * 1000) = 4e3
  - This is within the bounds of 16 bits
  - 2^16-1 = 65535

- Technically 1 clock cycle is used to restart the counter
  - Let's ignore that for now

22.5.10 Capture/Compare

Name: CCMP
Offset: 0x0C
Reset: 0x00
Property: -

The TCBn.CCMPL and TCBn.CCMPH register pair represents the 16-bit value TCBn.CCMP. The low byte [7:0] (suffix L) is accessible at the original offset. The high byte [15:8] (suffix H) can be accessed at offset + `0x01`.

This register has different functions depending on the mode of operation:

- For Capture operation, these registers contain the captured value of the counter at the time the capture occurs
- In Periodic Interrupt, Time-Out Check and Single-Shot mode, this register acts as the TOP value
- In 8-bit PWM mode, TCBn.CCMPL and TCBn.CCMPH act as two independent registers: The period of the waveform is controlled by CCMPL, while CCMPH controls the duty cycle.

Bit	15	14	13	12	11	10	9	8
	CCMP[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	CCMP[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 15:8 – CCMP[15:8] Capture/Compare Value High Byte
These bits hold the MSB of the 16-bit compare, capture and top value.

Bits 7:0 – CCMP[7:0] Capture/Compare Value Low Byte
These bits hold the LSB of the 16-bit compare, capture and top value.

Task: TCB Periodic Interrupt

22.4 Register Summary

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0		RUNSTDBY	CASCADE	SYNCUPD		CLKSEL[2:0]		ENABLE
0x01	CTRLB	7:0		ASYNC	CCMPINIT	CCMPEN			CNTMODE[2:0]	
0x02 ... 0x03	Reserved									
0x04	EVCTRL	7:0		FILTER		EDGE				CAPTEI
0x05	INTCTRL	7:0							OVF	CAPT
0x06	INTFLAGS	7:0							OVF	CAPT
0x07	STATUS	7:0								RUN
0x08	DBGCTRL	7:0								DBGRUN
0x09	TEMP	7:0	TEMP[7:0]							
0x0A	CNT	7:0	CNT[7:0]							
		15:8	CNT[15:8]							
0x0C	CCMP	7:0	CCMP[7:0]							
		15:8	CCMP[15:8]							

```
// Set PC6 as output
PORTC.DIRSET    = PIN6_bm;

// Set CNTMODE for TCB
TCB0.CTRLB      = TCB_CNTMODE_INT_gc;

// Enable TCB0 overflow interrupt
TCB0.INTCTRL    = TCB_CAPT_bm;

// Calculate the TCB TOP Value
TCB0.CCMP       =  (uint16_t)( ((float)F_CPU) / (1000.0) - 0.5 );
```

Alternatively set TCB0.CCMP = 4000

```
// Enable TCB and set clock source to F_CPU DIV1
TCB0.CTRLA      = TCB_CLKSEL_DIV1_gc
                | TCB_ENABLE_bm;
```

Task: TCB Periodic Interrupt

```c
#define F_CPU 4000000

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

volatile uint32_t millis = 0;

int main(void)
{
    // Set PC6 as output
    PORTC.DIRSET    = PIN6_bm;

    // Set CNTMODE for TCB
    TCB0.CTRLB      = TCB_CNTMODE_INT_gc;

    // Enable TCB0 overflow interrupt
    TCB0.INTCTRL    = TCB_CAPT_bm;

    // Calculate the TCB TOP Value
    TCB0.CCMP       =  (uint16_t)( ((float)F_CPU) / (1000.0) - 0.5 );

    // Enable TCB and set clock source to F_CPU DIV1
    TCB0.CTRLA      = TCB_CLKSEL_DIV1_gc
                    | TCB_ENABLE_bm;

    // Set Enable (Global) Interrupts
    sei();
```

```c
uint32_t led_millis_last = 0, led_millis_delay = 500;
while (1)
{
    // Check if 500 ms has passed
    if ((millis - led_millis_last) >= led_millis_delay)
    {
        // Update last millisecond value
        led_millis_last = millis;

        PORTC.OUTTGL = PIN6_bm;
    }
}


ISR(TCB0_INT_vect)
{
    // Clear interrupt flag
    TCB0.INTFLAGS = TCB_CAPT_bm;

    millis++;
}
```

Task: PWM

- Final task for today
- Use TCA to control a servomotor
- TCA is bit more complicated than TCB
  - Fear not, it is actually not that hard to get going
  - Most of these registers can be ignored for now

21.4 Register Summary - Single Mode

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0	RUNSTDBY				CLKSEL[2:0]			ENABLE
0x01	CTRLB	7:0		CMP2EN	CMP1EN	CMP0EN	ALUPD	WGMODE[2:0]		
0x02	CTRLC	7:0						CMP2OV	CMP1OV	CMP0OV
0x03	CTRLD	7:0								SPLITM
0x04	CTRLECLR	7:0					CMD[1:0]		LUPD	DIR
0x05	CTRLESET	7:0					CMD[1:0]		LUPD	DIR
0x06	CTRLFCLR	7:0					CMP2BV	CMP1BV	CMP0BV	PERBV
0x07	CTRLFSET	7:0					CMP2BV	CMP1BV	CMP0BV	PERBV
0x08	Reserved									
0x09	EVCTRL	7:0	EVACTB[2:0]			CNTBEI	EVACTA[2:0]			CNTAEI
0x0A	INTCTRL	7:0		CMP2	CMP1	CMP0				OVF
0x0B	INTFLAGS	7:0		CMP2	CMP1	CMP0				OVF
0x0C ... 0x0D	Reserved									
0x0E	DBGCTRL	7:0								DBGRUN
0x0F	TEMP	7:0	TEMP[7:0]							
0x10 ... 0x1F	Reserved									
0x20	CNT	7:0	CNT[7:0]							
		15:8	CNT[15:8]							
0x22 ... 0x25	Reserved									
0x26	PER	7:0	PER[7:0]							
		15:8	PER[15:8]							
0x28	CMP0	7:0	CMP[7:0]							
		15:8	CMP[15:8]							
0x2A	CMP1	7:0	CMP[7:0]							
		15:8	CMP[15:8]							
0x2C	CMP2	7:0	CMP[7:0]							
		15:8	CMP[15:8]							
0x2E ... 0x35	Reserved									
0x36	PERBUF	7:0	PERBUF[7:0]							
		15:8	PERBUF[15:8]							
0x38	CMP0BUF	7:0	CMPBUF[7:0]							
		15:8	CMPBUF[15:8]							
0x3A	CMP1BUF	7:0	CMPBUF[7:0]							
		15:8	CMPBUF[15:8]							
0x3C	CMP2BUF	7:0	CMPBUF[7:0]							
		15:8	CMPBUF[15:8]							

Task: PWM

- Controlling a servo motor
  - Pulse-Width controlled
    - On-time determines position
    - Dependent on servomotor
    - Typically 1000us to 2000us
      - For our servo this is 500us to 2500us
  - Pulse repetition
    - Usually 20 ms = 50 Hz
- Our servomotor has three pins
  - 5V
  - GND
  - SIG – This is the PWM signal that controls the position

Selection Guide for Clutch Servo						
Model	6kg 180°	6kg 300°	9g 180°	9g 300°	2kg 180°	2kg 300°
SKU	SER0051	SER0057	SER0049	SER0053	SER0050	SER0056
Operating Voltage	4.8-6VDC	4.8-6V DC	4.8-6V DC	4.8-6V DC	4.8-6V DC	4.8-6V DC
Quiescent Current	≤10mA at 6.0V	≤10mA at 6.0V	≤8mA at 6.0V	≤8mA at 6.0V	≤8mA at 6.0V	≤8mA at 6.0V
No-load Current	≤60mA at 6.0V	≤60mA at 6.0V	≤50mA at 4.8V ≤60mA at 6.0V	≤50mA at 4.8V ≤60mA at 6.0V	≤110mA at 4.8V ≤120mA at 6.0V	≤110mA at 4.8V ≤120mA at 6.0V
Stall Current	≤1.65A at 6.0V	≤1.65A at 6.0V	≤550mA at 4.8V ≤650mA at 6.0V	≤550mA at 4.8V ≤650mA at 6.0V	≤700mA at 4.8V ≤800mA at 6.0V	≤700mA at 4.8V ≤800mA at 6.0V
Rated Torque	≥4.4kg.com at 6.0V	≥4.4kg.com at 6.0V	≥0.32kgf·cm at 4.8V ≤0.4kgf·cm at 6.0V	≥0.32kgf·cm at 4.8V ≤0.4kgf·cm at 6.0V	≥0.45kgf·cm at 4.8V ≥0.55kgf·cm at 6.0V	≥0.45kgf·cm at 4.8V ≥0.55kgf·cm at 6.0V
Stall Torque	≥6kg·cm at 6.0V	≥6kg·cm at 6.0V	≥1.0kgf·cm at 4.8V ≤1.2kgf·cm at 6.0V	≥1.0kgf·cm at 4.8V ≤1.2kgf·cm at 6.0V	≥1.6kgf·cm at 4.8V ≥2.0kgf·cm at 6.0V	≥1.6kgf·cm at 4.8V ≥2.0kgf·cm at 6.0V
Operating Angle	180°±10°	300°±10°	180°±10°	300°±10°	180°±10°	300°±10°
Pulse Width Range	500~2500μs	500~2500μs	500~2500μs	500~2500μs	500~2500μs	500~2500μs
Communication Mode	PWM	PWM	PWM	PWM	PWM	PWM

Task: PWM

- We know
  - Period = 20 ms = 50 Hz
  - On time = 500 to 2500 us
- To get the best resolution we need to have a large TOP value
  - TOP = f_clk_tca / f_pwm
  - For f_clk_tca = 4 MHz we get:
    4e6 / 50 = 80000
    This is larger than 2^16 !!!
  - For f_clk_tca = 2 MHz we get
    40000 < 2^16
- Prescaler value must be 2 when F_CPU = 4 MHz
- TOP must be 40000

- Now for ON time
  - 500 us = 2000 Hz
    - CMP = 2 MHz / 2000 Hz = 1000
  - 2500 us = 400 Hz
    - CMP = 2 MHz / 400 Hz = 5000
  - General formula
    - duty_cycle = 0 to 100
    - CMP = duty_cycle*((5000-1000)/100) + 1000
    - Simplify:
      CMP = duty_cycle*40 + 1000
    - Sanity check
      0% = 0 * 40 + 1000 = 1000
      50% = 50 * 40 + 1000 = 3000
      100% = 100 * 40 + 1000 = 5000

Task: PWM

- Create a function that solves CMP value
  - TCA can use double buffering
  - By using double buffering the PWM signal is always valid

```c
void servo_set(uint8_t duty_cycle)
{
    // Optional guard
    if (duty_cycle > 100)
    {
        duty_cycle = 100;
    }

    // Calculate compare value
    uint16_t cmp = duty_cycle * 40 + 1000;

    // Update CMPxBUF register. CMPx register us updated by buffer on overflow
    // This ensures that the pulse on time is always valid
    TCA0.SINGLE.CMP0BUF = cmp;
}
```

TCA is a unique peripheral as its registers are different depending on if it is in SPLIT mode, or SINGLE mode! We must therefore use TCAx.SINGLE.PERIPHERAL

Task: PWM

- Setup
  - Calculate and set TOP value (PER)
    - Already calculated!
  - Pin override enable (CTRLB)
    - Make sure TCA can override OUT value for a physical pin
  - Set TCA to PWM mode (CTRLB)
    - WGMODE bits
  - Select prescaler value (CTRLA)
    - CLKSEL bits
  - Enable TCA instance (CTRLA)
    - ENABLE bit

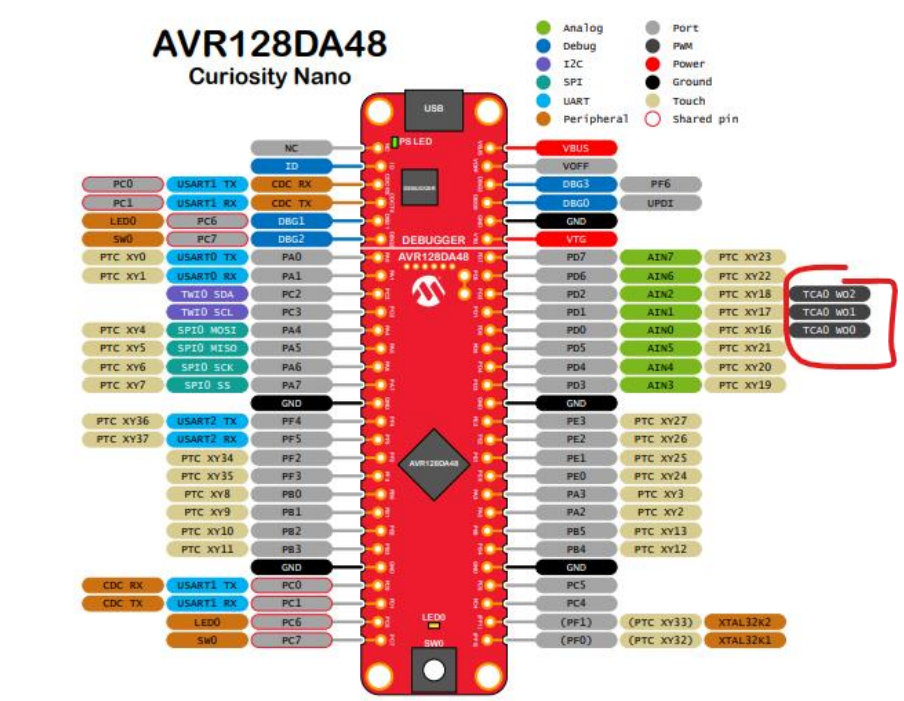
21.4 Register Summary - Single Mode

Offset	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x00	CTRLA	7:0	RUNSTDBY				CLKSEL[2:0]			ENABLE
0x01	CTRLB	7:0		CMP2EN	CMP1EN	CMP0EN	ALUPD	WGMODE[2:0]		
0x02	CTRLC	7:0						CMP2OV	CMP1OV	CMP0OV
0x03	CTRLD	7:0								SPLITM
0x04	CTRLECLR	7:0					CMD[1:0]		LUPD	DIR
0x05	CTRLESET	7:0					CMD[1:0]		LUPD	DIR
0x06	CTRLFCLR	7:0					CMP2BV	CMP1BV	CMP0BV	PERBV
0x07	CTRLFSET	7:0					CMP2BV	CMP1BV	CMP0BV	PERBV
0x08	Reserved									
0x09	EVCTRL	7:0	EVACTB[2:0]			CNTBEI	EVACTA[2:0]			CNTAEI
0x0A	INTCTRL	7:0		CMP2	CMP1	CMP0				OVF
0x0B	INTFLAGS	7:0		CMP2	CMP1	CMP0				OVF
0x0C ... 0x0D	Reserved									
0x0E	DBGCTRL	7:0								DBGRUN
0x0F	TEMP	7:0	TEMP[7:0]							
0x10 ... 0x1F	Reserved									
0x20	CNT	7:0	CNT[7:0]							
		15:8	CNT[15:8]							
0x22 ... 0x25	Reserved									
0x26	PER	7:0	PER[7:0]							
		15:8	PER[15:8]							

Task: PWM

- Find which pins are physically connected to a TCA instance
- Development board pinout
  - PD0 – PD2 are optional connections (Multiplexed)
- Default connections
  - PA0 – PA2
  - We will use PA0
- WOn
  - WO0, WO1, WO2 available in all modes
  - WO3, WO4, WO5 available ONLY in SPLIT mode

Figure 4-1. AVR128DA48 Curiosity Nano Pinout

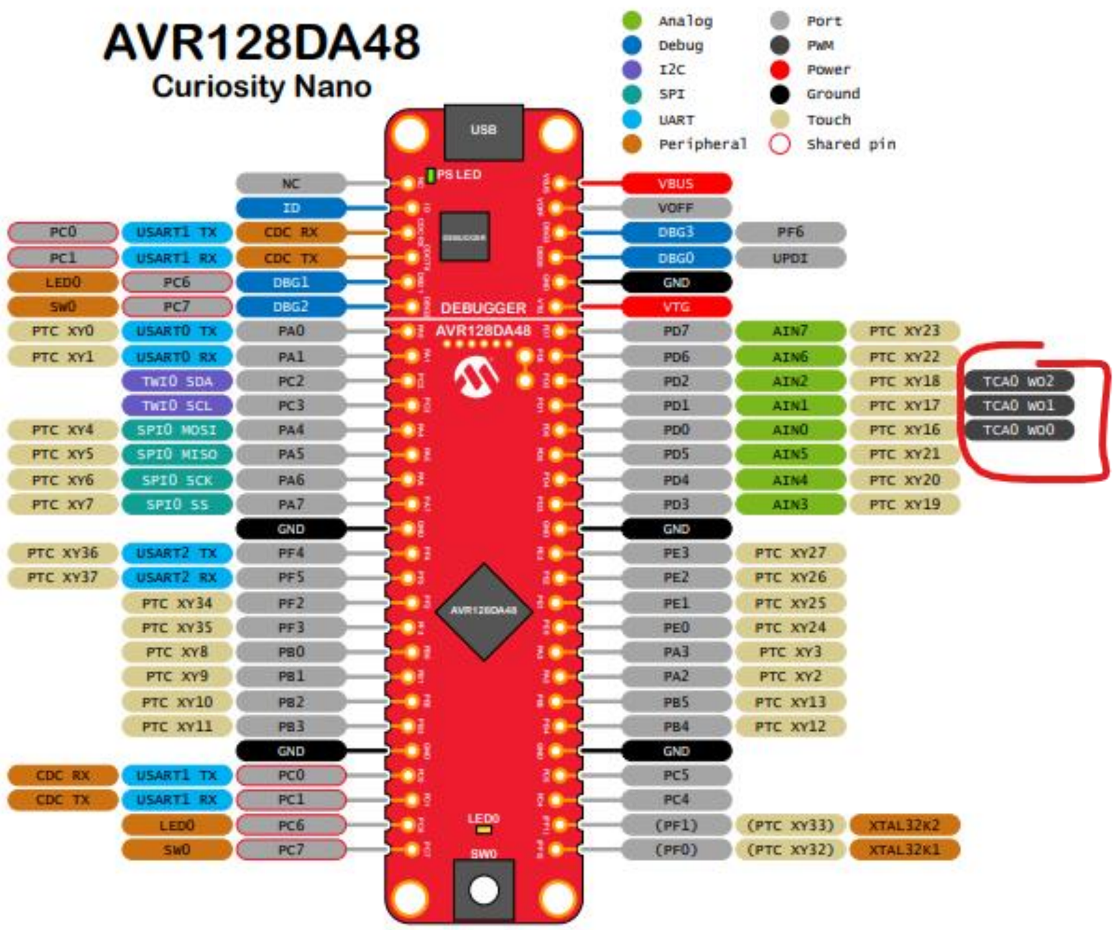


VQFN64/ TQFP64	VQFN48/ TQFP48	VQFN32/ TQFP32	SPDIP28/ SOIC28/ SSOP28	Pin name (1,2)	Special	ADC0	PTC	ACn	DAC0	ZCDn	USARTn	SPIn	TWIn(4)	TCA0	TCA1	TCBn	TCDn	EVSYS	CCL-LUTn
62	44	30	22	PA0	EXTCLK		XO/Y0				0,TxD			WO0					0,IN0
63	45	31	23	PA1			X1/Y1				0,RxD			WO1					0,IN1
64	46	32	24	PA2	TWI		X2/Y2				0,XCK		0,SDA(H)	WO2		0,WO		EVOUTA	0,IN2
1	47	1	25	PA3	TWI		X3/Y3				0,XDIR		0,SCL(H)	WO3		1,WO			0,OUT
2	48	2	26	PA4			X4/Y4				0,TxD[3]	0,MOSI		WO4			0,WOA		
3	1	3	27	PA5			X5/Y5				0,RxD[3]	0,MISO		WO5			0,WOB		
4	2	4	28	PA6			X6/Y6				0,XCK[3]	0,SCK					0,WOC		0,OUT[3]
5	3	5	1	PA7	CLKOUT		X7/Y7	0,OUT 1,OUT 2,OUT		0,OUT 1,OUT 2,OUT	0,XDIR[3]	0,$\overline{SS}$					0,WOD	EVOUTA [3]	
6				VDD															
7				GND															
8	4			PB0			X8/Y8				3,TxD			WO0[3]	WO0				4,IN0
9	5			PB1			X9/Y9				3,RxD			WO1[3]	WO1				4,IN1
10	6			PB2			X10/Y10				3,XCK		1,SDA(H)[3]	WO2[3]	WO2			EVOUTB	4,IN2
11	7			PB3			X11/Y11				3,XDIR		1,SCL(H)[3]	WO3[3]	WO3				4,OUT
12	8			PB4			X12/Y12				3,TxD[3]	1,MOSI[3]		WO4[3]	WO4	2,WO[3]	0,WOA[3]		
13	9			PB5			X13/Y13				3,RxD[3]	1,MISO[3]		WO5[3]	WO5	3,WO	0,WOB[3]		
14				PB6			X14/Y14				3,XCK[3]	1,SCK[3]	1,SDA(C)[3]				0,WOC[3]		4,OUT[3]
15				PB7			X15/Y15				3,XDIR[3]	1,$\overline{SS}$[3]	1,SCL(C)[3]				0,WOD[3]	EVOUTB [3]	
16	10	6	2	PC0							1,TxD	1,MOSI		WO0[3]		2,WO			1,IN0
17	11	7	3	PC1							1,RxD	1,MISO		WO1[3]		3,WO[3]			1,IN1

Task: PWM

- TCA0 is used
  - We use TCA0 in SINGLE mode
    - Only WO0, WO1, WO2 available
  - We can connect three servo-motors because WO0, WO1, WO2 are independent!
  - Let us connect to WO0

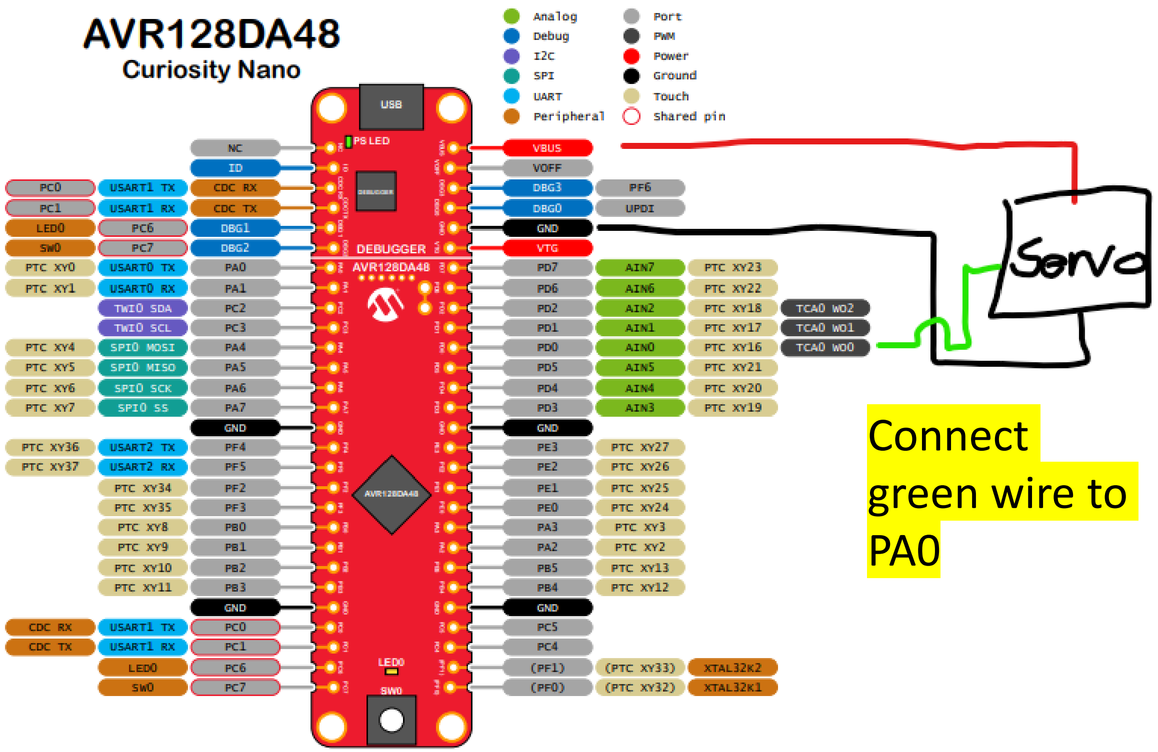
Figure 4-1. AVR128DA48 Curiosity Nano Pinout



Task: PWM

- Connect servo
  - VBUS to Supply pin (RED)
    - VBUS is always 5.0 V
  - GND to GND (BROWN)
  - Control signal to PA0 (YELLOW)
- GPIO setup
  - Make PA0 an OUTPUT
    - Set DIR register
  - For TCA0 to override PA0 we must set CMP0EN in TCA0.CTRLB

Figure 4-1. AVR128DA48 Curiosity Nano Pinout



Connect green wire to PA0

21.5.2 Control B - Normal Mode

Name: CTRLB
Offset: 0x01
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
		CMP2EN	CMP1EN	CMP0EN	ALUPD	WGMODE[2:0]		
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bits 4, 5, 6 – CMPEN Compare n Enable
In the FRQ and PWM Waveform Generation modes, the Compare n Enable (CMPnEN) bits will make the waveform output available on the pin corresponding to WOn.

Value	Description
0	Waveform output WOn will not be available on the corresponding pin
1	Waveform output WOn will override the output value of the corresponding pin

Task: PWM

- To Do
  - Set PA0 as output
  - PER
    - Set period
  - CTRLB
    - Enable pin override on PA0
    - Set waveform mode to PWM single-slope
      - Only up-count
  - CTRLA
    - Set clock to F_CPU / 2
    - Enable TCA0

Bits 2:0 – WGMODE[2:0] Waveform Generation Mode

This bit field selects the Waveform Generation mode and controls the counting sequence of the counter, TOP value, UPDATE condition, interrupt condition, and the type of waveform generated. No waveform generation is performed in the Normal mode of operation. The waveform generator output will only be directed to the port pins if setting the corresponding CMPnEN bit for all other modes. The port pin direction must be set as output.

Table 21-7. Timer Waveform Generation Mode

Value	Group Configuration	Mode of Operation	TOP	UPDATE	OVF
0x0	NORMAL	Normal	PER	TOP[1]	TOP[1]
0x1	FRQ	Frequency	CMP0	TOP[1]	TOP[1]
0x2	-	Reserved	-	-	-
0x3	SINGLESLOPE	Single-slope PWM	PER	BOTTOM	BOTTOM
0x4	-	Reserved	-	-	-
0x5	DSTOP	Dual-slope PWM	PER	BOTTOM	TOP
0x6	DSBOTH	Dual-slope PWM	PER	BOTTOM	TOP and BOTTOM
0x7	DSBOTTOM	Dual-slope PWM	PER	BOTTOM	BOTTOM

```
void servo_setup()
{
    #define TCA_CLKDIV 2
    #define TCA_FREQ 50.0
    // Calculate TOP value for TCA
    TCA0.SINGLE.PER     = (uint16_t)(  (((float)F_CPU) / ((float)TCA_CLKDIV))  / TCA_FREQ - 0.5  );

    TCA0.SINGLE.CTRLB   = TCA_SINGLE_CMP0EN_bm              // Enable pin override on CMP0 - WO0 - PD0
                        | TCA_SINGLE_WGMODE_SINGLESLOPE_gc; // TCA0 in PWM mode (Single slope)

    TCA0.SINGLE.CTRLA   = TCA_SINGLE_CLKSEL_DIV2_gc         // Set F_TCA = F_CPU/2
                        | TCA_SINGLE_ENABLE_bm;
}

int main(void)
{
    PORTA.DIRSET    = PIN0_bm;

    servo_setup();
```

Task: PWM

- Test Servo motor!
  - Let the motor operate in peace
    - Otherwise the fuse on the development board may pop
- Optional tasks:
  - Connect enable servo motor control on another GPIO
  - Make the servo change position on button press

```
int main(void)
{
    PORTA.DIRSET    = PIN0_bm;

    servo_setup();

    while (1)
    {
        servo_set(0);   // Servo 0%
        _delay_ms(500);

        servo_set(50);  // Servo 50%
        _delay_ms(500);

        servo_set(100); // Servo 100%
        _delay_ms(500);
    }
}
```

Next time

- Learn how to use a basic sensor
- Learn how to use SPI to connect to other devices
- Focus on hands-on experience from next session

- Questions?