ECE3411

Lecture 7

Timers

Sung Yeul Park
Department of Electrical & Computer Engineering
University of Connecticut
Email: sung_yeul.park@uconn.edu

Adapted from John Chandy ECE3411 – Fall 2024

Learning Objectives

- Explain how timers work
- Explain why you would want to use a 16-bit timer instead of an 8-bit timer
- Write code to initialize a timer for a particular interrupt period
- Write an ISR for a timer
- Write code to handle debouncing with a timer driven state machine
- Explain the disadvantages of using `delay_ms`
- Convert a task with `delay_ms` calls into a series of subtasks called through timer-based interrupts

Timer / Counter

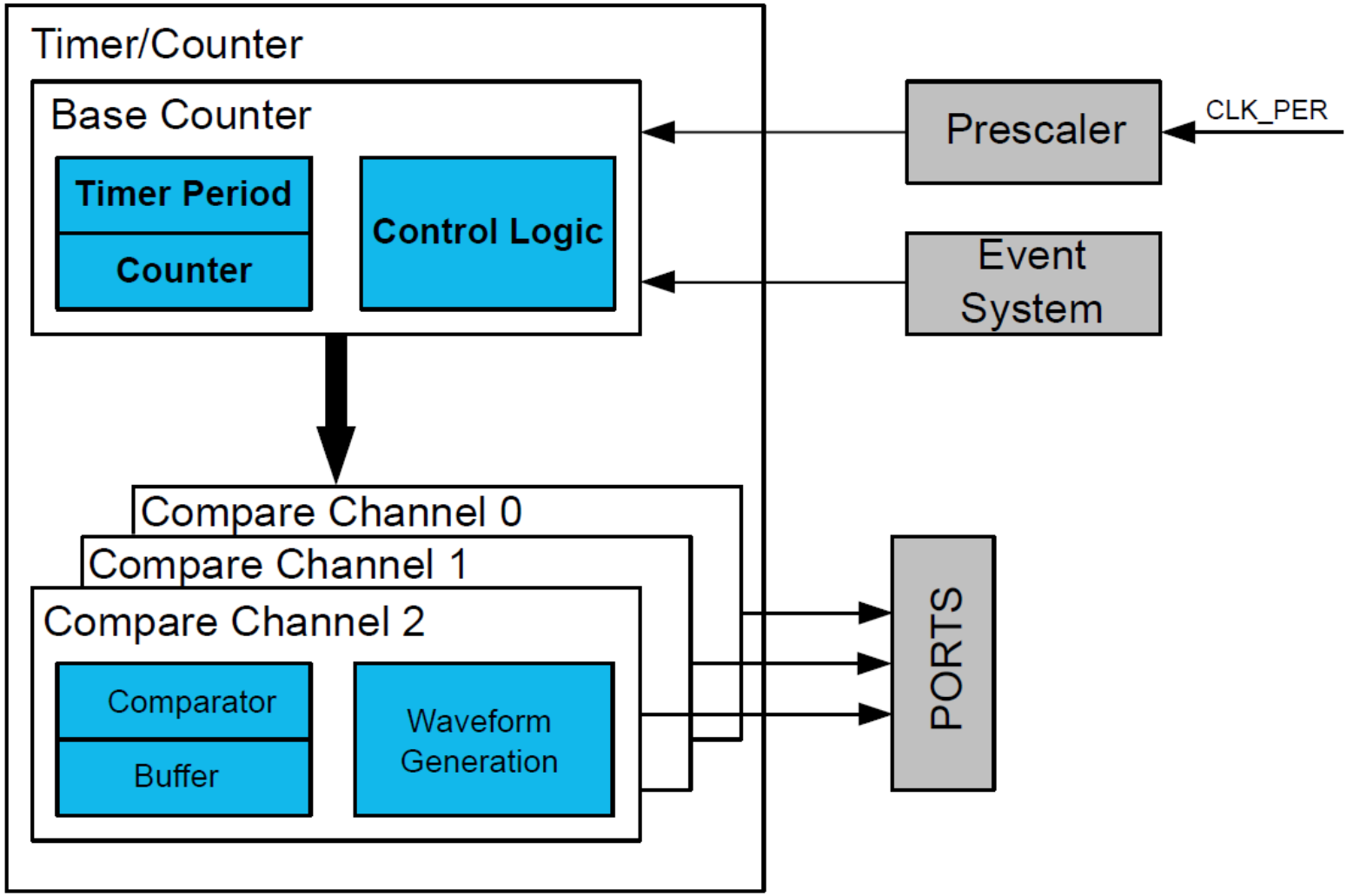
Timers

- Almost all microcontrollers have one or more timers that count time independently of the code
  - Timers avoid the use of delay functions
- Timer counter increments every clock cycle
  - Clock frequency can be altered
- Timers allow you to trigger precise time-based events
- Triggers happen when counter equals some predefined value or when counter overflows
- By varying the clock frequency or the counter trigger value, you can define a wide range of timer periods

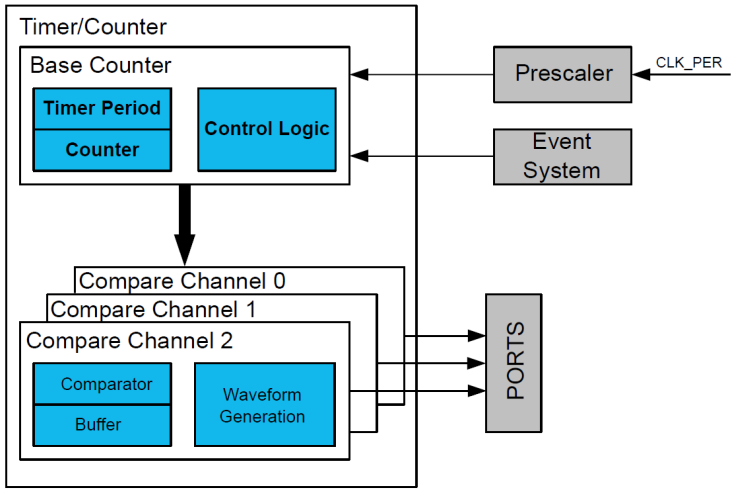
AVR Timer Types

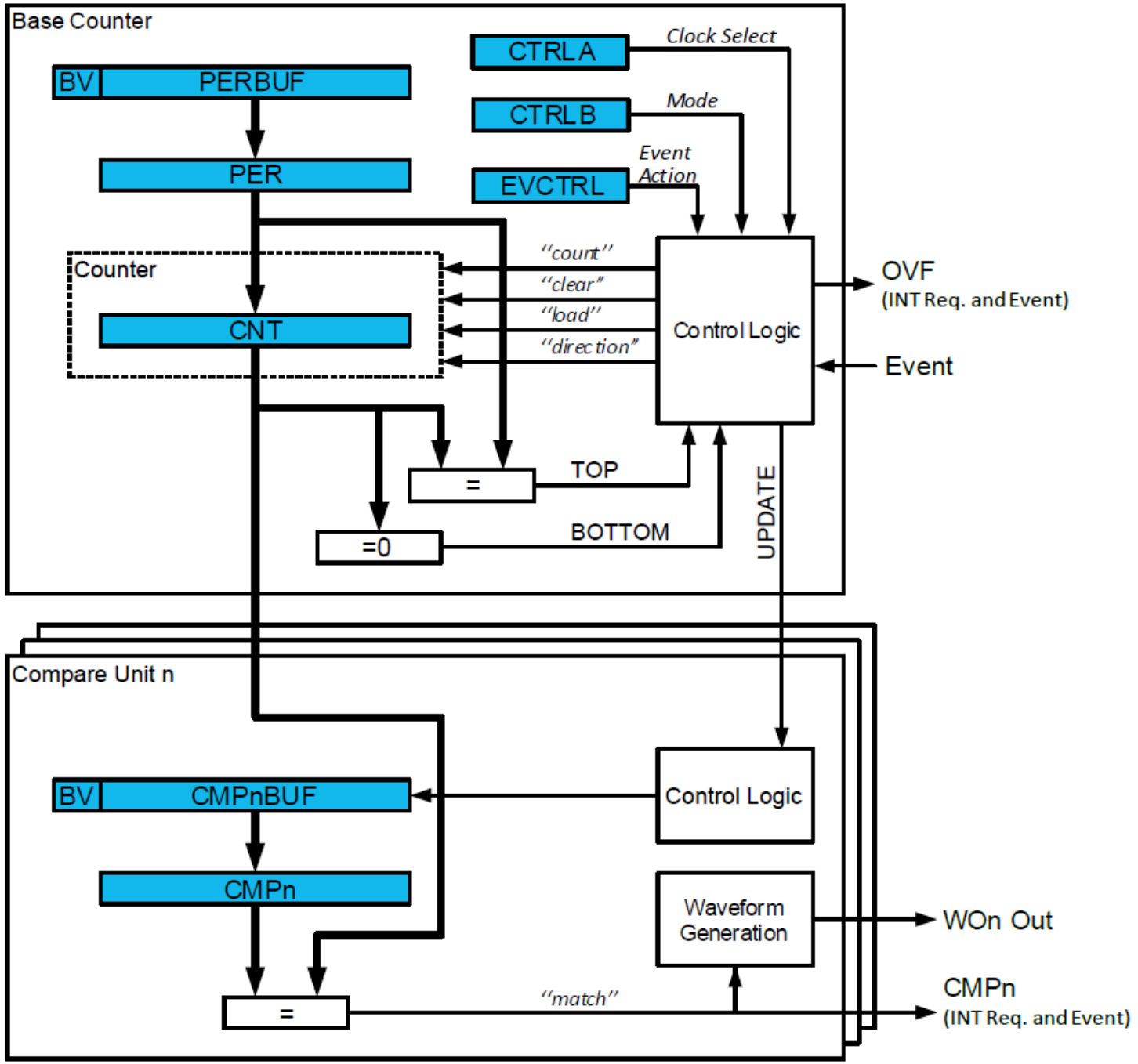
	Type A	Type B	Type D
Resolution	16 bit (0- - 65535)	16 bit (0- - 65535)	12 bit (0- - 65535)
Module names	TCA0	TCB0, TCB1, TCB2	TCD0
Key Features	- Single 16-bit timer - Split mode (two 8-bit timers) - 3 PWM channels - Event system support	- One-shot, periodic, and input capture modes - Event system support - High-resolution timing	- Advanced waveform generation - Dead time insertion - Fault protection - Complementary output
Use Cases	- PWM generation - Frequency measurement - Motor control	- Pulse width measurement - Periodic interrupts - Signal timing	- Power electronics - Full-bridge motor control - Inverter control

AVR Timer A


Timer/Counter
Base Counter
Timer Period
Counter
Control Logic
Prescaler
CLK_PER
Event System
Compare Channel 0
Compare Channel 1
Compare Channel 2
Comparator
Buffer
Waveform Generation
PORTS

AVR Timer A


Timer/Counter
Base Counter
Timer Period
Counter
Control Logic
Prescaler
CLK_PER
Event System
Compare Channel 0
Compare Channel 1
Compare Channel 2
Comparator
Buffer
Waveform Generation
PORTS


Base Counter
BV
PERBUF
PER
Counter
CNT
CTRLA
Clock Select
CTRLB
Mode
EVCTRL
Event Action
"count"
"clear"
"load"
"direction"
Control Logic
OVF
(INT Req. and Event)
Event
=
TOP
=0
BOTTOM
UPDATE
Compare Unit n
BV
CMPnBUF
CMPn
Control Logic
Waveform Generation
WOn Out
=
"match"
CMPn
(INT Req. and Event)

AVR Timer A Initialization

- To start using the timer/counter in a basic mode, follow these steps:
- 1. Write a TOP value to the Period (TCAn.PER) register.
- 2. Enable the peripheral by writing a ‘1’ to the Enable (ENABLE) bit in the Control A (TCAn.CTRLA) register.
- The counter will start counting clock ticks according to the prescaler setting in the Clock Select (CLKSEL) bit field in TCAn.CTRLA.
- 3. Optional: By writing a ‘1’ to the Enable Counter Event Input A (CNTAEI) bit in the Event Control (TCAn.EVCTRL) register, events are counted instead of clock ticks.
- 4. The counter value can be read from the Counter (CNT) bit field in the Counter (TCAn.CNT) register.

Example Timer TCA0

- 16MHz, 1ms ticks:

```
// 1ms Timer TCA0 assuming F_CPU = 16MHz
void InitTCA0(void)
{
    TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc; // Normal mode
    TCA0.SINGLE.PER = 249;                           // Set number of ticks for period

    // Set Prescalar to 64 & enable timer.  Each tick is 4us
    TCA0.SINGLE.CTRLA |= (TCA_SINGLE_CLKSEL_DIV64_gc | TCA_SINGLE_ENABLE_bm);
}
```

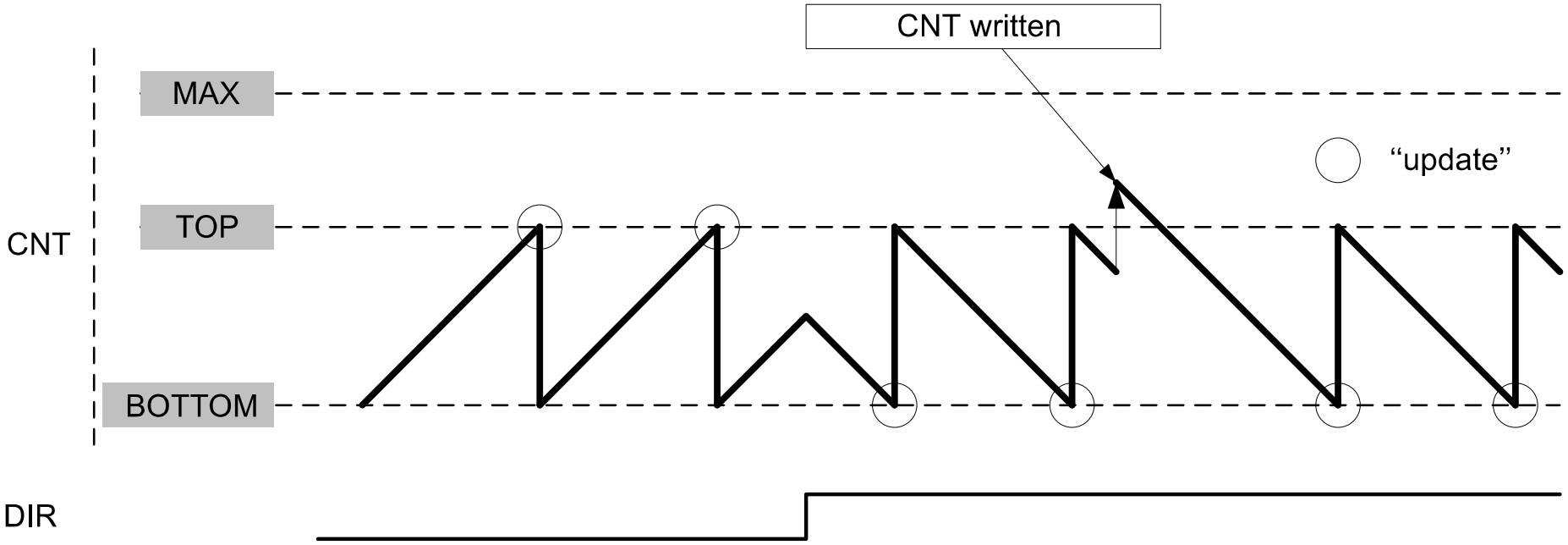
$$T = \frac{N \cdot (1 + PER)}{f_{clk-io}} = \frac{64 \cdot (1 + 249)}{16MHz} = 1ms$$

$$PER = \frac{T \cdot f_{clk-io}}{N} - 1 = \frac{1ms \cdot 16MHz}{64} - 1 = 249$$

Timer TCA0

- `TCA0.SINGLE.CNT` register holds the count
- `TCA0.SINGLE.PER` defines the match value (TOP)

Figure 21-4. Normal Operation



Timer TCA0

- You can vary the timer clock frequency with different dividers

$$N = 1, 2, 4, 8, 16, 64, 256, 1024$$

clock frequencies = 16 MHz, 8 MHz, 4 MHz, 2 MHz, 1 MHz, 250 kHz, 62500 Hz, 15625 Hz

periods = 62.5ns, 125ns, 250ns, 500ns, 1µs, 4µs, 16µs, 64µs

- The time it takes for the timer to reach the match value

$$T = \frac{N \cdot (1 + PER)}{f_{clk-io}}$$

TC0A.SINGLE.CTRLA

21.5.1 Control A - Normal Mode

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

Bit 7 – RUNSTDBY Run Standby
Writing a '1' to this bit will enable the peripheral to run in Standby sleep mode.

Bits 3:1 – CLKSEL[2:0] Clock Select
These bits select the clock frequency for the timer/counter.

Value	Name	Description
0x0	DIV1	$f_{TCA} = f_{CLK\_PER}$
0x1	DIV2	$f_{TCA} = f_{CLK\_PER}/2$
0x2	DIV4	$f_{TCA} = f_{CLK\_PER}/4$
0x3	DIV8	$f_{TCA} = f_{CLK\_PER}/8$
0x4	DIV16	$f_{TCA} = f_{CLK\_PER}/16$
0x5	DIV64	$f_{TCA} = f_{CLK\_PER}/64$
0x6	DIV256	$f_{TCA} = f_{CLK\_PER}/256$
0x7	DIV1024	$f_{TCA} = f_{CLK\_PER}/1024$

Bit 0 – ENABLE Enable

Value	Description
0	The peripheral is disabled
1	The peripheral is enabled

TC0A.SINGLE.CTRLB

21.5.2 Control B - Normal Mode

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 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. For all other modes, the waveform generator output will only be directed to the port pins if the corresponding CMPnEN bit has been set. 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

Note:

1. When counting up.

TCA0.SINGLE.INTCTRL

21.5.10 Interrupt Control Register - Normal Mode

Bit	7	6	5	4	3	2	1	0
		CMP2	CMP1	CMP0				OVF
Access		R/W	R/W	R/W				R/W
Reset		0	0	0				0

Bit 6 – CMP2 Compare Channel 2 Interrupt Enable
See CMP0.

Bit 5 – CMP1 Compare Channel 1 Interrupt Enable
See CMP0.

Bit 4 – CMP0 Compare Channel 0 Interrupt Enable
Writing the CMPn bit to '1' enables the interrupt from Compare Channel n.

Bit 0 – OVF Timer Overflow/Underflow Interrupt Enable
Writing the OVF bit to '1' enables the overflow/underflow interrupt.

TCA0.SINGLE.INTFLAGS

21.5.11 Interrupt Flag Register - Normal Mode

Bit	7	6	5	4	3	2	1	0
		CMP2	CMP1	CMP0				OVF
Access		R/W	R/W	R/W				R/W
Reset		0	0	0				0

Bit 6 – CMP2 Compare Channel 2 Interrupt Flag
See the CMP0 flag description.

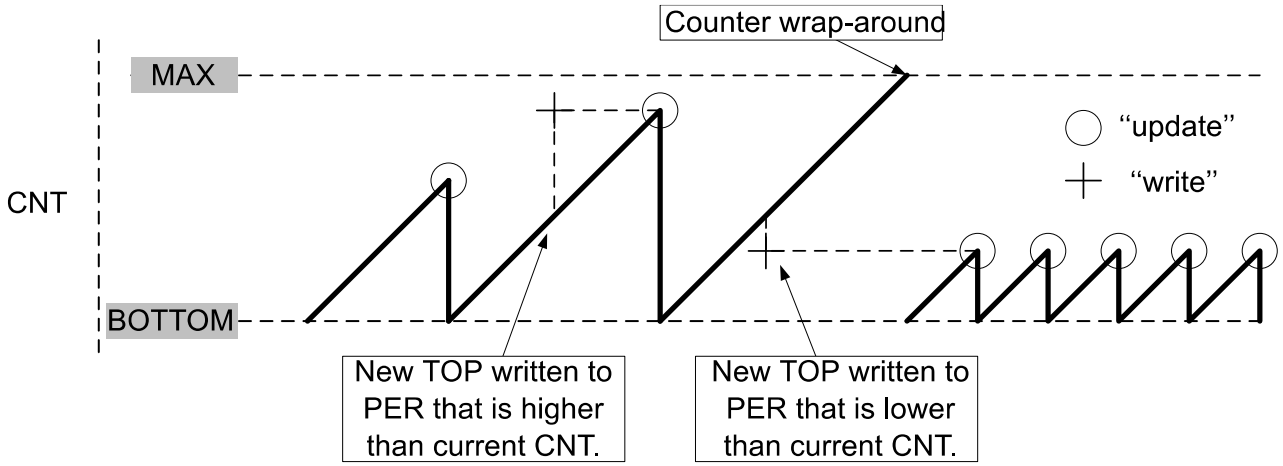
Bit 5 – CMP1 Compare Channel 1 Interrupt Flag
See the CMP0 flag description.

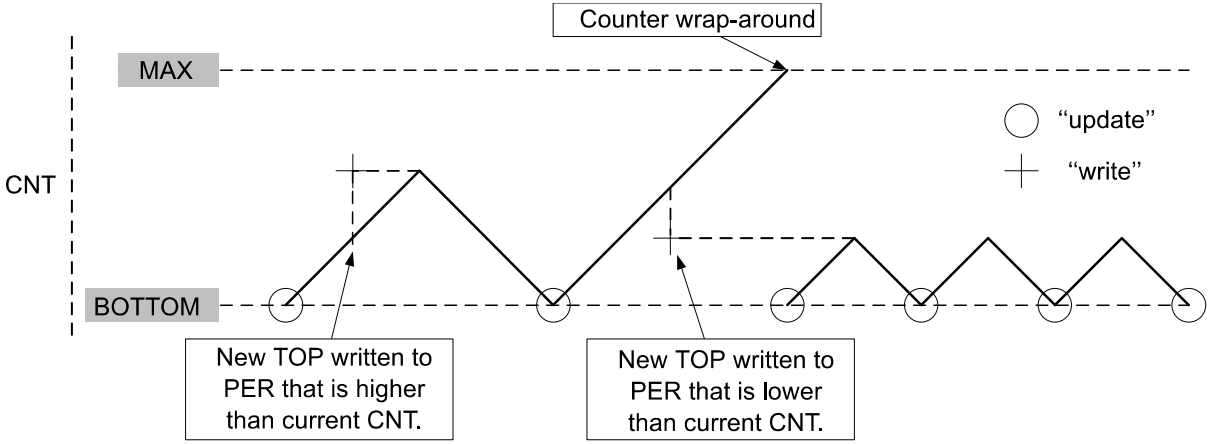
Bit 4 – CMP0 Compare Channel 0 Interrupt Flag
The Compare Interrupt (CMPn) flag is set on a compare match on the corresponding compare channel.
For all modes of operation, the CMPn flag will be set when a compare match occurs between the Count (TCAn.CNT) register and the corresponding Compare n (TCAn.CMPn) register. The CMPn flag is not cleared automatically. It will be cleared only by writing a '1' to its bit location.

Bit 0 – OVF Overflow/Underflow Interrupt Flag
This flag is set either on a TOP (overflow) or BOTTOM (underflow) condition, depending on the WGMODE setting.
The OVF flag is not cleared automatically. It will be cleared only by writing a '1' to its bit location.

Timer TCA0

- What happens when you change the PER value while the timer is running?

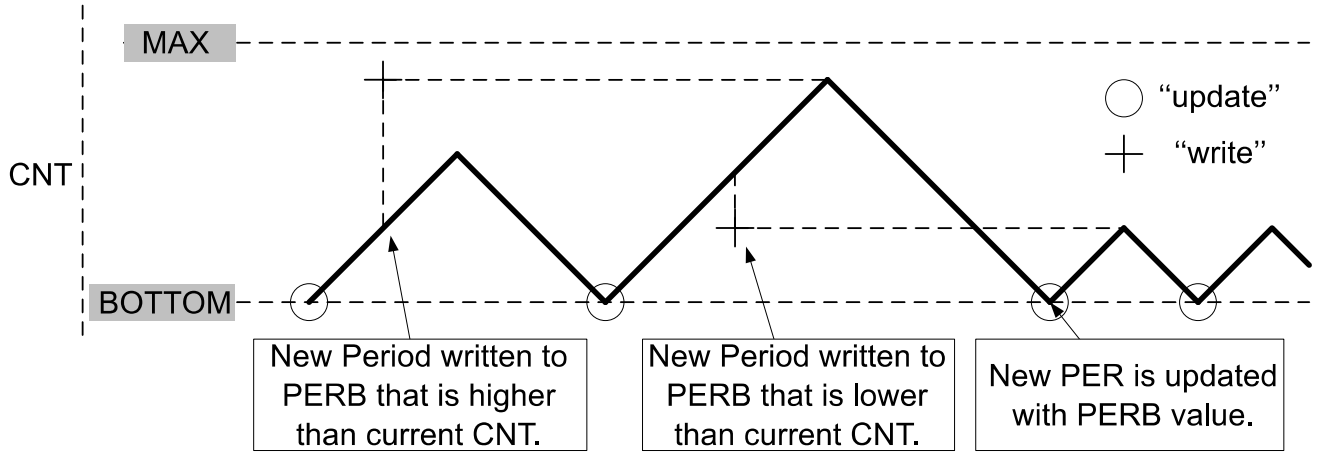




- Can cause the counter to go all the way to the MAX value
  - Depending on the application, could be a problem

Timer TCA0

- Fix is to “double-buffer” the PER register
  - `TCA0.SINGLE.PERBUF = 249;`



- The buffered value gets copied to the PER register at the “update”

Interrupts

Table 8-3. Interrupt Vector Mapping

Vector Number	Program Address (word)	Peripheral Source	Description
7	0x0E	TCA0 _OVF TCA0_LUNF	Normal: Timer/Counter Type A Overflow Interrupt Split: Timer/Counter Type A Low Underflow Interrupt
8	0x10	TCA0_HUNF	Normal: Unused Split: Timer/Counter Type A High Underflow Interrupt
9	0x12	TCA0_CMP0 TCA0_LCMP0	Normal: Timer/Counter Type A Compare 0 Interrupt Split: Timer/Counter Type A Low Compare 0 Interrupt
10	0x14	TCA0_CMP1 TCA0_LCMP1	Normal: Timer/Counter Type A Compare 1 Interrupt Split: Timer/Counter Type A Low Compare 1 Interrupt
11	0x16	TCA0_CMP2 TCA0_LCMP2	Normal: Timer/Counter Type A Compare 2 Interrupt Split: Timer/Counter Type A Low Compare 2 Interrupt
12	0x18	TCB0_INT	Timer Counter Type B Capture/Overflow Interrupt
13	0x1A	TCB1_INT	Timer Counter Type B Capture/Overflow Interrupt
14	0x1C	TCD0_OVF	Timer Counter Type D Overflow Interrupt
15	0x1E	TCD0_TRIG	Timer Counter Type D Trigger Interrupt

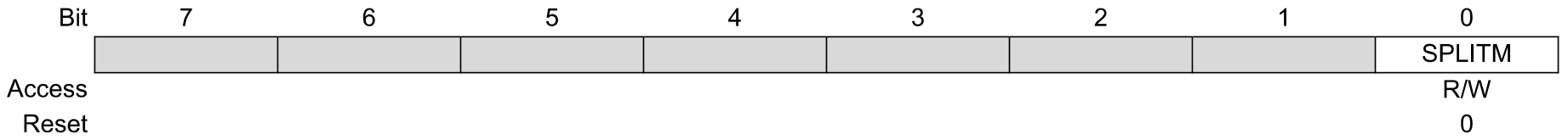
Timer interrupts

```
int EVENT_TIME = 500;
volatile int timerCount = EVENT_TIME;

ISR(TCA0_OVF_vect)
{
    if (timerCount >0) {timerCount--;}
    TCA0.SINGLE.INTFLAGS |= TCA_SINGLE_OVF_bm; // must clear the interrupt
}

// 1ms ISR for Timer TCA0 assuming F_CPU = 16MHz
void InitTCA0(void)
{
    TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc; // Normal mode
    TCA0.SINGLE.PER = 249;                           // Set number of ticks for period
    TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;         // Enable TCA0 Overflow ISR

    // Set Prescalar to 64 & enable timer.  Each tick is 4us
    TCA0.SINGLE.CTRLA |= (TCA_SINGLE_CLKSEL_DIV64_gc | TCA_SINGLE_ENABLE_bm);
}
```

This is very important. Otherwise, you will get continuously interrupted

Timer TCA0 Split Mode

- Splits the 16-bit timer into two 8-bit timers
  - No waveform generation
  - Different register names
    - **`TCA0.SPLIT.CTRLA` instead of `TCA0.SINGLE.CTRLA`**
    - **`TCA0.SPLIT.HPER` and `TCA0.SPLOT.LPER` instead of `TCA0.SINGLE.PER`**



Bit 0 – SPLITM Enable Split Mode
This bit sets the timer/counter in Split mode operation and will work as two 8-bit timer/counters. The register map will change compared to the normal 16-bit mode.

Timer TCA0 Split Mode

```
ISR(TCA0_LUNF_vect)
{
    if (timerCount >0) {timerCount--;}
    TCA0.SPLIT.INTFLAGS |= TCA_SINGLE_LUNF_bm; // must clear the interrupt
}

// 1ms ISR for Timer TCA0 assuming F_CPU = 16MHz
void InitTimerTCA0(void)
{
     TCA0.SPLIT.CTRLD = TCA_SPLIT_SPLITM_bm;     // split the registers
     TCA0.SPLIT.PERL = 249;                      // Set number of ticks for period
     TCA0.SPLIT.INTCTRL = TCA_SPLIT_LUNF_bm;     // Enable TCA0 Underflow ISR

     // Set Prescalar to 64 & enable timer.  Each tick is 4us
     TCA0.SPLIT.CTRLA |= (TCA_SPLIT_CLKSEL_DIV64_gc | TCA_SPLIT_ENABLE_bm);
}
```

TCB.CTRLA

24.5.1 Control A

Bit	7	6	5	4	3	2	1	0
		RUNSTDBY	CASCADE	SYNCUPD	CLKSEL[2:0]			ENABLE
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bits 3:1 – CLKSEL[2:0] Clock Select

Writing these bits selects the clock source for this peripheral.

Value	Name	Description
0x0	DIV1	CLK_PER
0x1	DIV2	CLK_PER / 2
0x2	TCA0	CLK_TCA from TCA0
0x3	TCA1	CLK_TCA from TCA1
0x4–0x6	-	Reserved
0x07	EVENT	Positive edge on event input

Bit 0 – ENABLE Enable

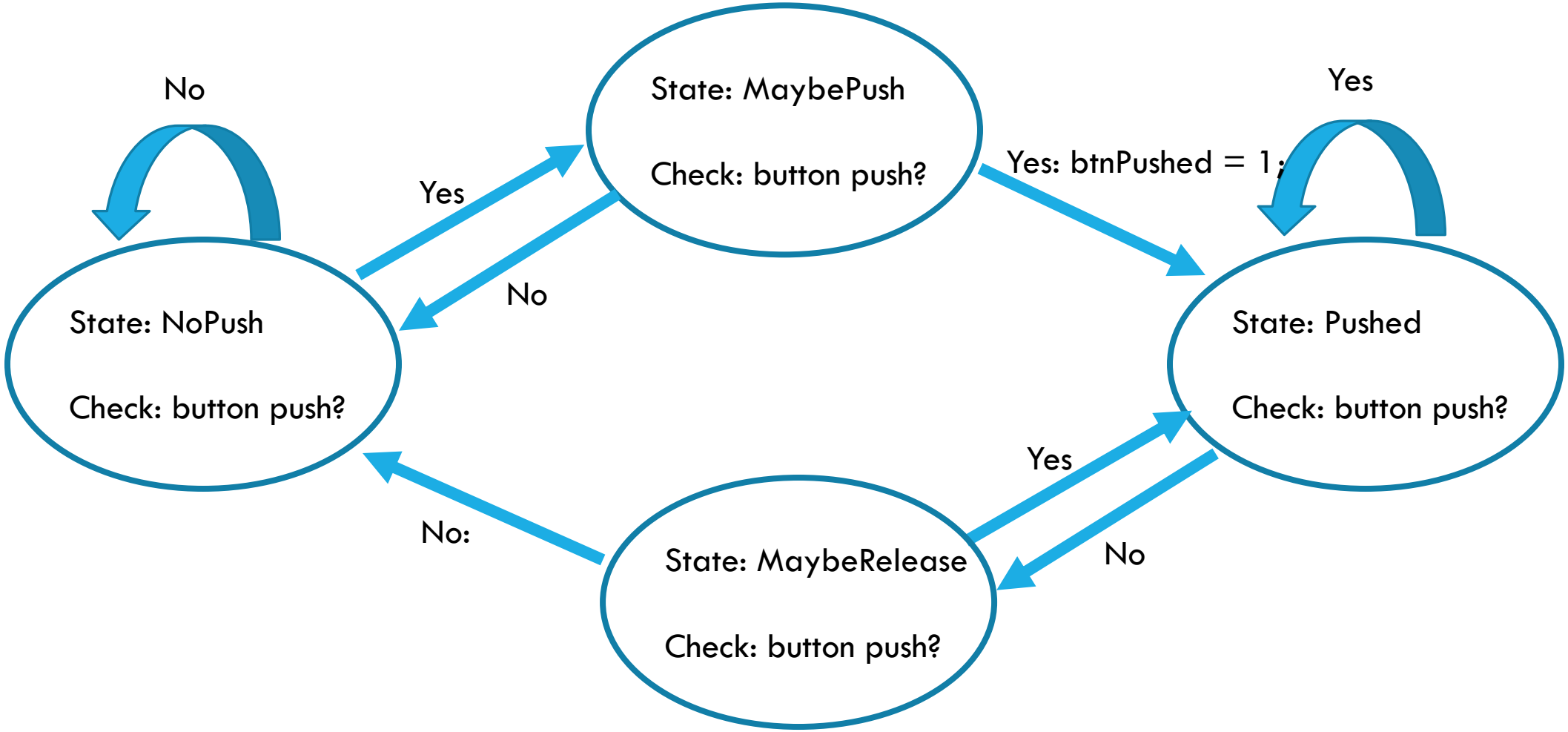
Writing this bit to '1' enables the Timer/Counter type B peripheral.

Set TOP value by changing the TCB.CCMP register

Timer-Driven State Machine

- State machines are a common programming paradigm that make it easier to deal with complex input behaviors
- If you always know the state of the program, you have less undefined behavior
- State transitions are usually defined by events (I/O, interrupts, etc.)
- Can be time driven as well
- Example: debouncing state machine

Debounce State Machine



Timer interrupts

```
#define DEBOUNCE_TIME 10
volatile int debounce_timerCount = DEBOUNCE_TIME;

ISR(TCA0_OVF_vect)
{
    if (debounce_timerCount >0) {
        debounce_timerCount--;
    } else {
        buttonSM();
        debounce_timerCount = DEBOUNCE_TIME;
    }
    TCA0.SINGLE.INTFLAGS |= TCA_SINGLE_OVF_bm; // must clear the interrupt
}

// 1ms ISR for Timer TCA0 assuming F_CPU = 16MHz
void InitTimerTCA0(void)
{
    TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc; // Normal mode
    TCA0.SINGLE.PER = 249;                           // Set number of ticks for period
    TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;         // Enable TCA0 Overflow ISR
    // Set Prescalar to 64 & enable timer.  Each tick is 4us
    TCA0.SINGLE.CTRLA |= (TCA_SINGLE_CLKSEL_DIV64_gc | TCA_SINGLE_ENABLE_bm);
}
```

Debounce State Machine

```
volatile bool btnPushed = false;

#define BTN (!(VPORTB.IN & PIN2_bm))

typedef enum {RELEASED, MAYBE_PUSHED,
              PUSHED, MAYBE_RELEASED
} btn_state_t;

void buttonSM(void)
{
  static btn_state_t state = RELEASED;
  switch (state)
  {
  case RELEASED:
     if (BTN)
        state = MAYBE_PUSHED;
     break;
```

```
  case MAYBE_PUSHED:
     if (BTN) {
        state = PUSHED;
        btnPushed = true;
     } else {
        state = RELEASED;
     }
     break;

  case PUSHED:
     if (!BTN)
        state = MAYBE_RELEASED;
     break;

  case MAYBE_RELEASED:
     if (BTN)
        state = PUSHED;
     else
        state = RELEASED;
     break;
  }
}
```

Debounce State Machine

```
int main()
{
    ...
    InitTCA0();

    while (1) {
        if (btnPushed == true) {
            btnPushed = false;
            handle_btn_push();
          }
     }
     return 0;
}
```

Lab practice #6: LED blink and button input without delay function

- 8 LEDs of port D as output
- Set up the blinking frequency of the 3rd LED to 3Hz and rest of the LEDs should be turned off.
- 2 buttons BTN1 and BTN2 as input
- Increase the blinking frequency of the LED by 1 Hz if button BTN1 is pressed
- Decrease the blinking frequency of the LED by 1 Hz if button BTN2 is pressed
  - Do not go below 1 Hz

- Use two buttons to set the frequency – one to increment and one to decrement.
- Every 5s, output the current frequency to the serial port
  - Do not use `delay_ms`
  - Implement as a second task

Changing the clock frequency

- Configuring AVR128DB48 to use 16MHz internal oscillator

```
#define F_CPU 16000000UL
void init_clock() {
      CPU_CCP = CCP_IOREG_gc;
      CLKCTRL.OSCHFCTRLA = CLKCTRL_FRQSEL_16M_gc;
}
```

- Configuring AVR128DB48 to use 16MHz external crystal – more accurate

```
#define F_CPU 16000000UL
void init_clock() {
      CPU_CCP = CCP_IOREG_gc;
      CLKCTRL.XOSCHFCTRLA = CLKCTRL_FRQRANGE_16M_gc |
                            CLKCTRL_ENABLE_bm;
      CPU_CCP = CCP_IOREG_gc;
      CLKCTRL.MCLKCTRLA = CLKCTRL_CLKSEL_EXTCLK_gc;
}
```

CCP: Configuration Change Protection

7.6.1 Configuration Change Protection

Name: CCP
Offset: 0x04
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	CCP[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 7:0 – CCP[7:0] Configuration Change Protection
Writing the correct signature to this bit field allows changing protected I/O registers or executing protected instructions within the next four CPU instructions executed.
All interrupts are ignored during these cycles. After these cycles are completed, the interrupts will automatically be handled again by the CPU, and any pending interrupts will be executed according to their level and priority.
When the protected I/O register signature is written, CCP[0] will read as '1' as long as the CCP feature is enabled.
When the protected self-programming signature is written, CCP[1] will read as '1' as long as the CCP feature is enabled.
CCP[7:2] will always read as '0'.

Value	Name	Description
0x9D	SPM	Allow Self-Programming
0xD8	IOREG	Unlock protected I/O registers

MCLKCTRLA: Main Clock Control A

12.5.1 Main Clock Control A

Name: MCLKCTRLA
Offset: 0x00
Reset: 0x00
Property: Configuration Change Protection

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

Bit 7 – CLKOUT Main Clock Out
This bit controls whether the main clock is available on the Main Clock Out (CLKOUT) pin or not, when the main clock is running.
This bit is cleared when a '0' is written to it or when a Clock Failure Detection (CFD) condition with the main clock as source occurs.
This bit is set when a '1' is written to it.

Value	Description
0	The main clock is not available on the CLKOUT pin
1	The main clock is available on the CLKOUT pin

Bits 3:0 – CLKSEL[3:0] Clock Select
This bit field controls the source for the Main Clock (CLK_MAIN).

Value	Name	Description
0x0	OSCHF	Internal high-frequency oscillator
0x1	OSC32K	32.768 kHz internal oscillator
0x2	XOSC32K	32.768 kHz external crystal oscillator
0x3	EXTCLK	External clock or external crystal, depending on the SELHF bit in XOSCHFCTRLA
Other	Reserved	Reserved

XOSCHFCTRLA: External High-Frequency Oscillator Control A(1)

12.5.12 External High-Frequency Oscillator Control A

Name: XOSCHFCTRLA
Offset: 0x20
Reset: 0x00
Property: Configuration Change Protection

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY		CSUTHF[1:0]		FRQRANGE[1:0]		SELHF	ENABLE
Access	R/W		R/W	R/W	R/W	R/W	R/W	R/W
Reset	0		0	0	0	0	0	0

Bit 7 – RUNSTDBY Run Standby
This bit controls whether the External High-Frequency Oscillator (XOSCHF) is always running or not, when the ENABLE bit is '1'.

Value	Description
0	The XOSCHF oscillator will only run when requested by a peripheral or by the main clock (1)
1	The XOSCHF oscillator will always run in Active, Idle and Standby sleep modes (2)

Notes:

1. The requesting peripheral, or the main clock, must take the oscillator start-up time into account.
2. The oscillator signal is only available if requested, and will be available after two XOSCHF cycles, if the initial crystal start-up time has already ended.

XOSCHFCTRLA: External High-Frequency Oscillator Control A(2)

Bits 5:4 – CSUTHF[1:0] Crystal Start-up Time
This bit field controls the start-up time for the External High-Frequency Oscillator (XOSCHF), when the Source Select (SELHF) bit is '0'.

Value	Name	Description
0x0	256	256 XOSCHF cycles
0x1	1K	1K XOSCHF cycles
0x2	4K	4K XOSCHF cycles
0x3	-	Reserved

Note: This bit field is read-only when the ENABLE bit or the External Crystal/Clock Status (XOSCHFS) bit in the Main Clock Status (MCLKSTATUS) register is '1'.

Bits 3:2 – FRQRANGE[1:0] Frequency Range
This bit field controls the maximum frequency supported for the external crystal. The larger the range selected, the higher the current consumption by the oscillator.

Value	Name	Description
0x0	8M	Max. 8 MHz XTAL frequency
0x1	16M	Max. 16 MHz XTAL frequency
0x2	24M	Max. 24 MHz XTAL frequency
0x3	32M	Max. 32 MHz XTAL frequency

Note: If a crystal with a frequency larger than the maximum supported CLK_CPU frequency is used and used as the main clock, it is necessary to divide it down by writing the appropriate configuration to the PDIV bit field in the Main Clock Control B register.

XOSCHFCTRLA: External High-Frequency Oscillator Control A(2)

Bit 1 – SELHF Source Select

This bit controls the source of the External High-Frequency Oscillator (XOSCHF).

Value	Name	Description
0	CRYSTAL	External Crystal on the XTALHF1 and XTALHF2 pins
1	EXTCLOCK	External Clock on the XTALHF1 pin

Note: This bit field is read-only when the ENABLE bit or the External Crystal/Clock Status (XOSCHFS) bit in the Main Clock Status (MCLKSTATUS) register is '1'.

Bit 0 – ENABLE Enable

This bit controls whether the External High-Frequency Oscillator (XOSCHF) is enabled or not.

Value	Description
0	The XOSCHF oscillator is disabled
1	The XOSCHF oscillator is enabled, and overrides normal port operation for the respective oscillator pins