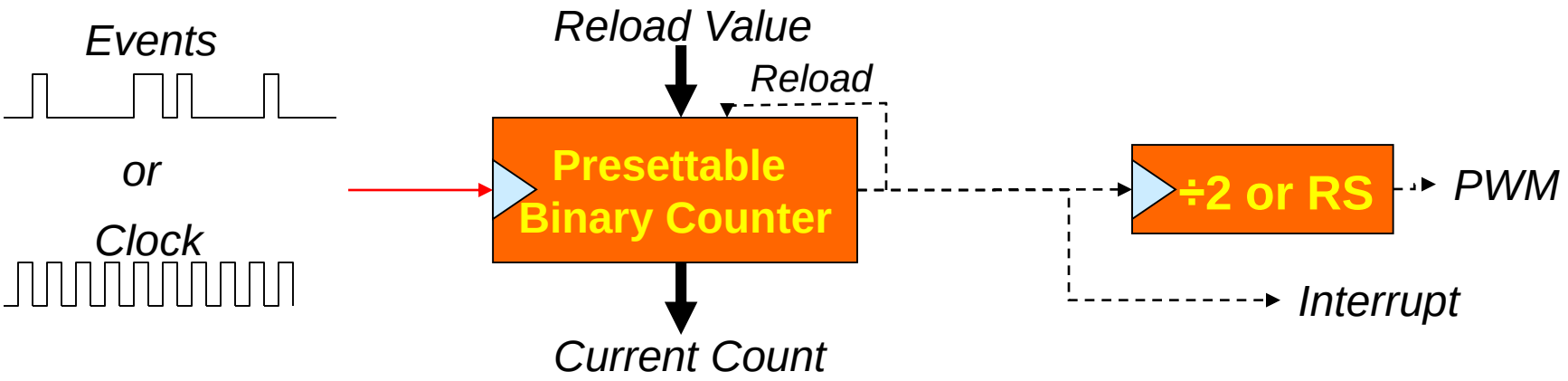
Timers and Event Counters

These lecture notes created by Mr. James B. (Jim) Carlson, NCSU

In These Notes . . .

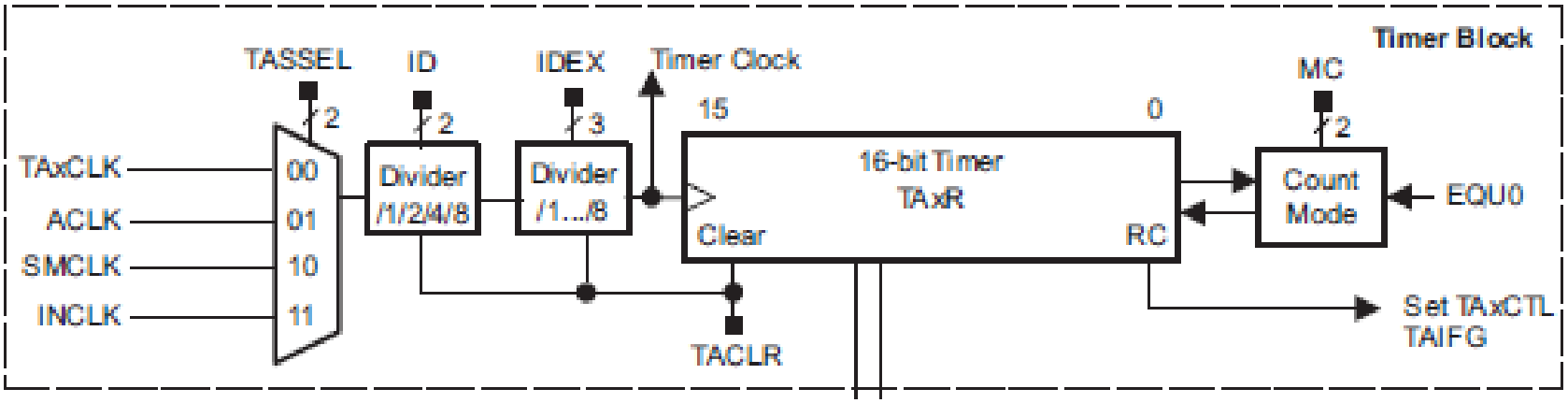
- We learn the basics of the Timer/Counter peripheral
  - Called generically as *timers*
- We examine how to set up the timers for different operation:
- We also examine how to use a microcontroller in the timer mode

Timer/Counter Introduction



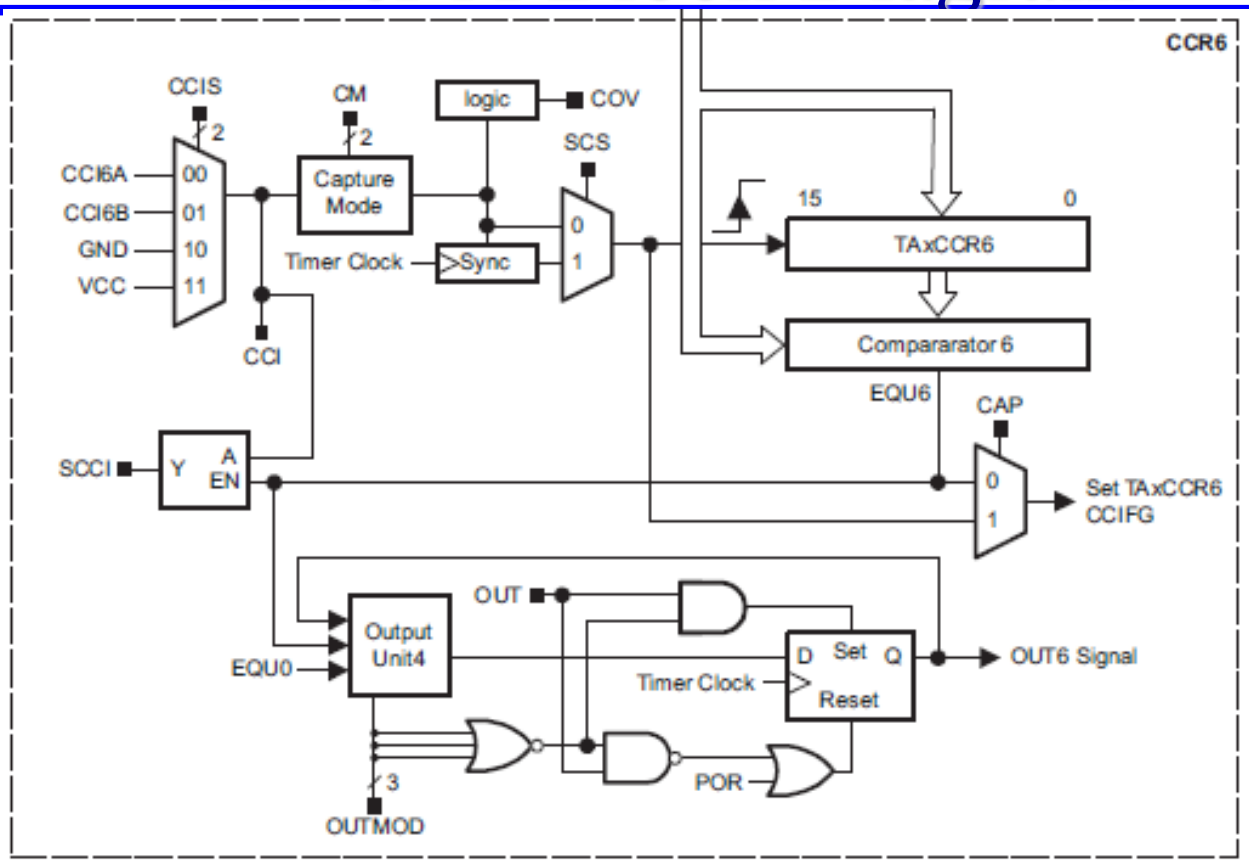
- Common peripheral for microcontrollers
- Based on presettable binary counter, enhanced with configurability
  - Count value can be read and written by MCU
  - Count direction can often be set to up, down, up-down, continuous
  - Counter's clock source can be selected
    - **Counter mode:** count **pulses** which indicate **events** (e.g. odometer pulses)
    - **Timer mode**: clock source is periodic, so counter value is proportional to **elapsed time** (e.g. stopwatch)
  - Counter's overflow/underflow action can be selected
    - Generate interrupt
    - Reload counter with special value and continue counting
    - Toggle hardware output signal
    - Stop!

Timer A Block Diagram



- Each timer has one input, which is selectable from several different sources.

Timer A Block Diagram



- Timer_A is a 16-bit timer/counter with up to seven capture/compare registers. Timer_A can support multiple capture/compares, PWM outputs, and interval timing. Timer_A also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

High-Level Timer A Block Diagram

- Timer_A features include:
  - Asynchronous 16-bit timer/counter with four operating modes
  - Selectable and configurable clock source
  - Up to seven configurable capture/compare registers
  - Configurable outputs with pulse width modulation (PWM) capability
  - Asynchronous input and output latching
  - Interrupt vector register for fast decoding of all Timer_A interrupts

16-Bit Timer Counter

- The 16-bit timer/counter register, TAxR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TAxR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.

- TAxR may be cleared by setting the TACLR bit. Setting TACLR also clears the clock divider state and resets the count direction for up/down mode to 0 going up.

Clock Source Select and Divider

- The timer clock can be sourced from ACLK, SMCLK, or externally via TAxCLK or INCLK. The clock source is selected with the TASSEL bits. The selected clock source may be passed directly to the timer or divided by 2, 4, or 8, using the ID bits.

- The selected clock source can be further divided by 2, 3, 4, 5, 6, 7, or 8 using the TAIDEX bits. The timer clock divider logic is reset when TACLR is set

Timer_A dividers

- After programming ID or TAIDEX bits, set the TACLR bit. This clears the contents of TAxR and resets the clock divider logic to a defined state.

- The clock dividers are implemented as down counters. Therefore, when the TACLR bit is cleared, the timer clock immediately begins clocking at the first rising edge of the Timer_A clock source selected with the TASSEL bits and continues clocking at the divider settings set by the ID and TAIDEX bits

Starting the Timer

- The timer may be started or restarted in the following ways:
  - The timer counts when MC > { 0 } and the clock source is active.
  - When the timer mode is either up or up/down, the timer may be stopped by writing 0 to TAxCCR0. The timer may then be restarted by writing a nonzero value to TAxCCR0. In this scenario, the timer starts incrementing in the up direction from zero.

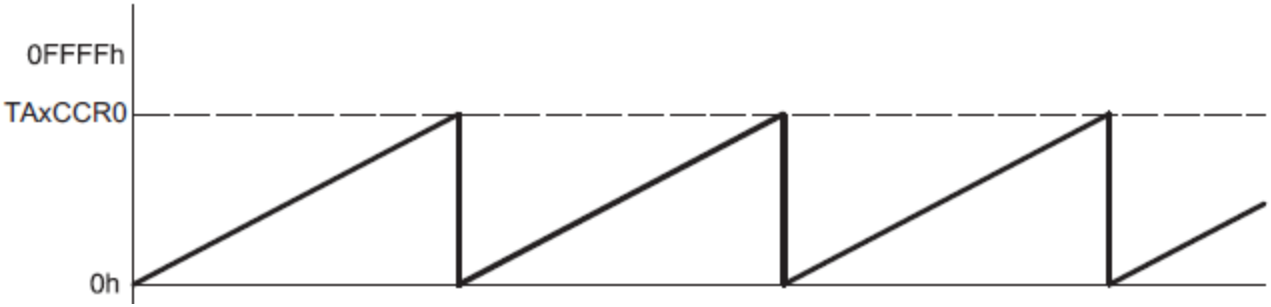
Timer Mode Control

- The timer has four modes of operation: stop, up, continuous, and up/down. The operating mode is selected with the MC bits.
- **MC Mode Description**

00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh.
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero.

Up Mode

- The up mode is used if the timer period must be different from 0FFFFh counts. The timer repeatedly counts up to the value of compare register TAxCCR0, which defines the period. The number of timer counts in the period is TAxCCR0 + 1. When the timer value equals TAxCCR0, the timer restarts counting from zero. If up mode is selected when the timer value is greater than TAxCCR0, the timer immediately restarts counting from zero.



TAxCCR0 CCIFG interrupt flag

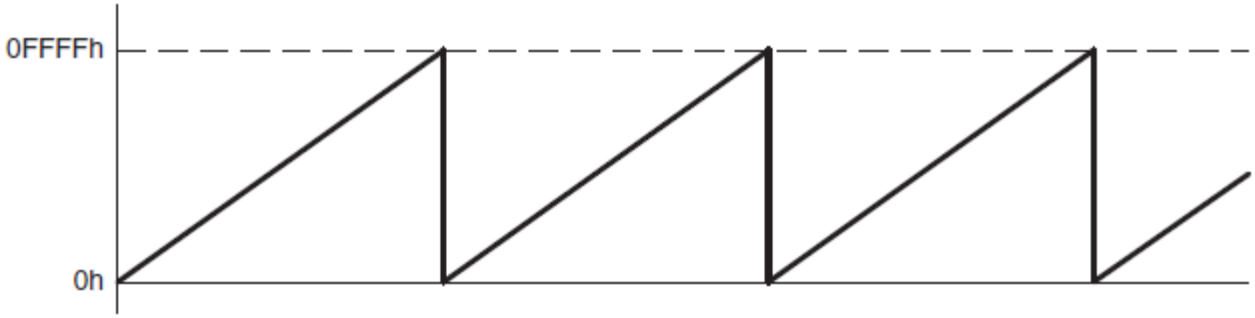
- The TAxCCR0 CCIFG interrupt flag is set when the timer counts to the TAxCCR0 value. The TAIFG interrupt flag is set when the timer counts from TAxCCR0 to zero.

Changing Period Register TAxCCR0

- When changing TAxCCR0 while the timer is running, if the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.

Continuous Mode

- In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero. The capture/compare register TAxCCR0 works the same way as the other capture/compare registers. The TAIFG interrupt flag is set when the timer counts from 0FFFFh to zero.

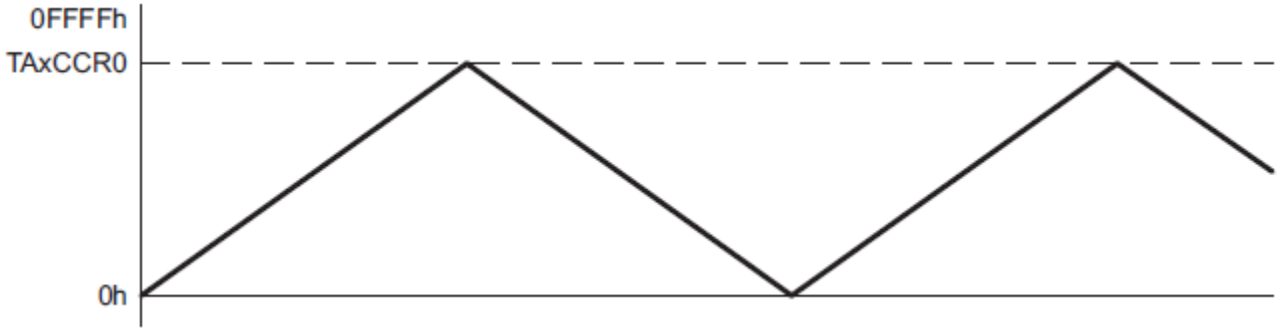


Use of Continuous Mode

The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TAxCCRn register in the interrupt service routine. In this usage, the time interval is controlled by hardware, not software, without impact from interrupt latency. Up to n (where n = 0 to 6), independent time intervals or output frequencies can be generated using capture/compare registers. Refer to the datasheet for the number of capture / compare registers available.

Up/Down Mode

The up/down mode is used if the timer period must be different from 0FFFFh counts, and if symmetrical pulse generation is needed. The timer repeatedly counts up to the value of compare register TAxCCR0 and back down to zero. The period is twice the value in TAxCCR0..



Timer_A Interrupts

- Two interrupt vectors are associated with the 16-bit Timer_A module:
  - TAxCCR0 interrupt vector for TAxCCR0 CCIFG
  - TAxIV interrupt vector for all other CCIFG flags and TAIFG
- In capture mode, any CCIFG flag is set when a timer value is captured in the associated TAxCCRn register.
- In compare mode, any CCIFG flag is set if TAxR counts to the associated TAxCCRn value. Software may also set or clear any CCIFG flag. All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set

TAxCCR0 Interrupt

- The TAxCCR0 CCIFG flag has the highest Timer_A interrupt priority and has a dedicated interrupt vector. The TAxCCR0 CCIFG flag is automatically reset when the TAxCCR0 interrupt request is serviced.

TAxIV, Interrupt Vector Generator

- The TAxCCRy CCIFG flags and TAIFG flags are prioritized and combined to source a single interrupt vector.
- The interrupt vector register TAxIV is used to determine which flag requested an interrupt. The highest-priority enabled interrupt generates a number in the TAxIV register.
- This number can be evaluated or added to the program counter to automatically enter the appropriate software routine.

- Disabled Timer_A interrupts do not affect the TAxIV value.

TAxIV, Interrupt Vector Generator

- Any access, read or write, of the TAxIV register automatically resets the highest-pending interrupt flag.
- If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

- For example, if the TAxCCR1 and TAxCCR2 CCIFG flags are set when the interrupt service routine accesses the TAxIV register, TAxCCR1 CCIFG is reset automatically. After the RETI instruction of the interrupt service routine is executed, the TAxCCR2 CCIFG flag generates another interrupt.

Timer Mode Demonstration

- Use Timer A0 to generate an interrupt every 5msec based on the 8 MHz system clock
- In the TA0 ISR
  - Update a 5 millisecond counter
  - Every 500 ms
    - Update a second counter
    - Toggle the LCD Back Light

Example – Setting Up Timer Mode

```
//-------------------------------------------------------------------------
// Timer A0 initialization sets up both A0_0 and A0_1-A0_2

void Init_Timer_A0(void) {
  TA0CTL = TASSEL__SMCLK;          // SMCLK source
  TA0CTL |= TACLR;                 // Resets TA0R, clock divider, count direction
  TA0CTL |= MC__CONTINOUS;         // Continuous up
  TA0CTL |= ID__2;                 // Divide clock by 2
  TA0CTL &= ~TAIE;                 // Disable Overflow Interrupt
  TA0CTL &= ~TAIFG;                // Clear Overflow Interrupt flag

  TA0EX0 = TAIDEX_7;               // Divide clock by an additional 8

  TA0CCR0 = TA0CCR0_INTERVAL;      // CCR0
  TA0CCTL0 |= CCIE;                // CCR0 enable interrupt

//  TA0CCR1 = TA0CCR1_INTERVAL;    // CCR1
//  TA0CCTL1 |= CCIE;              // CCR1 enable interrupt

//  TA0CCR2 = TA0CCR2_INTERVAL;    // CCR2
//  TA0CCTL2 |= CCIE;              // CCR2 enable interrupt
}
//-------------------------------------------------------------------------
```

Example – Setting Up Timer Mode

```
//-------------------------------------------------------------------------
// TimerA0 0 Interrupt handler
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer0_A0_ISR(void){
...... Add What you need happen in the interrupt ......
  TA0CCR0 += TA0CCR0_INTERVAL;          // Add Offset to TACCR0
}
//-------------------------------------------------------------------------
// TimerA0 1-2, Overflow Interrupt Vector (TAIV) handler
#pragma vector=TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR(void){
  switch(__even_in_range(TA0IV,14)){
    case  0: break;                     // No interrupt
    case  2:                            // CCR1 not used
...... Add What you need happen in the interrupt ......
      TA0CCR1 += TA0CCR1_INTERVAL;      // Add Offset to TACCR1
      break;
    case  4:                            // CCR2 not used
...... Add What you need happen in the interrupt ......
      TA0CCR2 += TA0CCR2_INTERVAL;      // Add Offset to TACCR2
      break;
    case 14:                            // overflow
...... Add What you need happen in the interrupt ......
      break;
    default: break;
  }
}
//-------------------------------------------------------------------------
```

Example – Setting Up Timer Mode

INTERVAL = Clock / Input Divider [ID] / Input Divider Expansion [T?xEX0] / (1 / Desired Time)

Clock = 8,000,000 [MCLK, ACLK, SMCLK]

Input Divider [ID] = 1,2,4,8 [this example 4]

Input Divider Expansion [T?xEX0] = 1,2,3,4,5,6,7,8 [this example 8]

Desired Time Between Interrupts =

= 8,000,000 / 4 / 8 / (1 / 5msec)

= 8,000,000 / 4 / 8 / 200

= 8,000,000 / 4 / 8 / 200

= 2,000,000 / 8 / 200

= 250,000 / 200

= 1,250

Example for Timer A0 Capture Compare Register 0 for desired interrupt every 5msec.

```
//#define TA0CCR0_INTERVAL    (1250) // 8,000,000 / 4 / 8 / (1 / 5msec)
```