Timers

- MSP430F2272 has two 16-bit timers
  - So can count to 0xFFFF
- Independent of the CPU
- Three capture/compare registers
  - TACCR0, TACCR1 & TACCR2 can generate interrupts when timer meets certain conditions (many modes/highly configurable)

<table>
<thead>
<tr>
<th>Register</th>
<th>Short Form</th>
<th>Register Type</th>
<th>Address</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer_A control</td>
<td>TACTL</td>
<td>Read/write</td>
<td>0160h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A counter</td>
<td>TAR</td>
<td>Read/write</td>
<td>0170h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare control 0</td>
<td>TACCTL0</td>
<td>Read/write</td>
<td>0162h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare 0</td>
<td>TACCR0</td>
<td>Read/write</td>
<td>0172h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare control 1</td>
<td>TACCTL1</td>
<td>Read/write</td>
<td>0164h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare 1</td>
<td>TACCR1</td>
<td>Read/write</td>
<td>0174h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare control 2</td>
<td>TACCTL2†</td>
<td>Read/write</td>
<td>0166h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare 2</td>
<td>TACCR2†</td>
<td>Read/write</td>
<td>0176h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A interrupt vector</td>
<td>TAIV</td>
<td>Read only</td>
<td>012Eh</td>
<td>Reset with POR</td>
</tr>
</tbody>
</table>
Timers

• 16-bit counter register TAR (and TBR)
  • Increments/decrements with each rising edge of the clock signal

• Four operating modes
  • Stop (not very interesting)
  • Up

The timer counts UP from zero to TACCR0 then resets to 0 and starts over.
Timers & Interrupts

- Four operating modes
  - Stop (not very interesting)
  - Up

The TACCR0 interrupt flag CCIFG is set when the timer count reaches the value in TACCR0.

The Timer_A interrupt flag TAIFG is set when the timer count overflows the value in TACCR0 and gets back to 0.
# Timer Configuration

<table>
<thead>
<tr>
<th>Register</th>
<th>Short Form</th>
<th>Register Type</th>
<th>Address</th>
<th>Initial State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer_A control</td>
<td>TACTL</td>
<td>Read/write</td>
<td>0160h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A counter</td>
<td>TAR</td>
<td>Read/write</td>
<td>0170h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare control 0</td>
<td>TACCTL0</td>
<td>Read/write</td>
<td>0162h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare 0</td>
<td>TACCR0</td>
<td>Read/write</td>
<td>0172h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare control 1</td>
<td>TACCTL1</td>
<td>Read/write</td>
<td>0164h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare 1</td>
<td>TACCR1</td>
<td>Read/write</td>
<td>0174h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare control 2</td>
<td>TACCTL2†</td>
<td>Read/write</td>
<td>0166h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A capture/compare 2</td>
<td>TACCR2†</td>
<td>Read/write</td>
<td>0176h</td>
<td>Reset with POR</td>
</tr>
<tr>
<td>Timer_A interrupt vector</td>
<td>TAIV</td>
<td>Read only</td>
<td>012Eh</td>
<td>Reset with POR</td>
</tr>
</tbody>
</table>

- **TACTL**—Timer_A Control Register  
  - Used to configure how the timer runs
- **TAR**—Timer_A Register  
  - Holds the current count of the timer
- **TACCTLx**—Timer_A Capture/Compare Control Register  
  - Used to configure the three capture/compare modules
- **TACCRx**—Timer_A Capture/Compare Register  
  - Holds the capture/compare values for the three capture/compare modules
- **TAIV**—Timer_A Interrupt Vector Register  
  - Used to decode non-TACCR0 interrupts
Timers—TACTL Timer_A Control Register

We must tell Timer_A what clock signal to use. For example, if we want the default Subsystem Clock (SMCLK), set bits 9-8 = %10 = TASSEL_2 (defined in header file).

We can slow down the counter.

Be sure to specify counting mode!
## 12.3.1 TACTL, Timer_A Control Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-10</td>
<td>Unused</td>
</tr>
<tr>
<td>9-8</td>
<td>TASSELx Timer_A clock source select</td>
</tr>
<tr>
<td>7-6</td>
<td>IDx Input divider</td>
</tr>
<tr>
<td>5-4</td>
<td>MCx Mode control</td>
</tr>
<tr>
<td>3</td>
<td>TACLR Timer_A clear</td>
</tr>
<tr>
<td>2</td>
<td>TAIE Timer_A interrupt enable</td>
</tr>
<tr>
<td>1</td>
<td>TAIFG Timer_A interrupt flag</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>TBD</td>
</tr>
<tr>
<td>14</td>
<td>TBD</td>
</tr>
<tr>
<td>13</td>
<td>Unused</td>
</tr>
<tr>
<td>12</td>
<td>TACLK</td>
</tr>
<tr>
<td>11</td>
<td>ACLK</td>
</tr>
<tr>
<td>10</td>
<td>SMCLK</td>
</tr>
<tr>
<td>9</td>
<td>INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)</td>
</tr>
<tr>
<td>8</td>
<td>/1</td>
</tr>
<tr>
<td>7</td>
<td>/2</td>
</tr>
<tr>
<td>6</td>
<td>/4</td>
</tr>
<tr>
<td>5</td>
<td>/8</td>
</tr>
<tr>
<td>4</td>
<td>Stop mode: the timer is halted</td>
</tr>
<tr>
<td>3</td>
<td>Up mode: the timer counts up to TACCR0</td>
</tr>
<tr>
<td>2</td>
<td>Continuous mode: the timer counts up to 0FFFFh</td>
</tr>
<tr>
<td>1</td>
<td>Up/down mode: the timer counts up to TACCR0 then down to 0000h</td>
</tr>
<tr>
<td>0</td>
<td>Unused</td>
</tr>
<tr>
<td>11</td>
<td>Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero</td>
</tr>
</tbody>
</table>

### Unused Bits
- Bits 15-10: Unused
- Bits 11-8: Unused

### MCx Mode control
- Bits 7-6: Input divider. These bits select the divider for the input clock.
  - 00: /1
  - 01: /2
  - 10: /4
  - 11: /8

### TAIE Timer_A interrupt enable
- Bit 1: This bit enables the TAIFG interrupt request.
  - 0: Interrupt disabled
  - 1: Interrupt enabled

### TAIFG Timer_A interrupt flag
- Bit 0: No interrupt pending
  - 0: No interrupt pending
  - 1: Interrupt pending
Timers—TACCTLx Cap/Comp Control Register

These bits let you tailor the output characteristics.

This bit enables the interrupt.

This bit tells you if the interrupt has occurred.
### 12.3.4 TACCTLx, Capture/Compare Control Register

| Bit 15-14 | CMx | Bit 13-12 | CCISx | Bit 11 | SCS | Bit 10 | SCCI | Bit 9 | Unused | Bit 8 | CAP | Bit 7-5 | OUTMODx | Bit 4 | CCIE | Bit 3 | CCI | Bit 2 | OUT | Bit 1 | COV | Bit 0 | CCF | |  
|-----------|-----|-----------|-------|--------|-----|--------|------|-------|--------|------|-----|--------|---------|------|------|------|-----|------|-----|--------|-----|-----|------|-----|-----|       |
|           |     |           |       |        |     |        |      |       |        |     |    |        |         |     |      |     |      |      |     |       |     |     |      |     |     |       |
|           |     |           |       |        |     |        |      |       |        |     |    |        |         |     |      |     |      |      |     |       |     |     |      |     |     |       |
|           |     |           |       |        |     |        |      |       |        |     |    |        |         |     |      |     |      |      |     |       |     |     |      |     |     |       |

**CMx** Bit 15-14: Capture mode
- 00: No capture
- 01: Capture on rising edge
- 10: Capture on falling edge
- 11: Capture on both rising and falling edges

**CCISx** Bit 13-12: Capture/compare input select. These bits select the TACCRx input signal. See the device-specific data sheet for specific signal connections.
- 00: CCIxA
- 01: CCIxB
- 10: GND
- 11: VCC

**SCS** Bit 11: Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.
- 0: Asynchronous capture
- 1: Synchronous capture

**SCCI** Bit 10: Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit.

**Unused** Bit 9: Unused. Read only. Always read as 0.

**CAP** Bit 8: Capture mode
- 0: Compare mode
- 1: Capture mode

**OUTMODx** Bits 7-5: Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0, because EQUx = EQU0.
- 000: OUT bit value
- 001: Set
- 010: Toggle/reset
- 011: Set/reset
- 100: Toggle
- 101: Reset
- 110: Toggle/set
- 111: Reset/set

**CCIE** Bit 4: Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.
- 0: Interrupt disabled
- 1: Interrupt enabled

**CCI** Bit 3: Capture/compare input. The selected input signal can be read by this bit.

**OUT** Bit 2: Output. For output mode 0, this bit directly controls the state of the output.
- 0: Output low
- 1: Output high

**COV** Bit 1: Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.
- 0: No capture overflow occurred
- 1: Capture overflow occurred

**CCIFG** Bit 0: Capture/compare interrupt flag
- 0: No interrupt pending
- 1: Interrupt pending
Timers—TACCTLx Cap/Comp Control Register

- **Two ways to go**
  - Timer + GPIO
    - Set/reset a GPIO pin inside timer ISR
    - Controlled by software, so requires extra CPU cycles
  - Timer output
    - Use the timer output directly
    - Can be totally controlled by hardware alone

This is what you want to do for the lab!

For example: You could use Timer_A in Up mode with output mode 7 and use the timer output signal on pin TA1 to control an LED.
Timers—TACCTLx Cap/Comp Control Register

- Variable width pulses

Use TACCR0 and TACCR1 control PWM period and duty cycle.
Fourier Representation of Signals

• Any periodic waveform can be represented as an (infinite) sum of sine and cosine waveforms.
• Used to determine the frequency content (spectrum) of signals

\[ y(t) = A_0 + \sum_{n=1}^{\infty} \left( A_n \cos(n\omega t) + B_n \sin(n\omega t) \right) \]

\[ A_n = \frac{1}{T} \int_0^T y(t) \cos(n\omega t) \, dt \quad B_n = \frac{1}{T} \int_0^T y(t) \sin(n\omega t) \, dt \]

\( T = \frac{2\pi}{f} \) is the period of the signal; \( f \) is the 1st harmonic, \( 2f \) is the 2nd harmonic, etc.
Frequency Domain Representation

- **Square Wave**

\[
y(t) = A_0 + \sum_{n=1}^{\infty} \left( A_n \cos(n\omega t) + B_n \sin(n\omega t) \right)
\]

\[
A_0 = \frac{1}{T} \int_0^T y(t) dt
\]
Frequency Domain Representation

- Square Wave

- DC Component ($A_0$)
- 1st harmonic (fundamental)
- 3rd harmonic

Frequency Spectrum
RC Low Pass Filter

\[ f_c = \frac{1}{2\pi RC} \]

\[ \frac{V_o}{V_i} = \frac{1}{\sqrt{1 + (f/f_c)^2}} \]

\[ \phi = -\tan^{-1}(f/f_c) \]