Administrivia

- Traveling next week.
- Schedule change for the next exam due to technical open house.
  - New date will be April 1.
  - No labs will be held during the week of March 24.
  - Lab 9 will be due the week of March 31.
Basic Principles of Output Compare

- **Output compare** can create square waves, generate pulses, implement time delays, and execute periodic interrupts.
- Can also use with input capture to measure frequency.
- Each output capture module has:
  - An external output pin, OCn
  - A flag bit
  - A force control bit FOCn
  - Two control bits, OMn, OLn
  - An interrupt mask bit (arm)
  - A 16-bit output compare register

Basic Principles of Output Compare (cont)

- Output compare pin can control an external device.
- Output compare event occurs and sets flag when either:
  - The 16-bit TCNT matches the 16-bit OC register
  - The software writes a 1 to the FOC bit.
- OMn, OLn bits specify effect of event on the output pin.
- Two or three actions result from a compare event:
  - The OCn output bit changes
  - The output compare flag is set.
  - An interrupt is requested if the mask is 1.
Control Bits and Flags

- Output compares are on port T (i.e., PTT).
- Set pin to output compare mode by setting bit to 1 in TIOS.
- Output compare registers are TC0, ..., TC7.
- Arm interrupts using TIE.
- Flags are found in TFLG1.
- Set effect of trigger using TCTL1 and TCTL2.
- Can force an output compare by setting bit to 1 in CFORC.

<table>
<thead>
<tr>
<th>OMn</th>
<th>OLn</th>
<th>Effect of when TOCn=TCNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Does not affect OCn</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Toggle OCn</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OCn=0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Set OCn=1</td>
</tr>
</tbody>
</table>

Applications of Output Compare

- Can create a fixed time delay.
  - Read the current 16-bit TCNT
  - Calculate TCNT+fixed
  - Set 16-bit output compare register to TCNT+fixed
  - Clear the output compare flag
  - Wait for the output compare flag to be set
- Delay of steps 1 to 4 sets the minimum delay.
- Maximum delay is 65,536 cycles.
# Periodic Interrupt Using Output Capture

```c
#define PERIOD 1000

unsigned short Timer;

void OC6_Init(void)
{
    asm sei // Make atomic
    TSCR1 = 0x80;
    TSCR2 = 0x02; // 1 MHz TCNT
    TIOS |= 0x40; // activate OC6
    TIE |= 0x40; // arm OC6
    TC6 = TCNT+50; // first in 50us
    Time = 0; // Initialize
    asm cli } // enable IRQ

void interrupt 14 OC6handler(void){
    TC6 = TC6+PERIOD; // next in 1 ms
    TFLG1 = 0x40; // acknowledge C6F
    Time++; }
```

---

## Pulse-Width Modulation

![Pulse-Width Modulation Diagram](image)
Pulse-Width Modulated Square-Wave

unsigned short High;  // Cycles High
unsigned short Low;   // Cycles Low
void Init(void){
    asm sei                // make atomic
    TIOS |= 0x08;         // enable OC3
    DDRT |= 0x08;         // PT3 is output
    TSCR1 = 0x80;         // enable
    TSCR2 = 0x01;         // 500 ns clock
    TIE |= 0x08;          // Arm output compare 3
    TFLG1 = 0x08;         // Initially clear C3F
    TCTL2 = (TCTL2&0x3F)|0x40; // toggle
    TC3 = TCNT+50;        // first right away
    asm cli
}

void interrupt 11 TC3handler (void){
    TFLG1 = 0x08;        // ack C3F
    if(PTT&0x08){        // PT3 is now high
        TC3 = TC3+High;  // 1 for High cyc
    }
    else{                // PT3 is now low
        TC3 = TC3+Low;   // 0 for Low cycles
    }
}
void main(void){
    High=8000; Low=2000;
    Init();
    while(1);
}
Pulse-Width Modulation Overhead

<table>
<thead>
<tr>
<th>Component</th>
<th>6812</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process the interrupt (cycles)</td>
<td>9</td>
</tr>
<tr>
<td>Execute the handler (cycles)</td>
<td>27-28</td>
</tr>
<tr>
<td>Total time T (cycles)</td>
<td>36-37</td>
</tr>
</tbody>
</table>

Frequency Measurement

- Direct measurement of frequency involves counting input pulses for a fixed amount of time.
- Can use input capture to count pulses, and output capture to create a fixed time interval.
- Input Capture handler increments Counter.
- Output compare handler calculates frequency:
  \[
  f = \frac{\text{Counter}}{\text{fixed time}}
  \]
- The frequency resolution is:
  \[
  f = \frac{1}{\text{fixed time}}
  \]
#define Rate 20000  // 10 ms
void Init(void) {
    asm sei    // make atomic
    TIOS |= 0x20; // enable OC5
    TSCR1 = 0x80; // enable
    TSCR2 = 0x01; // 500 ns clock
    TIE |= 0x22; // Arm OC5 and IC1
    TC5 = TCNT+Rate; // First in 10 ms
    TC54 = (TC54&0xF3)|0x04; /* C1F set on rising edges */
    Count = 0; // Set up for first
    Done = 0; // Set on measurements
    TFILG1 = 0x22; // clear C5F, C1F
    asm cli
}

Frequency Measurement (cont)

```c
void interrupt 9 TC1handler(void){
    Count++;    // number of rising edges
    TFLG1 = 0x02; // ack, clear C1F
}
void interrupt 13 TC5handler(void){
    TFLG1 = 0x20;    // Acknowledge
    TC5 = TC5+Rate;    // every 10 ms
    Freq = Count;    // 100 Hz units
    Done = 0xff;
    Count = 0;    // Setup for next
}
```

Conversion Between Frequency and Period

- Could measure frequency from period measurement:
  \[ f = \frac{1}{p} \]

- If range of period measurement is 36µs to 32ms with resolution of 500ns, frequency range is 31 to 27,778Hz.
  \[ f = \frac{1}{p} \cdot \frac{1}{500\text{ns}} = \frac{200000}{p} \]

- Resolution relationship is not as obvious:
  \[ \Delta f = \frac{1}{(1/f) - \Delta p} - f = \frac{1}{(1/f) - 500\text{ns}} - f \]
### Relationship Between Frequency and Period

<table>
<thead>
<tr>
<th>Frequency (Hz)</th>
<th>Period (µs)</th>
<th>Δf (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31,250</td>
<td>32</td>
<td>500</td>
</tr>
<tr>
<td>20,000</td>
<td>50</td>
<td>200</td>
</tr>
<tr>
<td>10,000</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>5,000</td>
<td>200</td>
<td>13</td>
</tr>
<tr>
<td>2,000</td>
<td>500</td>
<td>2</td>
</tr>
<tr>
<td>1,000</td>
<td>1,000</td>
<td>0.5</td>
</tr>
<tr>
<td>500</td>
<td>2,000</td>
<td>0.13</td>
</tr>
<tr>
<td>200</td>
<td>5,000</td>
<td>0.02</td>
</tr>
<tr>
<td>100</td>
<td>10,000</td>
<td>0.005</td>
</tr>
<tr>
<td>50</td>
<td>20,000</td>
<td>0.001</td>
</tr>
<tr>
<td>31.25</td>
<td>32,000</td>
<td>0.0005</td>
</tr>
</tbody>
</table>

### Pulse-Width Modulation (PWM) Configuration

- Dedicated hardware can create PWM signals on port P with no overhead.
- **MODRR** register can connect PWM system to port T pins.
- **PWME** register used to enable PWM channels.
- Either three 16-bit channels or up to six 8-bit channels.
- **CON01** bit connects two 8-bit channels to form one 16-bit channel (similarly for **CON23** and **CON45**).
PWM Configuration

- Output is high number of counts in corresponding **PWMDTY** register, and total counts in a cycle in the corresponding **PWMPER** register.

\[ PPOL_x = 1 \]

\[ PPOL_x = 0 \]

Clock Choice

- Many possible choices for the clock.
- A and B clocks configured by the **PWMPRCLK** register as a divided down version of the E clock between E and E/128.
- SA clock is the A clock divided by two times value in **PWMSCLA** register.
- SB clock is the B clock divided by two times value in **PWMSCLB** register.
- Channels 0, 1, 4, and 5 can use either A or SA clock while channels 2 and 3 use either the B or SB clock.
8-bit PWM Output (10ms Period)

void PWM_Init(void){
    MODRR |= 0x01;    // PT0 with PWM
    PWME |= 0x01;     // enable channel 0
    PWMPOL |= 0x01;   // PT0 high then low
    PWMCLK |= 0x01;   // Clock SA
    PWMPRCLK = (PWMPRCLK&0xF8)|0x04; // A=E/16
    PWMSCCLA = 5;    // SA=A/10, 0.25*160=40us
    PWMPER0 = 250;   // 10ms period
    PWMDTY0 = 0;     // initially off
}

// Set the duty cycle on PT0 output
void PWM_Duty0(unsigned char duty){
    PWMDTY0 = duty; // 0 to 250
}

8-bit PWM Output (1s Period)

void PWM_Init(void){
    MODRR |= 0x08;    // PT3 with PWM
    PWME |= 0x08;     // enable channel 3
    PWMPOL |= 0x08;   // PT3 high then low
    PWMCLK &=~0x08;   // Clock B
    PWMCTL |= 0x20;   // Concatenate 2+3
    PWMPRCLK = (PWMPRCLK&0x8F)|0x60; // B=E/64
    PWMPER23 = 62500; // 1s period
    PWMDTY23 = 0;     // initially off
}

// Set the duty cycle on PT3 output
void PWM_Duty(unsiged short duty){
    PWMDTY23 = duty; // 0 to 62500
}