Basic Principles of Output Capture

- **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 Components of Output Compare

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.

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.
Output Compare Interface on 6812

- 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 CFORCE.

<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>

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 CFORCE.

Output Compare 7

- When TC7=TCNT, can set or clear any output compare pins.
- OC7M register selects pins to be affected by OC7 event.
- OC7D register specifies value of pin after OC7 event.

Periodic Interrupt Using Output Capture

```c
#define PERIOD 1000
unsigned short Time;
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 } // enableIRQ
void interrupt 14 OC6handler(void){
    TC6 = TC6+PERIOD; // next in 1 ms
    TFLG1 = 0x40; // acknowledge C6F
    Time++; }
```

Square-Wave Generation

```c
unsigned short Period; // in usec
void ritual(void) {
    asm sei  // make atomic
    TIOS = 0x80; // enable OC3
    DDRT = 0x08; // PT3 is output
    TSCR1 = 0x80; // enable
    TSCR2 = 0x01; // 500 ns clock
    TCTL2 = (TCTL2&0x3F)|0x40; // toggle
    TIE = 0x00; // Arm output compare 3
    TFLG1 = 0x08; // Initially clear C3F
    TC3 = TCNT+50; // First one in 25 us
    asm cli } // enable IRQ
void interrupt 11 TC3handler(void){
    TFLG1 = 0x08; // ack C3F
    TC3 = TC3+Period; // calculate Next
```
Square-Wave Generation Overhead

<table>
<thead>
<tr>
<th>Component</th>
<th>6812</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process the interrupt (cycles, µs)</td>
<td>9 = 2.25 µs</td>
</tr>
<tr>
<td>Execute the entire handler (cycles, µs)</td>
<td>20 = 5 µs</td>
</tr>
<tr>
<td>Total time (µs)</td>
<td>7.25 µs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Freq.</th>
<th>Period</th>
<th>Interrupt every (cycles)</th>
<th>Time to process (cycles)</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Hz</td>
<td>100 ms</td>
<td>100,000</td>
<td>29</td>
<td>0.03</td>
</tr>
<tr>
<td>100 Hz</td>
<td>10 ms</td>
<td>10,000</td>
<td>29</td>
<td>0.3</td>
</tr>
<tr>
<td>1 kHz</td>
<td>1 ms</td>
<td>1000</td>
<td>29</td>
<td>3</td>
</tr>
<tr>
<td>5 kHz</td>
<td>200 µs</td>
<td>200</td>
<td>29</td>
<td>14.5</td>
</tr>
<tr>
<td>1/P</td>
<td>P (µs)</td>
<td>P</td>
<td>29</td>
<td>2900/P</td>
</tr>
</tbody>
</table>

Pulse-Width Modulation

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; |
| ritual(); |
| while(1); |

Delayed Pulse Generation

<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>
Delayed Pulse Generation

```c
void Pulse(unsigned short Delay,
           unsigned short Width){
    asm sei // make atomic
    TIOS |= 0x08; // enable OC3
    DDRT |= 0x08; // PT3 is output
    TSCR1 = 0x80; // enable
    TSCR2 = 0x01; // 500 ns clock
    TC7 = TCNT+Delay;
    TC3 = TC7+Width;
    OC7M = 0x08; // connect OC7 to PT3
    OC7D = 0x08; // PT3=1 when TC7=TCNT
    TCTL2=(TCTL2&0x3F)|0x80; // PT3=0 when TC3=TCNT
    TFLG1 = 0x08; // Clear C3F
    TIE |= 0x08; // Arm C3F
    asm cli
}
```

Delayed Pulse Generation (cont)

```c
void interrupt 11 TC3handler(void){
    OC7M = 0; // disconnect OC7 from PT3
    OC7D = 0;
    TCTL2 &=~0xC0; // disable OC3
    TIE &= ~0x08; // disarm C3F
}
```

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:

  \[
  f = \frac{1}{\text{fixed time}}
  \]

Frequency Measurement (cont)

```c
#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
    TCTL4 = (TCTL4&0xF3)|0x04; /* C1F set on rising edges */
    Count = 0; // Set up for first
    Done = 0; // Set on measurements
    TFLG1 = 0x02; // ack, clear C1F
    asm cli
}
```

```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 36\(\mu\)s to 32ms with resolution of 500ns, frequency range is 31 to 27,778Hz.

\[ f = \frac{1}{p} \cdot \frac{1}{500\,\text{ns}} = \frac{2000000}{p} \]

Resolution relationship is not as obvious:

\[ \Delta f = \frac{1}{(1/f) - \Delta p} \frac{1}{(1/f) - 500\,\text{ns}} - f \]

**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>

**Period Measurement with Δp = 1ms**

- Each rising edge generates input capture interrupt.
- Output compare used to increment a software counter, Time, every 1 ms.
- Period is number of 1-ms output compare interrupts between one rising edge to the next rising edge.
- Range is 0 to 65s determined by the 16-bit size of Time.

```
#define RESOLUTION 2000

void Ritual(void){
    asm sei // make atomic
    TIOS |= 0x08; // enable OC3
    TSCR1 = 0x80; // enable
    TSCR2 = 0x01; // 500 ns clock
    TFLG1 = 0x0A; // Clear C3F, CIF
    TIE = 0x0A; // Arm OC3 and IC1
    TCTL4 = (TCTL4&0xF3)|0x04; /* C1F set on rising edges */
    while((TFLG1&0x02)==0); // wait rising
    TFLG1 = 0x02; // Clear C1F
    TC3 = TCNT+RESOLUTION;
    Cnt=0; OverFlow=0; Done=0;
    asm cli
}
```

**Frequency Measurement with Δf = 0.1Hz**

- If count pulses in 10-s time interval, then number of pulses is frequency with units of 1/10s or 0.1 Hz.
- Setting output compare to interrupt every 25 ms, means that 400 interrupts creates a 10-s time delay.
- Number of input capture interrupts during this interval is the input frequency in units of 0.1 Hz.
**Basic Timing Involved in Frequency Measurement**

![Diagram of timing intervals]

**Frequency Measurement with $\Delta f = 0.1\text{Hz}$**

```c
#define PERIOD 50000 // 25 ms
void Init(void) {
    asm sei // make atomic
    TIOS| = 0x20; // enable OC5
    TSCR1 = 0x80; // enable
    TSCR2 = 0x02; // 500 ns clock
    TIE = 0x22; // Arm OC5 and IC1
    TCTL4 = (TCTL4&0xF3)|0x04; // rising
    Count = 0; // Set up for first
    Done = 0; // Set on measurement
    FourHundred = 0;
    TC5 = TCNT+PERIOD; // First in 25 ms
    TFLG1 = 0x22; // Clear C5F,C1F
    asm cli
}
```

**Frequency Measurement with $\Delta f = 0.1\text{Hz}$ (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+PERIOD; // every 25 ms
    if (++FourHundred==400){
        Freq = Count; // 0.1 Hz units
        FourHundred = 0;
        Done = 0xff;
        Count = 0; // Setup for next
    }
}
```

**Pulse Accumulator**

- Pulse accumulator is enabled when $\text{PAEN}$ is 1.
- 16-bit event counter ($\text{PACNT}$).

<table>
<thead>
<tr>
<th>PAMOD</th>
<th>PEDGE</th>
<th>Mode</th>
<th>Counts when</th>
<th>Sets $\text{PAIF}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Event counting</td>
<td>PT7 falls</td>
<td>PT7 falls</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Event counting</td>
<td>PT7 rises</td>
<td>PT7 rises</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Gated time</td>
<td>PT7=1</td>
<td>PT7 falls</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Gated time</td>
<td>PT7=0</td>
<td>PT7 rises</td>
</tr>
</tbody>
</table>

- In *gated time accumulation* mode, counter incremented by a free-running clock (E clock divided by 64).
- Interrupt occurs when $\text{PAIF}$ is set if $\text{PAII}$ is set.
- $\text{PAOVF}$ bit is set when $\text{PACNT}$ overflows.
- If $\text{PAOV}$ is set, interrupt occurs on overflow.

**Frequency Measurement Using Pulse Accumulator**

```c
;returns Reg D = freq in Hz
Freq bclr DDRT,#$80 ;PT7 is input
    movb #$40,PACTL ;count falling
    movw #0,PACNT
    movb #$02,PAFLG ;clear PAOVF
    ldy #100
    br Timer_Wait10ms ;Program 2.6
bad ldd #65535 ;too big
    bra out
ok ldd PACNT ;units in Hz
    out rts
```

**Pulse-Width Measurement Using Pulse Accumulator**

```c
;returns Reg D = pulse width in 16us
Puls bclr DDRT,#$80 ;PT7 is input
    movb #$60,PACTL ;measure high
    movw #0,PACNT
    movb #$02,PAFLG ;clear PAOVF
    loop bclr PAFLG,#$01,loop
    bclr PAFLG,#$02,ok ;check PAOVF
bad ldd #65535 ;too big
    bra out
ok ldd PACNT ;units in 16us
    out rts
```
Pulse-Width Modulation on the MC9S12C32

- 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).

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

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.

void PWM_Init(void){
MODRR |= 0x01; // PT0 with PWM
PWME |= 0x01; // enable channel 0
PWMPOL |= 0x01; // PT0 high then low
PWMPCLK |= 0x01; // Clock SA
PWMPRCLK = (PWMPRCLK|0xF8)|0x04; // A=E/16
PWMSCLA = 5; // SA=A/10, 0.25*160=40us
PWMPER0 = 250; // 10ms period
PWMDTY0 = 0; // initially off
}
void PWM_Duty0(unsigned char duty){
PWMDTY0 = duty; // 0 to 250
}

void PWM_Init(void){
MODRR |= 0x08; // PT3 with PWM
PWME |= 0x08; // enable channel 0
PWMPOL |= 0x08; // PT3 high then low
PWMPCLK &=~0x08; // Clock B
PWMPCTL |= 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(unsigned short duty){
PWMDTY23 = duty; // 0 to 62500
}