**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**

<table>
<thead>
<tr>
<th>small</th>
<th>middle</th>
<th>large</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCNT</th>
<th>OC reg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mask</td>
</tr>
</tbody>
</table>

**Basic Principles of Output Compare (cont)**

- Output compare pin can control an external device.
- Output compare event occurs and sets flag when either:
  1. The 16-bit TCNT matches the 16-bit OC register
  2. 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:
  1. The OCn output bit changes
  2. The output compare flag is set.
  3. An interrupt is requested if the mask is 1.
Applications of Output Compare

- Can create a **fixed** time delay.
  1. Read the current 16-bit TCNT
  2. Calculate TCNT+fixed
  3. Set 16-bit output compare register to TCNT+fixed
  4. Clear the output compare flag
  5. 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.

Control Bits and Flags

<table>
<thead>
<tr>
<th>OMn</th>
<th>OLn</th>
<th>Effect 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>

Output Compare Interface on 68HC11

| TCNT | Set 16-bit counter incremented every 4 clock cycles |
| T0C1 |          |
| T0C2 | OC1F set when TOC2=TCNT |
| T0C3 | OC3F set when TOC3=TCNT |
| T0C4 | OC4F set when TOC4=TCNT |
| T0C5 | OC5F set when TOC5=TCNT |

Setting the TFLG1 Register

- Care again must be taken when clearing TFLG1.
- The following works:
  ```
  ldaa #$20
  TFLG1 = 0x20;
  staa $1023
  ```
- The following does not:
  ```
  ldax #$1000
  TFLG1 | 0x20;
  bset $23,X,$01
  ```
Control Bits and Flags

Control Bits and Flags

<table>
<thead>
<tr>
<th>OC1M</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PA7</td>
<td>PA6</td>
<td>PA5</td>
<td>PA4</td>
<td>PA3</td>
<td>OC1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OC1M</td>
<td>OC1M</td>
<td>OC1M</td>
<td>OC1M</td>
<td>OC1M</td>
<td>OC1M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OC1D</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PA7</td>
<td>PA6</td>
<td>PA5</td>
<td>PA4</td>
<td>PA3</td>
<td>OC1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OC1D</td>
<td>OC1D</td>
<td>OC1D</td>
<td>OC1D</td>
<td>OC1D</td>
<td>OC1D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\text{Slide 9}$

Init for Periodic Interrupt Using Output Compare

```
TIME rmb 2 ; inc every 1 ms
Init sei ; make atomic
1daa TMSK1 ; Old value
oraa #$08 ; TMSK1 OC5I = 1
staa TMSK1 ; Arm OC5F
1dd #0
std TIME ; initialize
1daa #$08 ; clear OC5F
staa TFLG1
1dd TCNT ; current time
add #2000 ; first in 1 ms
std TOCS
cli ; enable
rts
```

$\text{Slide 11}$

Periodic Interrupt Using Output Capture

```
Component & 6811
---|---
Time with interrupts disabled & T
Longest instruction (cycles, $\mu$s) & 41 = 20.5 $\mu$s
Process the interrupt (cycles, $\mu$s) & 14 = 7 $\mu$s
Execute the handler (cycles, $\mu$s) & 13 = 6.5 $\mu$s
Max latency ($\mu$s) & T + 34 $\mu$s
```

$\text{Slide 10}$

ISR for Periodic Interrupt Using Output Compare

```
OCSHAN ldx TIME [5]
inx [3]
stx TIME [5]
1daa #$08 ; clear OC5F
staa TFLG1 ; Acknowledge
1dd TOCS
add #2000 ; next
std TOCS
rti
org $FFE0
fdb OCSHAN
```

$\text{Slide 12}$
**Periodic Interrupt Using Output Capture**

```c
#define Rate 2000
#define OC5 0x08
unsigned int Time; // Inc every 1ms
#pragma interrupt_handler TOC5handler()
void TOC5handler(void){
    TFLG1=OC5; // Ack interrupt
    TOC5=TOC5+Rate; // Executed every 1 ms
    Time++; }
void ritual(void) {
    asm("sei"); // make atomic
    TMSK1|=OC5; // Arm output compare 5
    Time = 0;
    TFLG1=OC5; // Initially clear OC5F
    TOC5=TCNT+Rate; // First one in 1 ms
    asm(" cli"); }
```

**Initialization for Square-Wave Generation**

```c
Period rmb 2 ;units sec
Init sei ;make atomic
    ldaa TMSK1 ;Old value
    oraa #$20 ;TMSK1 OC3i=1
    staa TMSK1 ;Arm OC3F
    ldaa TCTL1
    anda #$0F ;OM3=0
    oraa #$10 ;GL3=1
    staa TCTL1
    ldaa #$20 ;clear OC3F
    staa TFLG1
    ldd TCNT ;current time
    addd #2000 ;first in 1 ms
    std TOC3
    cli ;enable
    rts
```

**Square-Wave Generation**

<table>
<thead>
<tr>
<th>Freq.</th>
<th>Period (cycles)</th>
<th>Interrupt every</th>
<th>Time-to-process (cycles)</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Hz</td>
<td>50,000</td>
<td>48</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>100 Hz</td>
<td>10</td>
<td>48</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1 kHz</td>
<td>500</td>
<td>48</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5 kHz</td>
<td>200</td>
<td>48</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>1/P</td>
<td>P (μs)</td>
<td>48</td>
<td>4800/P</td>
<td></td>
</tr>
</tbody>
</table>

**ISR for Square-Wave Generation**

```asm
OC3HAN ldaa #$20 ;clear OC3F [2]
staa TFLG1 ;Ack [4]
ldd TOC3 [5]
addd Period ;next [6]
std TOC3 [5]
rti [12]
org $FFE4
fdb OC3HAN
```
Square-Wave Generation in C

```c
unsigned int Period; // Period in usec
#pragma interrupt_handler TOC3handler()
void TOC3handler(void){
    TOC3 = TOC3 + Period; // calculate Next
    TFLG1 = 0x20; // ack, OC3F=0
    void ritual(void){
        asm("sei"); // make atomic
        TFLG1 = 0x20; // clear OC3F
        TMSK1 = 0x20; // arm OC3
        TCTL1 = (TCTL1 & 0xCF) | 0x10;
        TOC3 = TCNT + 50; // first right away
        asm("cli"); }
}
```

Init for Pulse-Width Modulated Square-Wave

```c
High rmb 2 ;number of cycles high
Low rmb 2 ;number of cycles low
RITUAL sei ;make atomic
ldaa TMSK1 ;Old value
oraa #$20 ;TMSK1 OC3I=1
staa TMSK1 ;Arm OC3F
ldaa TCTL1
oraa #$30 ;OM3=1, OL3=1
staa TCTL1
ldaa #$20 ;clear OC3F
staa TFLG1
ldd TCNT ;current time
add #50 ;first in 25s
std TOC3
ci ;enable
rts
```

Pulse-Width Modulation

<table>
<thead>
<tr>
<th>Component</th>
<th>6811</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process the interrupt (cycles)</td>
<td>14</td>
</tr>
<tr>
<td>Execute the handler (cycles)</td>
<td>53-56</td>
</tr>
<tr>
<td>Total time T (cycles)</td>
<td>67-70</td>
</tr>
</tbody>
</table>

ISR for Pulse-Width Modulated Square-Wave

```c
OC3HAN ldaa #$20 ;clear OC3F [2]
staa TFLG1 ;Ack [4]
ldaa TCTL2 ;rise/fall? [4]
bita #$10 [2]
bq zero [3]
one ldd TOC3 [5]
add #High ;OC3 is 1 [6]
std TOC3 [5]
ldaa TCTL2 [4]
bradone [3]
```
**ISR for Pulse-Width Modulated Square-Wave (cont)**

```assembly
zero   ld T0C3              [6]
add Low ;DC is 0           [6]
std T0C3                   [5]
ldaa TCTL2                 [4]
oraa #$40                   [2]
staa TCTL2                 [4]
done rti                   [12]
org $FFE4
fdb OC3HAN
```

**Pulse-Width Modulated Square-Wave (cont)**

```c
// Period is High+Low Cycles
#pragma interrupt_handler TOC3handler()

void TOC3handler(void){
    if(TCTL1&0x10){ // PA5 is now high
        T0C3=T0C3+High; // 1 for High cyc
        TCTL1|=0xEF;} // clear on next
    else { // PA5 is now low
        T0C3=T0C3+Low; // 0 for Low cycles
        TCTL1|=0x10;} // set on next int
    TFLG1=0x20;} // ack, clear OC3F
```

**Pulse-Width Modulated Square-Wave**

```c
unsigned int High; // Num of Cycles High
unsigned int Low; // Num of Cycles Low

void ritual(void){
    asm("sei"); // make atomic
    TFLG1 = 0x20; // initially OC3F=0
    TMSK1|= 0x20; // arm OC3
    TCTL1|= 0x30; // PA5 set on next int
    T0C3 = TCNT+50; // first right away
    asm("cli"); }

void main(void){
      High=8000; Low=2000;
      ritual();
      while(1);}
```

**Delayed Pulse Generation**

```
6811/6812

OC3

Delay Width
```
Delayed Pulse Generation in C

```c
void Pulse(unsigned int Delay,
           unsigned int Width)
{
    asm("sei"); // make atomic
    TOC1=TCNT+Delay;
    TOC3=TOC1+Width;
    OC1M=0x20;   // connect OC1 to PA5/OC3
    OC1D=0x20;   // PA5=1 when TOC1=TCNT
    TCTL1=(TCTL1&0xCF)|0x20; // connect OC1 to PA5/OC3
    TOC1=TCNT+Delay;
    TOC3=TOC1+Width;
    OC1M=0x20;   // disconnect OC1 from PA5
    OC1D=0x20;   // PA5=0 when TOC3=TCNT
    TFLG1 = 0x20; // Clear OC3F
    TMSK1|= 0x20; // Arm OC3F
    asm("cli");
}
```

Delayed Pulse Generation in C (cont)

```c
#pragma interrupt_handler TOC3handler()
void TOC3handler(void)
{
    OC1M=0;   // disconnect OC1 from PA5
    OC1D=0;
    TCTL1|=0x20; // disable OC3
    TMSK1|=0x20; // disarm OC3F
}
```

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 in C

```c
#define IC1F 0x04 // connected here
#define Rate 20000 // 10 ms
#define OC5F 0x08

void ritual(void) {
  asm("sei"); // make atomic
  TMSK1 |= OC5F + IC1F; // Arm OC5 and IC1
  TOC5 = TCNT + Rate; // First in 10 ms
  TCTL2 = (TCTL2 & 0xCF) | 0x10;
  /* IC1F set on rising edges */
  Count = 0; // Setup for first
  Done = 0;
  /* Set on the subsequent measurements */
  TFLG1 = OC5F + IC1F; // clear OC5F, IC1F
  asm(" cli");
}
```

### 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.
  \[ f = \frac{1}{p}, \quad \frac{1}{500\text{ns}} = \frac{2000000}{p} \]

- 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>
Period Measurement with $\Delta_T = 1\text{ms}$

- Each rising edge generates input capture interrupt.
- Output compare is used to increment a software counter, $\text{Time}$, 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 65536 determined by the 16-bit size of $\text{Time}$.

Period Measurement in C

```c
#define resolution 2000

void Ritual(void)
{
    asm("sei"); // make atomic
    TFLG1 = 0x24; // Clear OC3F, IC1F
    TMSK1 = 0x24; // Arm OC3 and IC1
    TCTL2 = 0x10; // rising edges
    while((TFLG1&0x04)==0);
    // wait for first rising
    TFLG1 = 0x04; // Clear IC1F
    T0C3=TCNT+resolution;
    Cnt=0; OverFlow=0; Done=0;
    asm("cli"); }
```

Period Measurement in C (cont)

```c
#include <avr/interrupt.h>

#pragma interrupt_handler TOC3handler()
void TOC3handler(void)
{
    T0C3=T0C3+resolution; // every 1 ms
    TFLG1=0x24; // ack, clear OC3F
    Cnt++;
    if(Cnt==0) OverFlow=0xFF;
}

#pragma interrupt_handler TIC1handler()
void TIC1handler(void)
{
    TFLG1=0x04; // ack, clear IC1F
    if(OverFlow)
    {
        Period=65535;
        OverFlow=0;
    }
    else Period=Cnt;
    Cnt=0;
    Done=0xFF;
}
```

Frequency Measurement with $\Delta_f = 0.1\text{Hz}$

- If count pulses in 10-s time interval, then number of pulses is frequency with units of $1/10\text{s}$ 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.
Frequency Measurement in C

```c
#define IC1F 0x04 // connected here
#define Rate 50000 // 25 ms
#define OC5F 0x08

void ritual(void) {
    asm("sei"); // make atomic
    TMSK1 = OC5F+IC1F; // Arm OC5 and IC1
    TOC5 = TCNT + Rate; // First in 25 ms
    TCTL2 = (TCTL2 & 0xCF) | 0x10;
    /* IC1F set on rising edges */
    Count = 0; // Set up for first
    Done = 0; // Set on subsequent meas
    FourHundred = 0;
    TFLG1 = OC5F+IC1F; // Clear OC5F IC1F
    asm("cli");
}
```

Frequency Measurement in C (cont)

```c
#pragma interrupt_handler TIC1handler()
void TIC1handler(void) {
    Count++; // number of rising edges
    TFLG1 = IC1F; // ack, clear IC1F
}
```

```c
#pragma interrupt_handler TOC5handler()
void TOC5handler(void) {
    TFLG1 = OC5F; // Acknowledge
    TOC5 = TOC5 + Rate; // every 25 ms
    if (++FourHundred == 400) {
        Freq = Count; // 0.1 Hz units
        FourHundred = 0;
        Done = 0xff;
        Count = 0;
    }
}
```

Basic Time Involved in Frequency Measurement