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 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.

Basic Components of Output Compare

(See Figures 6.16 and 6.17)
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.

Output Compare Interface on 68HC11

(See Figure 6.21)

Control Bits and Flags

(See Table 6.11)

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:
  
  ```
  ldx #$1000  TFLG1 |= 0x20;
  bset $23,X,$01
  ```
Control Bits and Flags

(See Page 321)

Periodic Interrupt Using Output Capture

#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"); }

Init for Periodic Interrupt Using Output Compare

TIME rmb 2 ;inc every 1ms
init sei ;make atomic
ldaa TMSK1 ;Old value
ora #008 ;TMSK1 OC5I=1
staa TMSK1 ;Arm OC5F
ldd #0
std TIME ;initialize
ldaa #008 ;clear OC5F
staa TFLG1
ldd TCNT ;current time
addd #2000 ;first in 1 ms
std TOC5
cli ;enable
rts

(See Table 6.13)
ISR for Periodic Interrupt Using Output Compare

OC5HAN ldx TIME [5]
inx [3]
stx TIME [5]
ldaa #$08 ;clear OC5F
staa TFLG1 ;Acknowledg
ldd TOC5
add #2000 ;next
st TOC5
rti
org $FFE0
fdb OC5HAN

Square-Wave Generation in 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"); }

Square-Wave Generation

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
  TOC3=TOC3+Period; // calculate Next
  TFLG1=0x20; // clear OC3F
  TMSK1|= 0x20; // arm OC3
  TCTL1 = (TCTL1&0xCF)|0x10;
  TOC3 = TCNT+50; // first right away
  asm(" cli"); }

Initialization for Square-Wave Generation

Period rmb 2 ;units sec
Init sei ;make atomic
ldaa TMSK1 ;Old value
ora #820 ;TMSK1 OC3I=1
staa TMSK1 ;Arm OC3F
ldaa TCTL1
anda #$CF ;OM3=0
ora #110 ;OL3=1
staa TCTL1
ldaa #$20; clear OC3F
staa TFLG1
ldd TCNT ;current time
add #2000 ;first in 1 ms
st TOC3
cli ;enable
rts

(See Figure 6.23 and Tables 6.14 and 6.15)
ISR for Square-Wave Generation

Slide 17

Pulse-Width Modulation

(See Figure 6.24 and Table 6.16)

Slide 18

Pulse-Width Modulated Square-Wave (cont)

// Period is High+Low Cycles
#pragma interrupt_handler TOC3handler()
void TOC3handler(void){
    if(TCTL1&0x10){  // PA5 is now high
        TOC3=TOC3+High;  // 1 for High cyc
        TCTL1|=0xEF;  // clear on next
    } else {  // PA5 is now low
        TOC3=TOC3+Low;  // 0 for Low cycles
        TCTL1|=0x10;  // set on next int
    }
    TFLG1=0x20;  // ack, clear OC3F
}

Slide 19

Pulse-Width Modulated Square-Wave

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
    TOC3 = TCNT+50;  // first right away
    asm(" cli");  }
void main(void){
    High=8000;  Low=2000;
    ritual();
    while(1);}

Slide 20
Init for Pulse-Width Modulated Square-Wave:

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
cli ;enable
rts

ISR for Pulse-Width Modulated Square-Wave:

OC3HAN ld #20 ;clear OC3F [2]
staa TFLG1 ;Ack [4]
ldaa TCTL2 ;rise/fall? [4]
bita #$10 [2]
beq zero [3]
one ldd TOC3 [6]
add #High ;OC3 is 1 [6]
std TOC3 [5]
ldaa TCTL2 [4]
bra done [3]

ISR for Pulse-Width Modulated Square-Wave (cont):

zero ldd TOC3 [6]
add #Low ;OC is 0 [6]
std TOC3 [5]
ldaa TCTL2 [4]
oraa #$40 [2]
staa TCLT2 [4]
done rti [12]
org $FFE4
fdb OC3HAN

Delayed Pulse Generation

(See Figure 6.25)
Delayed Pulse Generation in C

```c
void Pulse(unsigned int Delay,
           unsigned int Width){
    asm("sei"); // make atomic
    TOC1=TCNT+Delay;
    TOC3=TOC1+Width;
    DC1M=0x20; // connect OC1 to PA5/OC3
    DC1D=0x20; // PA5=1 when TOC1=TCNT
    TCTL1=(TCTL1&0xCF)|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){
    DC1M=0; // disconnect OC1 from PA5
    DC1D=0;
    TCTL1&=0xCF; // disable OC3
    TMSK1&=0xDB; // 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:
  \[ f = \frac{1}{\text{fixed time}} \]

(See Figures 6.26 and 6.27)
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 = 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 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:
  \[ \Delta f = \frac{1}{(1/f) - \Delta p} - f = \frac{1}{(f - 500\text{ns})} - f \]

Frequency Measurement in C (cont)

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

Relationship Between Frequency and Period

(See Table 6.17)
Period Measurement with $\Delta p = 1\text{ms}$

- Each rising edge generates input capture interrupt.
- Output compare is 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*.

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
    TOC3=TCNT+resolution;
    Cnt=0; OverFlow=0; Done=0;
    asm(" cli");}
```

Period Measurement in C (cont)

```c
#pragma interrupt_handler TOC3handler()
void TOC3handler(void){
    TOC3=TOC3+resolution; // every 1 ms
    TFLG1=0x20; // 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 1/10s 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 Time Involved in Frequency Measurement

(See Figure 6.28)

---

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;
    }
}
```

---

Slide 37

Slide 38

Slide 39