CS/ECE 6780/5780

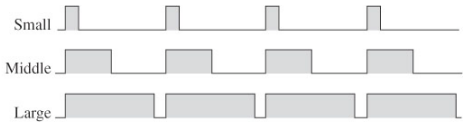
Al Davis

Today's topics:

- Output capture
- Pulse Width Modulation
- Pulse Accumulation
  - all useful options for Lab7

Output Compare

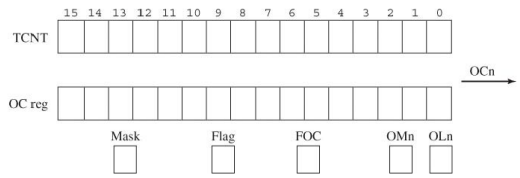
- Basic output control
  - create square waves
    - » including PWM duty cycle controlled pulses
      - for motor and actuator controls



  - Implement time delays
  - can be used w/ Input capture to measure frequency
- Similar to Input capture
  - MC9S12C32 has 8 OC channels/modules

Each OC Module

- Components
  - External output pin – OCn
  - Flag bit
  - Force output compare control bit – FOCn
  - 2 control bits: OMn, OLn
  - Interrupt mask bit
  - 16-bit output compare register TCn



Basic OC Operation

- OCn pin used to control an external device
- OC *event* occurs and sets the flag when either:
  - 16-bit TCNT register matches the 16-bit OC register
  - the software writes a 1 to the FOC bit
- OMn & OIn
  - specify the effect of the event on the output pin
- 2 or 3 actions possible when an OC *event* happens
  - always
    - » OCn output bit changes
    - » OC compare FLAG is set
  - If the mask bit is 1
    - » Interrupt is requested
  - very similar to the Input compare functionality

Control Bits & Flags

- Same as with Input capture
  - TEN must be set = TSCR1[7] to enable TCNT functions
    - » TCNT prescale bits must be set = TSCR2[2:0]
  - OCn associated with PTT[n]
    - » TTL level signal
  - Mask/Arm bits are in TIE
  - Flag bits are in TFLG1
  - TOF is in TFLG2[7]
- Differences
  - use OCn → TIOS[n]=1
    - » for Input capture TIOS[n]=DDRT[n] = 0
    - » for output compare TIOS[n] = 1 implies output
      - DDRT register value is ignored – no need to set it
  - OM & OL for modules 7:4 are in TCTL1
    - » for modules 3:0 are in TCTL2
    - » TCTL3 & TCTL4 were used for Edge bits for Input capture
  - FOCn bits are in CFORC[n] register

OM & OL Semantics

OMn	OLn	Effect of when TOCn=TCNT
0	0	Does not affect OCn
0	1	Toggle OCn
1	0	Clear OCn=0
1	1	Set OCn=1

Grrr – this could have been more intuitive – how?

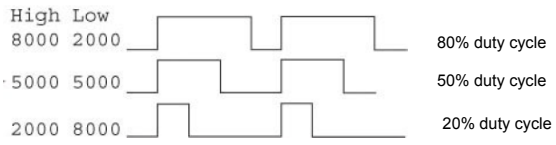
Example Application

- Create a `fixed` time delay
  1. read current 16-bit TCNT
  2. calculate `TCNT+fixed`
  3. set 16-bit TCn register to `TCNT+fixed`
  4. clear CnF flag = TFLG1[n]
     - » same semantics as with Input capture
     - » writing a 1 to the flag clears it
       - OC HW event sets the flag
       - SW can't set the flag explicitly
  5. wait for the CnF to be set
- Note
  - similar to the max latency issue w/ Input capture
    - » minimum delay is set by the delay of steps 1-4 above
    - » maximum TCNT delay is 65,536 = 2^{16} cycles

Periodic Interrupt Using Output Compare

```
#define PERIOD  1000
unsigned short Time;
void OC6_Init(void){
  asm sei         // Make atomic
  TSCR1 = 0x80;   // Turn on timer
  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 (PWM)



Parameterized PWM Duty Cycle

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


Parameterized PWM Duty Cycle (cont'd)

```
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);
}
```


PWM Overhead

- **Similar to max latency issue for input capture**
- **Need**
  - **to figure out the time it takes to process the interrupt**
  - **plus the time to execute the handler**
    - » **the if-then-else branch pattern in the handler creates a 1 cycle uncertainty**
    - » **In general you'll only care about the worst case**
      - **since that will govern your real time schedule**
- **For the previous code:**

Component	6812
Process the interrupt (cycles)	9
Execute the handler (cycles)	27-28
Total time T (cycles)	36-37

Alternative Frequency Measurement Method

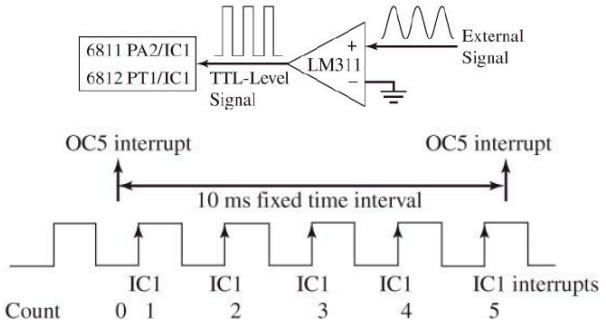
- **Direct measurement**
  - **count input (rising edge) pulses for a fixed amount of time**
    - » **use input capture to count pulses**
    - » **use output compare to create a fixed time interval**
- **Output compare handler calculates frequency**

$$f = \frac{\text{Counter}}{\text{fixed time}}$$

- **Frequency resolution is:**

$$f = \frac{1}{\text{fixed time}}$$

Alternative Method Illustrated



Frequency Measurement

```
#define Rate 20000  //  10 ms
void Init(void) {
  asm sei       // make atomic
  TSCR1 = 0x80; // Turn on timer
  TSCR2 = 0x01; // 500 ns clock
  TIOS |= 0x20; // enable OC5
  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 = 0x22; // clear C5F, C1F
  asm cli
}
```

This code makes some assumptions – what are they?

Frequency Measurement (cont'd)

```
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
}
```

What would main() look like if you wanted to keep sampling?

More PWM Options

- 6812 has a lot of support for PWM
  - pulse accumulator (PA) – 2 modes
    - » external event counting
      - PACNT is a separate 16-bit event counter
        - PAOVF = PAFLG[1] set on overflow
        - PAOVI = PACTL[1] – arms interrupt request on PAOVF
        - PAIF = PAFLG[0] – set when selected PT7 event happens
        - PAI = PACTL[0] – arms interrupt request on PAIF event
        - NOTE PAFLG bits cleared by writing a 1 (similar to other flags)
    - » gated time accumulation
      - useful for pulse width measurement
  - also associated with PTT[7]
- Primary setup using the PACTL register
  - PACTL[6] = PAEN
    - » set to 1 to enable the PA functions
  - PACTL[5:4] = PAMOD:PEDGE semantics
    - » 00 – PT7 falling edge increments PACNT, sets PAIF on falling edge
    - » 01 – PT7 rising edge increments PACNT, sets PAIF on rising edge
    - » 10 – gated time, counts when PT7=1, sets PAIF on falling edge
    - » 11 – gated time, counts when PT7=0, sets PAIF on rising edge

Directionality for PA Functions

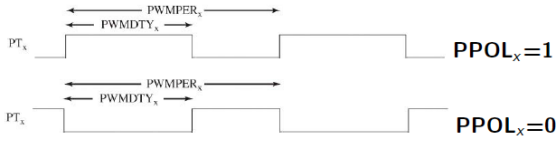
- PA can work on PT7 events
  - Independent of DDRT7 direction
    - » 0:input – stimulus comes from external device
    - » 1:output – stimulus comes from internal device
  - nice flexibility option

More PWM Options

- Dedicated hardware can create PWM signals on Port P
  - benefit is no overhead
- MODRR register can connect PWM system to Port T pins
  - MODRR[n] set connects PTT[n] to PWM system
    - » n can be 0:4
- PWME register is used to enable PWM channels
  - either 6 8-bit channels
  - or 3 16-bit channels
    - » channels 0 & 1 connected if CON01 bit is set (PWMCTL[4])
      - similarly with CON23 = PWMCTL[5]
      - and CON45 = PWMCTL[6]
- Each channel has two associated count/duration controls
  - PWMDTY – controls how long output is high
  - PWMPER – controls the period
    - » naming: PWMPER01 if CON01 set, PWMPER0 & PWMPER1 otherwise

PWM Polarity Control

- PWMPOL register controls polarity
  - e.g. whether duty cycle is high vs. low output value
  - PPOLx = PWMPOL[x]
    - » x can be 0:5 assuming 6 8-bit channels
    - » NOTE if 16 bit channels are used
      - I'm not clear on whether both PPOL bits need to be set appropriately or whether just one suffices
      - If anybody tries it let me know the answer
      - For now to be safe set both



Clock Choice

- Lots of options here
  - A & B clocks are scaled down versions of the E clock
    - » prescale bits are similar to the TCNT prescale
      - e.g. 2^V where V is the 3-bit prescale value
      - B clock prescale bits in PWMPRCLK[6:4]
      - A pescale bits in PWMPRCLK[2:0]
- Both A & B clocks can be further scaled
  - SA clock = A/PWMSCLA (8-bit register)
  - similarly SB clock = B/PWSCLB
- PWM channels & clock select
  - channels 0,1,4,5 can use A or SA clock
    - » e.g. PWMCLK[0]=0 use A clock for channel 0
      - If set to 1 use SA clock
  - channels 2 and 3 use B or SB clock
- PHEW! Lots of options & lots to remember
  - or look up sec. 6.7 of your text

8-bit PWM Output Example

```
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
  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
}
```


16-bit PWM Output Example

```
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(unsigned short duty){
  PWMDTY23 = duty; // 0 to 62500
}
```


Concluding Remarks

- Lots of fine grain detail
  - which you won't remember
  - but hopefully you now get the basic ideas
- Key is that there are 3 important concepts
  - output compare is a broadly useful technique
    - » for taking a specified action at a precise time
  - dedicated PWM modules
    - » provide same opportunity but w/ super low overhead
  - pulse accumulator useful for counting events
    - » for the 6812 they can be internally or externally sourced
- Devil is in the details
  - and there are a lot of them unfortunately
  - BUT
    - » you now have multiple LAB7 implementation options
    - » enjoy them (I hope)