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 & Ol
  * 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
  - TSK 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 \( \rightarrow \) TIOB[n]=1
    - for input capture TIOB[n]=DDRT[n]=0
    - for output compare TIOB[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

<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>

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

Periodic Interrupt Using Output Compare

```c
#define PERIOD 1000
unsigned short Time;
void O06_Init(void){
    asm sei; // Make atomic
    TSCR1 = 0x80; // Turn on timer
    TSCR2 = 0x02; // 1 MHz TCNT
    TIDT |= 0x40; // activate O06
    TIR |= 0x40; // arm O06
    TC6 = TCNT+50; // first in 50us
    Time = 0; // Initialize
    asm cl1; // enable IRQ
    void interrupt 14 O06Handler(void){
        TC6 = TC6+PERIOD; // next in 1 ms
        TFLG1 = 0x40; // acknowledge O6F
        Time++; }
```
**Pulse-Width Modulation (PWM)**

<table>
<thead>
<tr>
<th>High</th>
<th>Low</th>
<th>Duty Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>2000</td>
<td>80% cycle</td>
</tr>
<tr>
<td>5000</td>
<td>5000</td>
<td>50% cycle</td>
</tr>
<tr>
<td>2000</td>
<td>8000</td>
<td>20% cycle</td>
</tr>
</tbody>
</table>

**Parameterized PWM Duty Cycle**

```c
void Init(void){
    asm "se1 // make atomic
    TSCR1 = 0x00; // Turn on timer
    TSCR2 = 0x01; // 500 ns clock
    TO8 = 0x08; // enable OC3
    DDNT |= 0x08; // PT3 is output
    TIE |= 0x08; // Arm output compare 3
    TFL0 = 0x08; // Initially clear C3F
    TUL2 = (TUL2&0x3F)|0x40; // toggle
    TC3 = TCNT+50; // first right away
    asm cli }
```

**Parameterized PWM Duty Cycle (cont'd)**

```c
void interrupt 11 TCHandler (void){
    TFL0 = 0x08; // Ack C3F
    if (PT3 is now high
        TC3 = TC3+High; // 1 for high cycle
    } else(
        if PT3 is now low
        TC3 = TC3-Low; // 0 for low cycles
    }
    void main(void){
        High=8000; Low=2000;
        Init();
    }
```

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

```c
#define Rate 200000 // 10 ms  
void Init(void) {  
  asm sei; // make atomic  
  T8CR1 = 0x50; // Turn on timer  
  T8CR2 = 0x10; // 500 ns clock  
  T8IS = 0x00; // enable OCS  
  TIE = 0x22; // Arm OCS and IC1  
  TCS = TCNT8Rate; // First in 10 ms  
  TCLK = ((TCLK4800F2)/0x04); // CIF set on rising edges */  
  Count = 0; // Set up for first  
  Done = 0; // Set on measurements  
  TFLG1 = 0x22; // clear OSF, CIF  
  asm cli;  
}
```

This code makes some assumptions – what are they?

Frequency Measurement (cont’d)

```c
void interrupt 9 T8CHandler(void){  
  Count++; // number of rising edges  
  TFLG1 = 0x02; // ack, clear CIF  
}
```

```c
void interrupt 13 TC5Handler(void){  
  TFLG1= 0x20; // Acknowledge  
  TCS = TC5Rate; // every 10 ms  
  Freq = Count; // 100 Hz units  
  Done = C0ff;  
  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
      - PAOVF = PAFLG[1] causes interrupt request on PAOVF
      - PA0V = PACNT[0] - set when selected PT7 event happens
      - PAI = PACNT[0] - arms interrupt request on PAOVF event
    - gated time accumulation
      - useful in pulse width measurement
      - also associated with PT7[7]
  - PACNT[1] = PACMOD: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

Primary setup using the PACTL register
- PACTL[6] = PAEN
  - set to 1 to enable the PA functions
  - 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 DDR7[7] 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[0] set connects PT7[0] to PWM system
    - n can be 04
- PWME register is used to enable PWM channels
  - 8 of 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, PWMPERO & 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 05 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
      - A 8-bit prescale bits in PWMPRCLK[6:4]
    - A prescale 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
    - 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

```c
void PWM_Init(void){
    MODRR |= 0x01; // PT0 with PWM
    PWME |= 0x01; // enable channel 0
    PWMPOL |= 0x01; // PT0 high then low
    PWMCNCL &~0x08; // Clock B
    PWMCTRL |= 0x20; // Concatenate 2+3
    PWMP8CLK = (PWM8CLK&0x8F)|0x60; // B=E/64
    PWMPER23 = 62500; // 1ms period
    PWMDTY23 = 0; // initially off
}
// Set the duty cycle on PT3 output
void PWM_Duty(unsigned short duty){
    PWMDTY23 = duty; // 0 to 62500
}
```

16-bit PWM Output Example

```c
void PWM_Init(void){
    MODRR |= 0x08; // PT3 with PWM
    PWME |= 0x08; // enable channel 3
    PWMPOEL |= 0x08; // PT3 high then low
    PWMP8CLK &~0x08; // Clock B
    PWMP8CLK = (PWM8CLK&0x8F)|0x60; // B=E/64
    PWMPER23 = 62500; // 1ms period
    PWMDT32 = 0; // initially off
}
// Set the duty cycle on PT3 output
void PWM_Duty(unsigned short duty){
    PWMDT32 = 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 with 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)