CS/ECE 6780/5780

Al Davis

Today's topics:

·Input capture

·particular focus on timing measurements

·useful for 5780 Lab 7

So Far

- Familiar with
  - threads, semaphores, & interrupts
- Now move on to
  - capturing edge based inputs which generate interrupts
  - use of the TCNT timer to measure things like
    - » frequency/period of a square wave
    - » delay between events
    - » etc.
- Use this in 5780 Lab 7
  - 6870 students move into project land rather than the "weekly" labs

Input Capture Basics

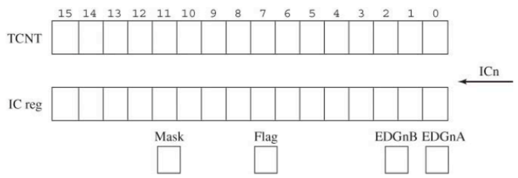
- Trigger interrupts on rising/falling/both edges
  - of TTL level external inputs
- 6812 has 8 input capture modules
- Each input capture module has
  - an external input pin: ICn
    - » associated with Port T
  - a flag bit: indicates an output has been captured
    - » not a normal memory location
      - can only be set by input capture (or output compare) event
      - SW can clear the flag by writing a 1
        - – write 0 has no effect on the flag
  - Two edge control bits
    - » EDGnB, EDGnA → care about rising, falling, or both edges
  - An interrupt mask bit (book calls this "arm")
  - A 16 bit input capture register
    - » e.g. grab the value of the TCNT timer when the event occurs

Usage Examples

- Find the frequency of a periodic square wave
  - measure the period
    - » time between a pair of rising edges
  - frequency = 1/period
- Find the duty cycle of a periodic square wave
  - duty cycle = % of period the input is a 1
  - procedure
    - » first find the period
    - » then measure the time the input is high or "ON"
      - = time between rising and falling edge
      - period/high_time = duty cycle %
- Measure jitter
  - difference between max and min time between rising (or falling) edge transitions

Basic HW Components per Channel

only 1 TCNT register however



Input Capture

- **Hardware can be set up to capture the events**
  - **registers you care about**
    - » **TSCR1[7] (a.k.a. TEN) – must be set to enable timer functions**
    - » **TSCR2[2:0] – timer prescale bits PR2, Pr1, PR0**
    - » **TIOS – set corresponding bit to 0 for input capture**
      - **same with DDRT bit**
    - » **TIE – contains the mask/arm bits for the 8 possible channels**
    - » **TFLG1 – contains the flag bits C7F … C0F**
    - » **TCTL3 – contains edge bits for IC[4:7]**
    - » **TCTL4 – contains edge bits for IC [3:0]**
    - » **8 Input Capture registers: TC0 – TC7 (one for each IC channel)**
- **On event capture**
  - **2 or 3 things happen**
    - » **always**
      - **current TCNT value is copied into the Input Capture (IC) register**
      - **input capture flag is set**
    - » **IF mask is 1**
      - **interrupt is requested**

Edge Bits

- **TCTL3 and TCTL4**

EDGnB	EDGnA	Active edge
0	0	None
0	1	Capture on rising
1	0	Capture on falling
1	1	Capture on both rising and falling

  - **TCTL3**
    - » **[EDG7B, EDG7A, …. , EDG4B, EDG4A]**
  - **TCTL4**
    - » **[EDG3B, EDG3A, … , EDG0B, EDG0A]**

Clearing and Setting Flag Bits

- **Setting can only be done by an input capture event**
- **Clearing can be done by SW**
  - **but in a seemingly weird fashion**
    - » **e.g. explicit write of 1 to the particular flag bit clears it**
- **Assume you want to clear C0F**
  - **the following works**

```
TFLG1 = 0x01;     ldy  #$1000
                  ldaa #$01
                  staa $23,Y
```

  - **this one doesn't**

```
TFLG1 |= 0x01;    ldx  #$1000
                  bset $23,X,$01
```

  - **WHY?**

Avoid bset & |= for Flag bits

- Both bset and |=
  - read current value of TFLG1
  - bitwise OR with the mask
  - $01 in this case
- Result
  - C0F gets cleared as desired
  - BUT
    - » if any of the C7F:C1F bits were set then they will be cleared as well – not as desired most likely
- Usually you will clear the flags as an acknowledge that the event has been processed
  - hence wise to avoid both
    - » bset in asm
    - » |= in C

ICn Mapping & Prescale Control

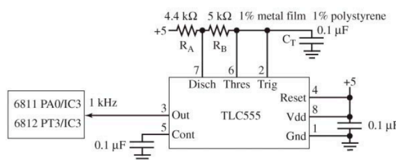
- To map ICn to PTn
  - set TIE[n] = 0
  - set DDRT[n] = 0
  - e.g. for IC3

```
DDRT  |= $08
TIE  |= $08
```

    - » note |= is fine for DDRT & TIE
      - just don't use it for TFLG1 flag manipulation
- Prescale bits – low order 3 bits of the TSCR2 register
  - taken as a value P
  - they mean divide by 2^P
    - » e.g. for a 4 MHz E Clock
    - » P=7 → divide by 128 → event every 32 µs
    - » P=3 → divide by 8 → event every 2 µs
  - if you prefer tables rather than basic idea
    - » see Table 6.5 in your text book

Input Capture Example

- Use TLC555 astable multivibrator
  - book companion CD has specs for a variety of 555 timers
  - TLC555 period = 0.693 x C_T x (R_A + $2R_B$)
    - » for 1 kHz
      - R_A = 4.4 kΩ
      - R_B = 5 kΩ
      - C_T = 0.1 µF
  - schematic



    - » stability will be based on combined R & C tolerances

Interrupt Handler Latency

- Max latency to handle the interrupt (best case 6812)
  - finish current instruction
    - » 13 cycles or 3.25 µsec
  - process the interrupt
    - » 9 cycles or 2.25 µsec
  - execute the ISR including changing `TIME` value
    - » 11 cycles or 2.75 µsec
  - max latency = 8.25 µsec
- Note best case assumes
  - no other interrupts
  - `main` doesn't disable interrupts
- What's the point
  - if clock period is faster than max latency
    - » you can't measure it correctly
  - hence important to calculate the max latency
    - » harder if it's not the best case (this example)

Example: Init & ISR C Code

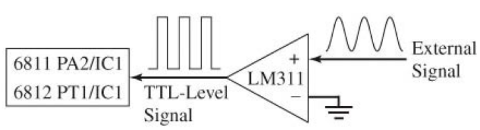
```
unsigned short Time;          // incremented
void Init(void){
  asm sei                     // make atomic
  TIOS &=~0x08;               // PT3 input capture
  DDRT &=~0x08;               // PT3 is input
  TSCR1 = 0x80;               // enable TCNT
  TSCR2 = 0x01;               // 500ns clock
  TCTL4 = (TCTL4&0x3F)|0x40;
  TIE |= 0x08;                // Arm IC3, rising
  TFLG1 = 0x08;               // initially clear
  Time = 0;
  asm cli }
void interrupt 11 IC3Han(void){
  TFLG1 = 0x08;               // acknowledge
  Time++; }
```


Period Measurement

- **Resolution**
  - **is the smallest change that can be detected**
    - » **for TCNT varies from 250 ns to 32 µs (4 MHz E Clock)**
  - **also the basic units of measurement**
    - » **e.g. TCNT ticks**
- **Precision**
  - **the number of separate & distinguishable measurements**
    - » **for TCNT = 2^{16} = 65,536 (a.k.a. 64K)**
- **Range**
  - **min and max values that can be measured**
    - » **min = 0**
    - » **max = 65,535**
- **Good measurement systems should detect**
  - **underflow and overflow**
    - » **for TCNT: TOF = TFLG2[7] indicates timer overflow**
      - **we'll ignore this for now**

Setting up a Period Measurement Experiment

- **Oh say like in Lab 7**
- **Use a waveform generator**
  - **set to TTL signal levels (5v, 0v)**
- **Or convert a sign wave to a square wave**
  - **simple OpAmp circuit**



Setup

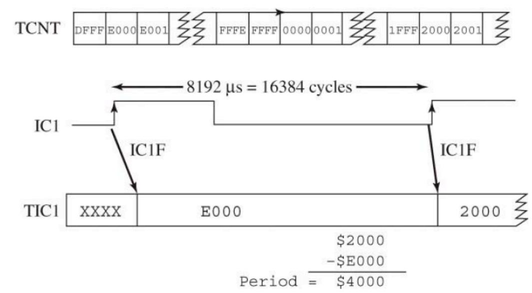
- **Some convenient assumptions to ease the example**
  - **input period is 8192 µs or every 16,384 500ns cycles**
  - **16,384 = $4000**
    - » **note subtraction of time values doesn't care if TOF occurs or not**
- **Resolution set by cycle time = 500 ns**
- **Precision**
  - **less than 2^{16}**
    - » **note need to compensate of max latency of ISR issue**
    - » **interrupts faster than max latency**
      - **some will be missed**
    - » **interrupts > but near max latency**
      - **handler occupancy goes to near 100%**
  - **in this case not a problem**

Max Latency vs. Occupancy

Component	6812
Process the interrupt (cycles,μs)	9=2.25μs
Execute the entire handler (cycles,μs)	31=7.75μs
Minimum period (cycles,μs)	40=10μs

Period (μs)	Cycles/interrupt	Time in handler (%)
10	40	100
20	40	50
100	40	10
P	40	1000/P

Period Measurement Example



Period Measurement Initialization

```
unsigned short Period;          // 500 ns units
unsigned short First;           // TCNT first edge
unsigned char Done;             // Set each rising
void Init(void){
  asm sei                       // make atomic
  TIOS &=~0x02;                 // PT1 input capture
  DDRT &=~0x02;                 // PT1 is input
  TSCR1 = 0x80;                 // enable TCNT
  TSCR2 = 0x01;                 // 500ns clock
  TCTL4 = (TCTL4&0xF3)|0x04;    // rising
  First = TCNT;                 // first will be wrong
  Done = 0;                     // set on subsequent
  TFLG1 = 0x02;                 // Clear C1F
  TIE |= 0x02;                  // Arm IC1
  asm cli }
```


Period Measurement ISR

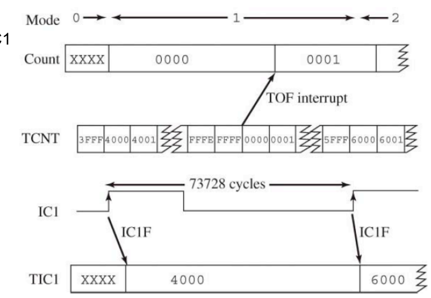
```
void interrupt 9 TC1handler(void){
  Period = TC1-First; // 500ns resolution
  First = TC1;        // Setup for next
  TFLG1 = 0x02;       // ack by clearing C1F
  Done = 0xFF;
}
```

Increasing Resolution to 32-bits

- **Every time TCNT overflows ($FFFF → $0000)**
  - **TOF flag is set**
- **So count # of times TOF is set**
  - **16 bits of precision there**
  - **plus the original 16 bits in TCNT**
  - **VIOLA (Utah French) you end up with 32-bit precision**
- **To do this**
  - **arm both input capture and timer overflow interrupts**
  - **for each timing measurements**
    - » **high order 16-bits are TOF count**
    - » **low order 16-bits are the input capture value difference**

32-bits Illustrated

MODES:
0: look for IC1
1: look for next IC1
2: measurement done

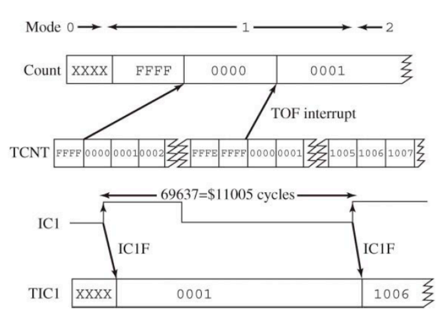


73728 = ($6000 - $4000) + 2^{16} = 0x00012000

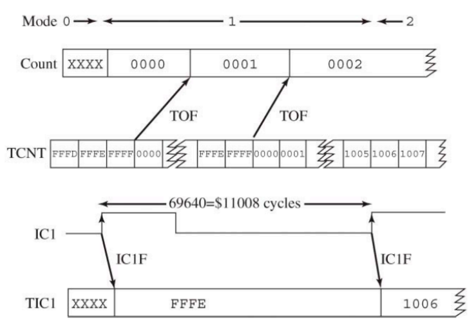
Tricky Bit

- **When IC1F and TOF get set at approximately the same time**
  - **note IC1F has a higher priority than TOF**
  - **if on first IC1F if TOF is not set**
    - » **then time is simple TIC1 value**
  - **if TOF was set**
    - » **then TOF value could have occurred just before first IC1 event in which case the TOF count is off by +1**
      - **if this is the case the high order bit of TIC1 will be 0**
      - **fix is to check for this and decrement count**
        - **effectively disable the next increment**
    - » **or it could have been set just after the first IC1 event in which case the TOF value is correct**
      - **in this case high order bit of TIC1 will be 1**
      - **in which case all is well**

TOF set Just Before IC1F Flag



TOF Set Just After IC1F Flag



32-bit Period IC1 ISR

```
void interrupt 9 TIC1handler(void){
  if(Mode==0){                // first edge
    First = TC1; Count=0;
    Mode=1;
    if(((TC1&0x8000)==0)&&(TFLG2&0x80)) Count--;
  } else {                    // second edge
    if(((TC1&0x8000)==0)&&(TFLG2&0x80)) Count++;
    Mode = 2;                 // measurement done
    MsPeriod = Count;
    LsPeriod = TC1-First;
    if(TC1<First){
      MsPeriod--;             // borrow
    }
    TIE=0x00; TSCR2=0x00; } // Disarm
  }
  TFLG1 = 0x02; }             // ack, clear C1F
```


32-bit TOF ISR

```
void interrupt 16 TOhandler(void){
  TFLG2 = 0x80;  // ack
  Count++;
  if(Count==65535){          // 35 minutes
    MsPeriod=LsPeriod=65535;
    TIE=0x00; TSCR2=0x00;  // Disarm
    Mode = 2;              // done
  }
}
```


Concluding Remarks

- **Lots of measurements are time based**
  - **6812 has a reasonably evolved set of HW support for making these measurements reasonably easy**
  - **today it was all about input capture**
    - » **and the use of the TCNT timer module**
  - **all you really need for Lab7**
- **HW timer can be much more precise than reading a clock register via SW even though there is the max latency interrupt fudge factor**
- **Next – we'll find some other interesting interrupt options**
  - **some of you already figured this out in Lab 5**
    - » **which is pretty cool**