

NET3001 F/07

Timers

MSP430 Timers

- there are 2 timers on chip
- each is a 16 bit counter
 - clocked at a certain, constant rate
 - you can read or write the count at any time
 - TAR(0x170), TBR(0x190)
- you can count up to
 - 0xFFFF (default)
 - or any other number (stored in TACCR0)
- can generate an interrupt at end of timer
- all of this is controlled by the TACTL register

Timers as Helpers

- a timer can keep track of time while our code runs
- can inform us of the passage of time
 - we can create accurate delays
 - we can act periodically
 - we can measure intervals
- these timers can also count events
 - by reconfiguring
 - we won't be using this mode in this course

TACTL

- a 16 bit register (0x160) which controls the operation of Timer A
 - clock source: bits 9/8
 - 0x0200: use the SMCLK
 - clock divider: bits 7/6
 - 0x0000: use the clock as it is
 - 0x0040: divide by 2
 - 0x0080: divide by 4
 - 0x00C0: divide by 8
 - operating mode: bits 5/4
 - 0: timer is stopped
 - 0x0010: count up to TACCR0
 - 0x0020: count up to 0xFFFF
 - interrupt enable: bit 1
 - interrupt flag: bit 0
- example: **TACTL = 0x02E0;**

Using Timers

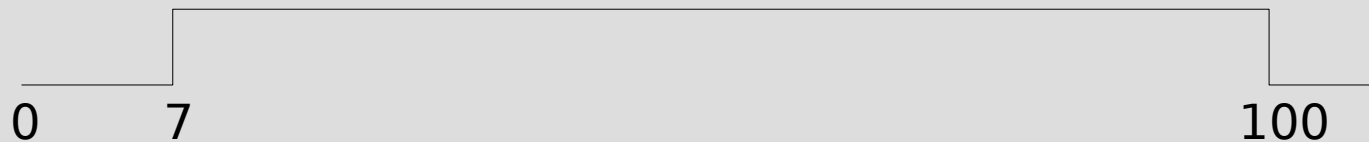
- initialize the timer by selecting the mode and clock that you want
- watch the timer frequently in your code
- ***OR***
- use interrupts
 - request the timer cause an interrupt at the end of it's count cycle

Timer Controlled Pulse Width Modulation

- these timers can also be set to change an output pin
- use the PxSEL register to select this mode
 - available on P1.1, P1.2, P1.3, P1.5, P1.6, P1.7, P2.2, P2.3, P2.4 and all of P4
 - the most interesting ones for us are
 - P2.2.....connected to the beeper
 - P1.2.....connected to the DC motor

Pulse Width Modulation

- in this mode, the timer counts up to TACCR0 (0x172)
- it sets the output pin high once the timer counts over the value in TACCR1 (0x174)
- you can create a “mostly low” or “mostly high” pattern
- example: TACCR0=100, TACCR1=7



- example: TACCR0=100, TACCR1=80



Pulse Width Modulation

- by selecting the amount of time that the pin is hi, you can control exactly the *average* signal on the pin
- this is how a microcontroller can create an analog signal
 - as long as the output device responds to the *average* signal
 - many devices do that
 - speaker
 - motor
 - LED

Time Base

- for much of this course, we want a *time base*
 - use the timer to create a periodic signal that we can use to keep our program aware of time
 - a good choice is 10msec
 - faster than human perception
 - appropriate for controlling a motor
 - slow enough to give good power saving
 - you may also chose 1msec
- simply ask Timer A to divide the master clock (1MHz) by 10,000

Initialization TimerA

– initialize at the same time you set up the direction registers

- count at 1MHz, count to 10,000 with interrupt

```
TACTL = 0x0212; // use SMCLK, interrupts on  
TACCR0 = 10000;
```

- count at 1MHz, count to 1000 with interrupt

```
TACTL = 0x0212; // use SMCLK, interrupts on  
TACCR0 = 1000;
```

- count at 15,625Hz

```
BCSCTL2 = 6; // main clock divide by 8  
TACTL = 0x02E0; // use SMCLK, divide by 8  
// interrupts off
```

Initialization TimerB

- initialize at the same time you set up the direction registers
- count at 1MHz, count to 10,000 with interrupt

```
TBCTL = 0x0210; // use SMCLK
TBCCTL0 = 0x10; // enable interrupt
TBCCR0 = 10000;
```
- count at 1MHz, count to 1000 with interrupt

```
TBCTL = 0x0210; // use SMCLK
TBCCTL0 = 0x10; // enable interrupts on
TBCCR0 = 1000;
```
- use `TIMERB0_VECTOR` for the interrupt
 - acknowledge by clearing `CCIFG` (bit0) in `TBCCTL0`

Initialization TimerB (alternate)

- initialize at the same time you set up the direction registers
- count at 1MHz, count to 10,000 with interrupt
`TBCTL = 0x0212; // count up at 1MHz, enable interrupt`
`TBCCR0 = 10000;`
- count at 1MHz, count to 1000 with interrupt
`TBCTL = 0x0212; // count up at 1MHz, enable interrupt`
`TBCCR0 = 1000;`
- use `TIMERB1_VECTOR` for the interrupt
 - acknowledge by clearing `TBIFG` (bit0) in `TBCTL`