Programming for Embedded Systems
Lecture 2

Bernhard Firner

January 26, 2013

Blinking an LED

- The "hello world" program of embedded systems
- Actually much harder than printing a string

How to Blink an LED

- We will need to control digital output
  - This means using a couple of registers
- We will also need to use a timer
  - This means using an interrupt

Step 1: Using Code Composer

1. Open CCS
2. *File* $\rightarrow$ *New* $\rightarrow$ *CCS Project*
3. For target, select MSP430G2553 (MSP430Gxxx Family)
4. Enter a project name
5. Select "Empty Project (with main.c)" in the project template
6. Click "Finish"

A Starting Point

- CCS will create a main.c with this for content:

```
#include <msp430.h>

/*
 * main.c
 */
int main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer

    return 0;
}
```

If You Aren't Using CCS

- The msp430.h include really just points to the processor that you specified when making the project
- If you are using mspdebug on Linux or OSX then use this include:

```
#include <msp430g2553.h>
```

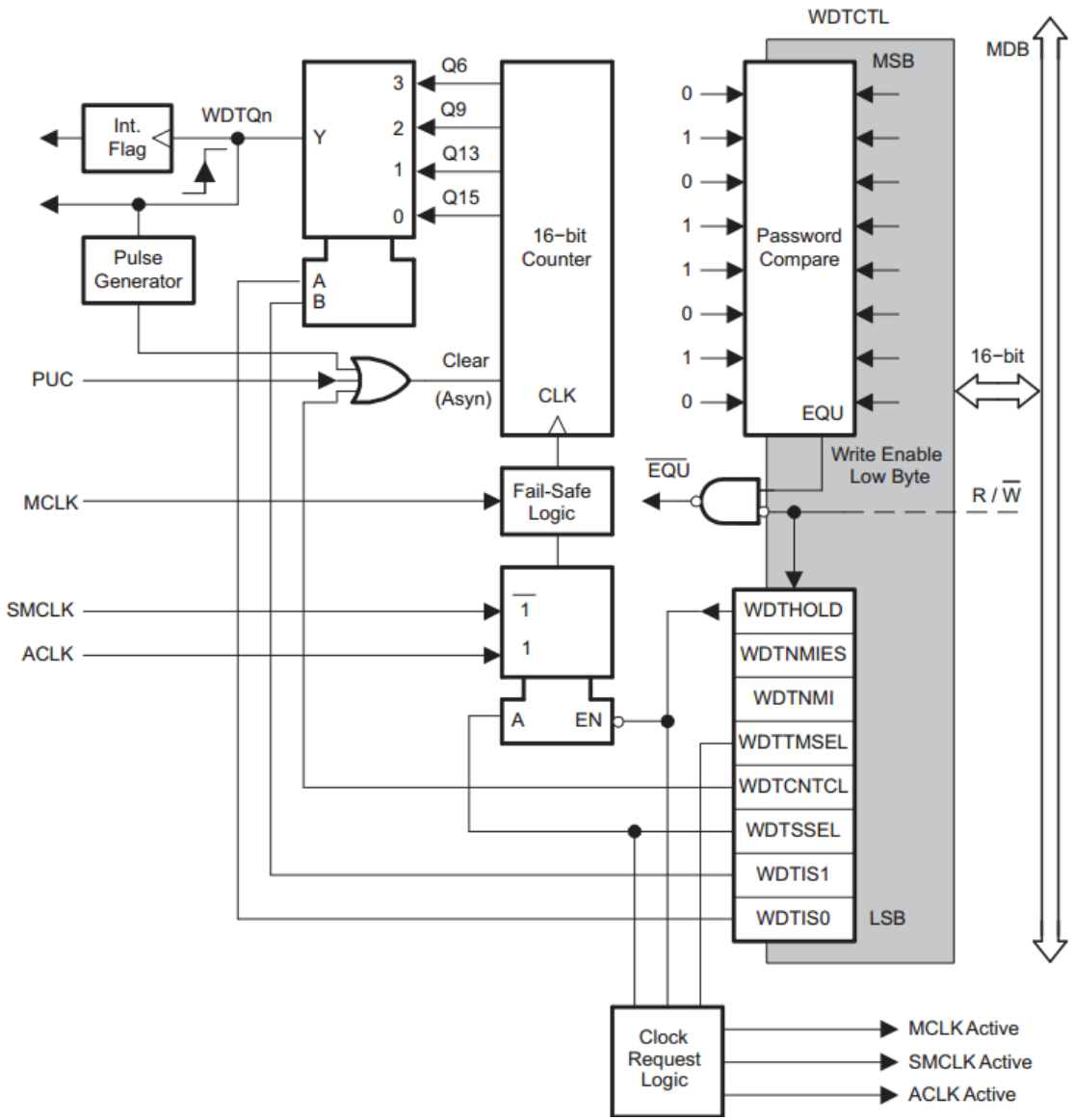
What is a Watchdog Timer?

- The default main function stops the watchdog timer
  - What is it? And why do we stop it?
- An MSP430 system that resets the chip if code stays in the same function too long
- Guards against programs that get stuck in infinite loops
- Not useful since blinking an LED *is* an infinite loop

Controlling the Watchdog Timer

- WDTCTL is the register that controls the watchdog timer
  - Stands for Watchdog Timer Control
- Covered in section 10 of the MSP430x2xx family user's guide

Block Diagram



- Watchdog block diagram from the user's guide
- Clock sources on left (MCLK, SMCLK, ACLK)
- Control register on right

Using WDTCTL

- Need to set WDTHOLD to hold the timer (8th bit)
- Must write "password" bits to prevent erroneous writing
  - This stops accidental disabling of the auto-reset
  - The header file defines this as the macro "WDTPW"

```
//Write password bits and 8th bit simultaneously
WDTCTL = WDTPW | WDTHOLD;
```

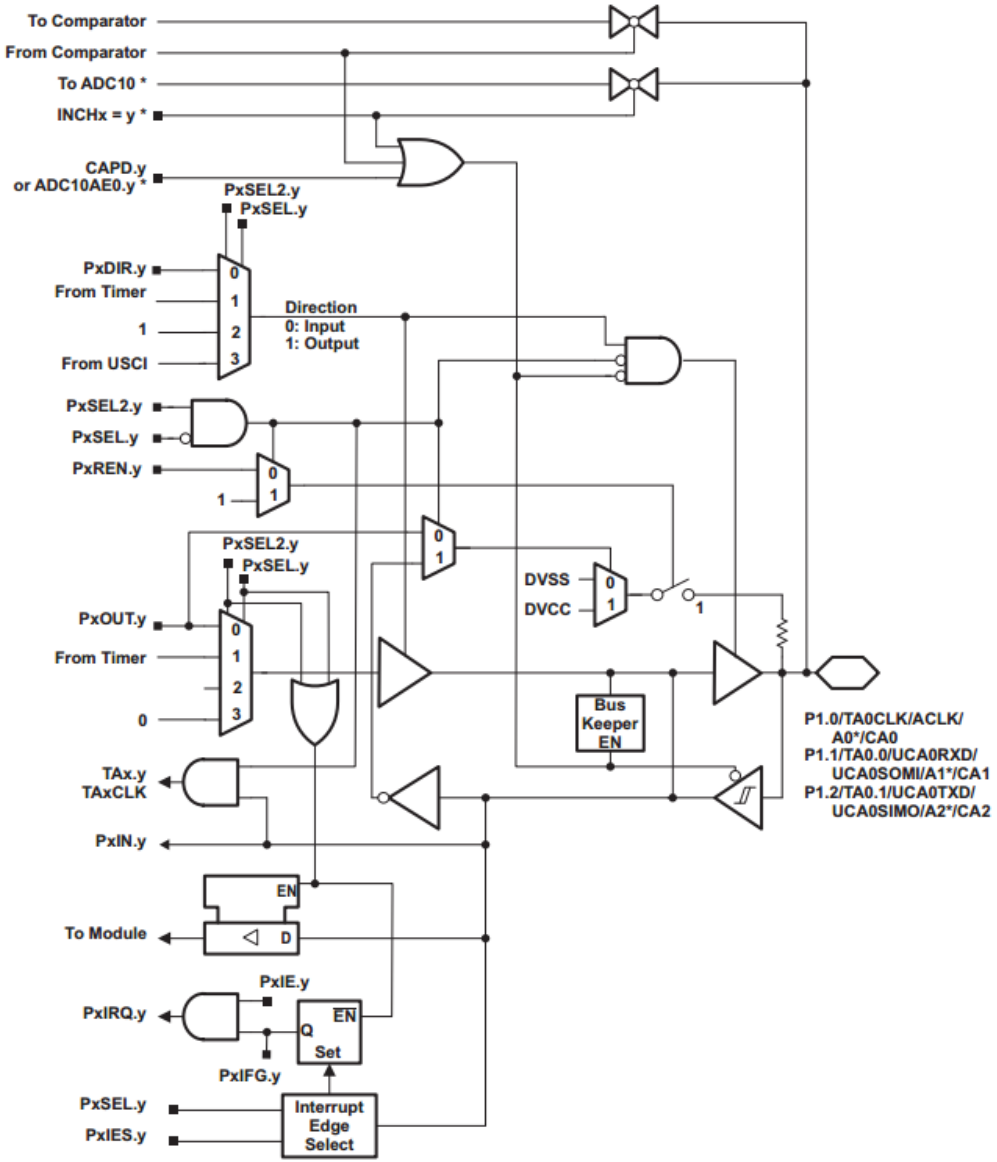
The LED

- The launchpad board has two LEDs on it
- Let's blink the one connected to P1.0
  - This means port 1, pin 0
  - There are 8 pins per port
  - We should leave the other 7 alone

Port 1 Output

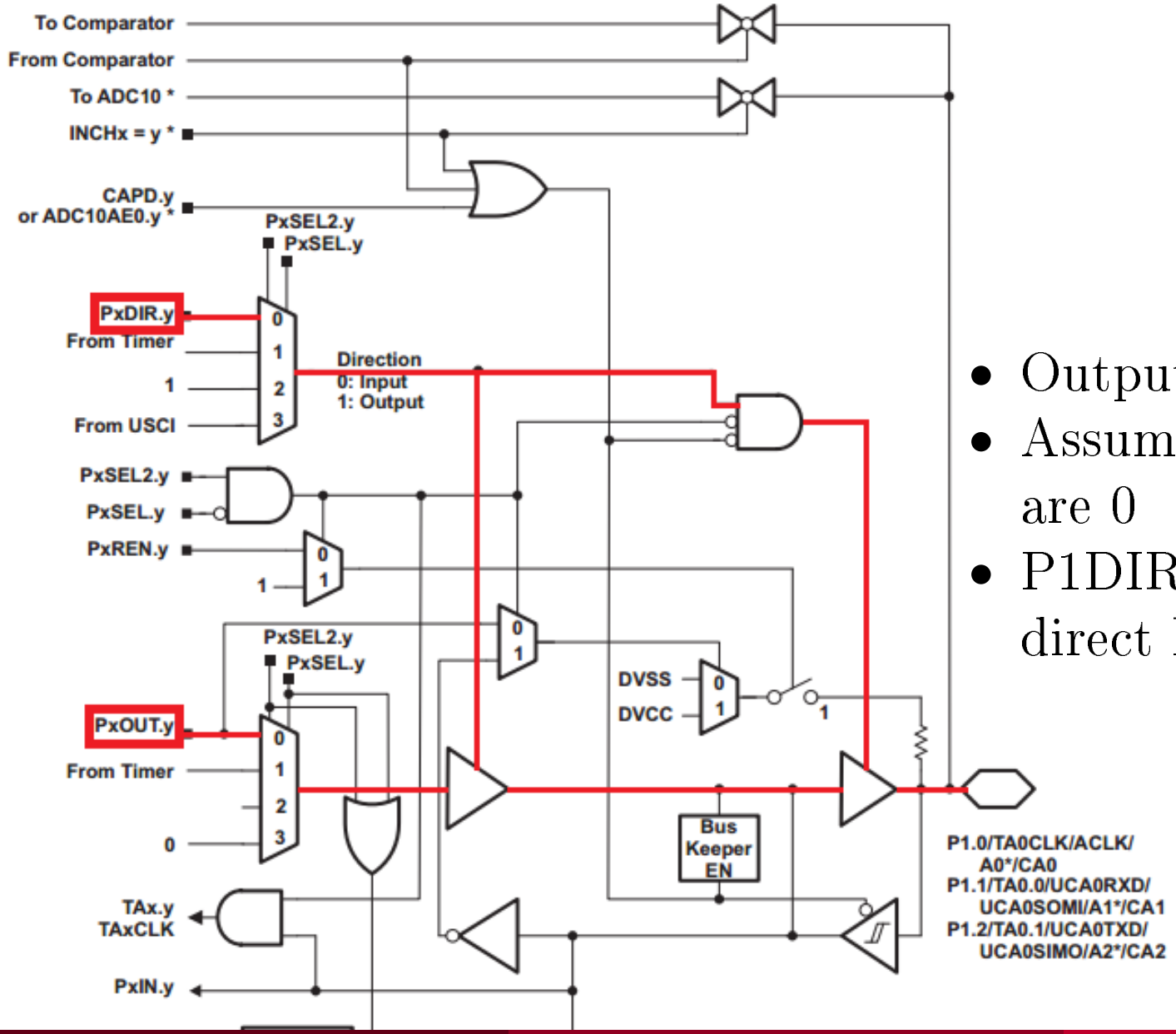
- Port 1's pins can be used for input and output
- We will need to set P1.0 to output mode
  - Controlled by *P1DIR* register
- We also need to control P1.0 on/off state
  - Controlled by *P1OUT* register
- See chapter 8 in the family user's guide

Port 1 Schematic



- Port 1 has many functions
- For clarify, in PxOUT.y
  - x is port number
  - y is pin number

Port 1 Schematic



- Output logic is in red
- Assume other inputs are 0
- P1DIR=1 enables direct line from P1OUT

What to Remember

- You don't need to study the schematics
- Do remember the P1OUT and P1DIR registers

Turning on the LED

- We can now turn on the LED
- The header file defines names for each bit
  - 16 bits in total, BIT0 - BITF

```
//Set port 1 pin 0 to output mode
P1DIR |= BIT0;
//Turn on the LED
P1OUT |= BIT0;
```

- Using the |= operator prevents us from changing other bits

Turning Off the LED

- We need to clear P1OUT.0 to turn off the LED
- Again, we want to only clear just that bit

```
//Turn off the LED
P1OUT &= ~BIT0;
```

Toggling the LED

- We could keep track of the state of the LED and use & or | depending upon its state
- It is far simpler to use the xor operation to toggle it though

```
//Toggle the state of the LED pin
P1OUT ^= BIT0;
```

Continually Toggling the LED

```
//The proper include for the chip we will be using
#include <msp430g2553.h>

//Turn on the LED and go into an infinite loop
void main(void) {
    //Turn off the Watchdog Timer
    WDTCTL = WDTPW | WDTHOLD;
    //Set port 1 pin 0 to output mode
    P1DIR |= BIT0;
    //An infinite loop
    while (1) {
        //Toggle the LED
        P1OUT |= BIT0;
    }
}
```

- This is toggling faster than our eyes can see though!

Delays

- The simplest delay just wastes time in a loop

```
while (1) {
    //Wait for 3000 loops
    int delay = 3000;
    while (0 < delay) {
        --delay;
    }
    //Toggle the LED
    P1OUT |= BIT0;
}
```

- How long is this delay though? How many clock cycles are consumed in each loop? How fast is the microcontroller clock?

Timers

- Timers are a simple way to get an exact delay
- MSP430 has several timers
- We'll use the one we already know about: the WDT
- We just need to change its mode from reset to timer

Setting the WDT Mode

- We need to set several bits to control the timer
- For ease of use they are already predefined
- Here are the definitions from msp430g2553.h

```
#define WDT_MDLY_32     (WDTPW+WDTTMSEL+WDTCNTCL)
/* 32ms interval (default) */
#define WDT_MDLY_8      (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS0)
/* 8ms     " */
#define WDT_MDLY_0_5    (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1)
/* 0.5ms   " */
#define WDT_MDLY_0_064 (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1+WDTIS0)
/* 0.064ms " */
```

- Writing those into WDTCTL will set up the timer

Setting the Clock Speed

- We still need to check the clock speed though
- There are four clock registers
- Two of them set the main clock speed
  - DCOCTL (digitally controlled oscillator control)
  - BCSCTL1 (Basic clock system control 1)

Side Topic: Clock Calibration

- Every single chip has slightly different characteristics
- Different MCUs with the same clock register settings will have different speeds
- So how do we get the same speed on different boards?
  - Each board calibrated by manufacturer
  - Calibration settings written to non-volatile memory

Using Precalibrated Values

- The precalibrated value registers are all named in the header file
- Settings for DCOCTL register are in CALDCO_XXMHZ
  - XX is a desired speed
- Settings for BCSCTL1 are in CALBC1_XXMHZ

Precalibrated Values: 16MHz

- Let's set the MSP430G2553 to 16MHz
  - This is the maximum speed

```
//Set the main clock to 16MHz
DCOCTL = CALDCO_16MHZ;
BCSCTL1 = CALBC1_16MHZ;
```

Using the WDT

- So far our code looks like this

```
int main(void) {
    //Turn off the watchdog timer while we set everything up
    WDTCTL = WDTPW | WDTHOLD;

    //Select Port 1 Pin 0 as an output pin for the LED
    P1DIR |= BIT0;

    //Set up clock at 16MHz (the maximum speed)
    DCOCTL = CALDCO_16MHZ;
    BCSCTL1 = CALBC1_16MHZ;

    //Trigger at 32 ms assuming a 1MHz clock
    //This will actually be every 2 ms since we are running a
    WDTCTL = WDT_MDLY_32;

}
```

- Now we need to set up the WDT interrupt!

Interrupts in the MSP430

- Three kinds of interrupts
  - System reset: Triggered at power on
  - Non-maskable interrupts: High priority
    - Reset pin triggered, oscillator fault, flash memory access violation
  - Maskable interrupts: Lower priority
    - Things like timers and input interrupts
    - Can all be enabled or disabled as a group

Enabling Interrupts

- The WDT interrupt is a maskable interrupt
- We must enable the maskable interrupts
  - This is done with an *intrinsic function*
  - Compilers directly convert these to assembly instructions
  - __enable_interrupt() enables maskable interrupts
  - __disable_interrupt() disables them

Enabling the Watchdog Timer Interrupt

- There are two maskable interrupt enable registers
- IE1 and IE2
- The WDT interrupt enable bit is in IE1
    - Named WDTIE

```
//Enable the WDT interrupt
IE1 |= WDTIE;
//Enable maskable interrupts
__enable_interrupt();
```

The Code Thus Far

```
int main(void) {
    //Turn off the watchdog timer
    WDTCTL = WDTPW | WDTHOLD;
    //Select Port 1 Pin 0 as an output pin
    P1DIR |= BIT0;

    //Set up clock at maximum speed
    DCOCTL = CALDCO_16MHZ;
    BCSCTL1 = CALBC1_16MHZ;

    //Trigger at 32 ms assuming a 1MHz clock
    //This will actually be every 2 ms since we are running at 16MHz
    WDTCTL = WDT_MDLY_32;

    //Let the hardware know that our software will handle the interru
    IE1 |= WDTIE;
    __enable_interrupt();

    //Infinite loop
    while (1) {
    }
}
```

Writing the Interrupt Handler

- Only the interrupt is left
- Written like a regular function with some extra syntax
- A pragma tells the compiler to put the function address in the interrupt vector table
- Interrupts do not accept parameters or return values
- The interrupt will clear the previous delay command so we need to set it again each time

```
#pragma vector=WDT_VECTOR
__interrupt void wdt(void) {
    //Set the delay again
    WDTCTL = WDT_MDLY_32;
    //Toggle the LED
    P1OUT ^= BIT0;
}
```

Choosing a Delay

- Right now the interrupt triggers every 2ms
- This is too fast
- How can we toggle the LED once every second?

Things to Remember

- Review and remember what these registers are:
  - P1DIR
  - P1OUT
  - WDTCTL
  - IE1
  - DCOCTL
  - BCSCTL1
- Remember the intrinsics __enable_interrupt() and __disable_interrupt()
- Also make sure you know how interrupt vectors work