Today's Plan:

Announcements

Review Activities 1&2

Programming in C

Using peripherals on the MSP430 (if time)

Activity 3

Announcements:

- Midterm coming on Feb 9. Will need to write simple programs in C and/or assembler for MSP430, you will need to extract information from data sheets (which I will provide), and will need to analyze/explain code samples. See examples on web site from previous years.

Announcements:

• Projects: should be starting to think about. You will need to provide a short written description (eg 1/2 page) of what you plan to assemble/build, and what parts you will require. **Bring ideas to the lab next week! Submit 1/2 page in the lab Jan 30 – Feb 3.**

• Project Scope: Your project **must** use the MSP430 as a central component. Your project should incorporate *at least one* non-trivial external hardware component (sensor, motor, display etc). Your project may (but is not required) to communicate with a host computer for display or user interaction.

• Parts: We have access to many electronic components, some mechano and many motors and sensors that you may borrow for your project. Who pays for parts that need to be acquired will depend on what the parts are, who gets to keep them, and their price.

Activity 1

- Write commands which will configure all pins of port 1 as inputs, and move the value from port 1 to register R7. Write the binary number which will be in the 16 bit register R7 after these operations assuming that the pins of port 1 were connected to 3V.

- Port P1 registers:

- P1REN ; Port P1 resistor enable
- P1SEL ; Port P1 selection
- P1DIR ; Port P1 direction
- P1OUT ; Port P1 output
- P1IN ; Port P1 input

Activity 1

- Write commands which will configure all pins of port 1 as inputs, and move the value from port 1 to register R7. Write the binary number which will be in the 16 bit register R7 after these operations assuming that the pins of port 1 were connected to 3V.

mov.b 0x00, &P1DIR
mov.b &P1IN, R7

R7 = 0xXXFF

unknown?

mov.b used so as to write to only P1DIR and not to P1DIR and whatever is next in memory (P1IFG).

mov.b used to copy only P1IN, and not also whatever is next in memory (P2OUT).

Activity 2

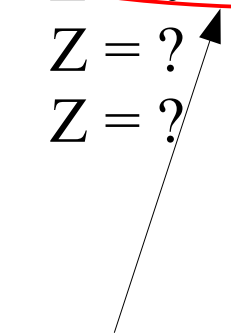
What are the values of R7 and the
Z, N, and C bits after the following commands
(assuming they were all 0 initially)

	Z = 0	N = 0	C = 0	R7 = 0
mov.w #0xF0F0, R7	Z = ?	N = ?	C = ?	R7 = ?
add.w #0xF000, R7	Z = ?	N = ?	C = ?	R7 = ?
sub.w #0xE0F0, R7	Z = ?	N = ?	C = ?	R7 = ?

Activity 2

What are the values of R7 and the
Z, N, and C bits after the following commands
(assuming they were all 0 initially)

	Z = 0	N = 0	C = 0	R7 = 0
mov.w #0xF0F0, R7	Z = 0	N = 0	C = 0	R7 = 0xF0F0
add.w #0xF000, R7	Z = ?	N = ?	C = ?	R7 = ?
sub.w #0xE0F0, R7	Z = ?	N = ?	C = ?	R7 = ?

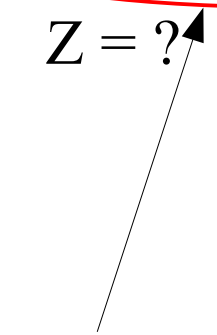


mov doesn't touch the status bits

Activity 2

What are the values of R7 and the
Z, N, and C bits after the following commands
(assuming they were all 0 initially)

	Z = 0	N = 0	C = 0	R7 = 0
mov.w #0xF0F0, R7	Z = 0	N = 0	C = 0	R7 = 0xF0F0
add.w #0xF000, R7	Z = 0	N = 1	C = 1	R7 = 0xE0F0
sub.w #0xE0F0, R7	Z = ?	N = ?	C = ?	R7 = ?



0xE0F0 is negative (if interpreted as signed)

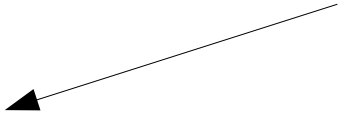
Activity 2

What are the values of R7 and the
Z, N, and C bits after the following commands
(assuming they were all 0 initially)

	Z = 0	N = 0	C = 0	R7 = 0
mov.w #0xF0F0, R7	Z = 0	N = 0	C = 0	R7 = 0xF0F0
add.w #0xF000, R7	Z = 0	N = 1	C = 1	R7 = 0xE0F0
sub.w #0xE0F0, R7	Z = 1	N = 0	C = 1	R7 = 0x0000

The carry bit is set because of the way the subtraction is done. Subtraction is done by adding the inverse of the first operand, plus one. The carry bit is set if there is a carry from the addition.

Programming in C



include header file, similar to .include in assembly. Defines symbols like P1OUT

```
#include <msp430.h>

volatile unsigned int i=0;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x41;

    for(;;){
        for ( i = 0 ; i < 20000 ; i++ ){
            if ( i  ==  0 )
                P1OUT ^= 0x01;
            if ( i  == 6000)
                P1OUT ^= 0x40;
        }
    }
}
```

Programming in C

```
#include <msp430.h>

volatile unsigned int i=0;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x41;

    for(;;){
        for ( i = 0 ; i < 20000 ; i++ ){
            if ( i  ==  0 )
                P1OUT ^= 0x01;
            if ( i  == 6000)
                P1OUT ^= 0x40;
        }
    }
}
```

Declare a global variable.
Global variables can be used anywhere in the program. The volatile keyword tells the compiler that the variable might change unexpectedly (eg in an interrupt) so it should store the variable in RAM, not just in a register.

A variable declared within a set of braces can only be accessed within those braces.

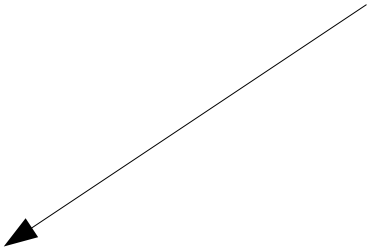
Programming in C

```
#include <msp430.h>

volatile unsigned int i=0;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x41;

    for(;;){
        for ( i = 0 ; i < 20000 ; i++ ){
            if ( i  ==  0 )
                P1OUT ^= 0x01;
            if ( i  == 6000)
                P1OUT ^= 0x40;
        }
    }
}
```

Every C program must have a routine called main. The compiler generates the code necessary for the address of the main routine to go into the reset vector. The (void) says that no parameters are passed to the function.

Programming in C

These look like ordinary C assignments, but the symbol names are special values defined in the include file.



```
#include <msp430.h>

volatile unsigned int i=0;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x41;

    for(;;){
        for ( i = 0 ; i < 20000 ; i++ ){
            if ( i  ==  0 )
                P1OUT ^= 0x01;
            if ( i  == 6000)
                P1OUT ^= 0x40;
        }
    }
}
```

Programming in C

```
#include <msp430.h>

volatile unsigned int i=0;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x41;

    for(;;){
        for ( i = 0 ; i < 20000 ; i++ ){
            if ( i  ==  0 )
                P1OUT ^= 0x01;
            if ( i  == 6000)
                P1OUT ^= 0x40;
        }
    }
}
```

Turn on the bits to ensure that P1.6 and P1.0 are outputs.
|= is an operator. This statement Is equivalent to:
P1DIR = P1DIR | 0x41;
where | is the bitwise OR operation.

Programming in C

```
#include <msp430.h>

volatile unsigned int i=0;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x41;

    for(;;){
        for ( i = 0 ; i < 20000 ; i++ ){
            if ( i  ==  0 )
                P1OUT ^= 0x01;
            if ( i  == 6000)
                P1OUT ^= 0x40;
        }
    }
}
```

for(initialization ; condition ; increment expression)

Programming in C

```
#include <msp430.h>

volatile unsigned int i=0;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x41;

    for(;;){
        for ( i = 0 ; i < 20000 ; i++ ){
            if ( i  ==  0 )
                P1OUT ^= 0x01;
            if ( i  == 6000)
                P1OUT ^= 0x40;
        }
    }
}
```


Test if i is 0. Note that equality is tested with ==
A single = is an assignment.

if (i = 0) is “valid” code, but probably doesn't do what you want!

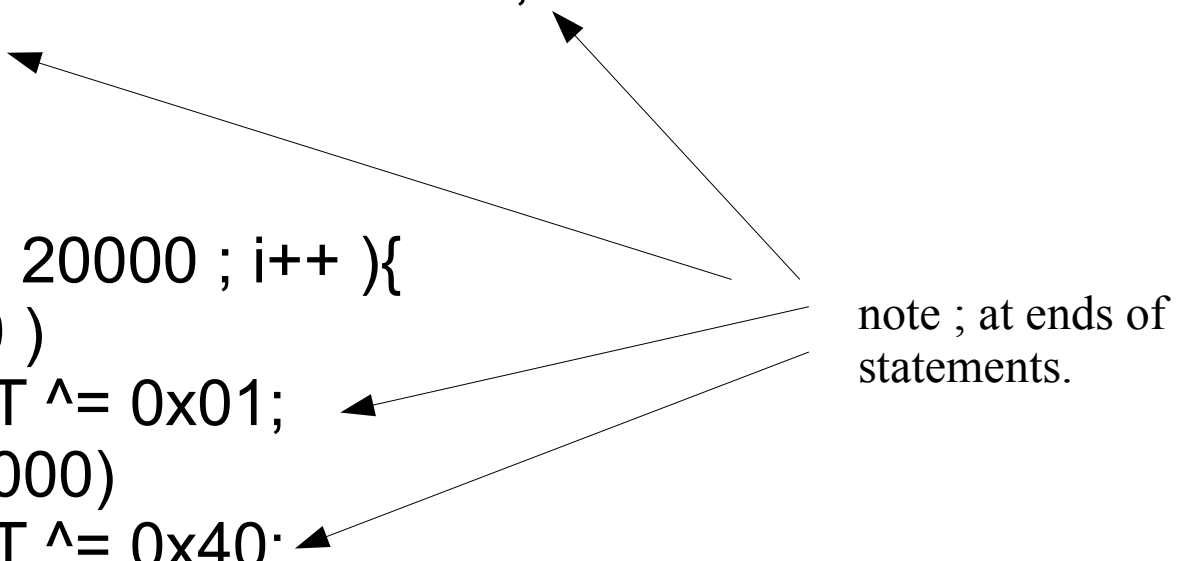
Programming in C

```
#include <msp430.h>

volatile unsigned int i=0;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x41;

    for(;;){
        for ( i = 0 ; i < 20000 ; i++ ){
            if ( i  ==  0 )
                P1OUT ^= 0x01;
            if ( i  == 6000)
                P1OUT ^= 0x40;
        }
    }
}
```

This is equivalent to
P1OUT = P1OUT ^ 0x01;

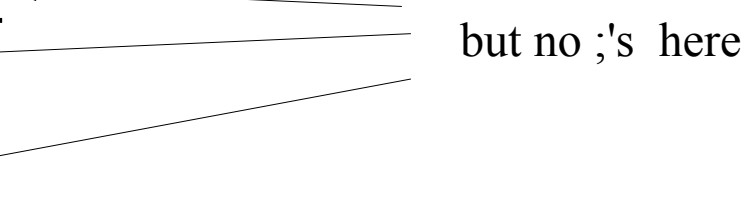
Programming in C

```
#include <msp430.h>

volatile unsigned int i=0;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x41;

    for(;;){
        for ( i = 0 ; i < 20000 ; i++ ){
            if ( i  ==  0 )
                P1OUT ^= 0x01;
            if ( i  == 6000)
                P1OUT ^= 0x40;
        }
    }
}
```



Programming in C

```
#include <msp430.h>

volatile unsigned int i=0;
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x41;

    for(;;){
        for ( i = 0 ; i < 20000 ; i++ ){
            if ( i  ==  0 )
                P1OUT ^= 0x01;
            if ( i  == 6000)
                P1OUT ^= 0x40;
        }
    }
}
```



```
if ( i == 0 ){
    a = 2;
    b = 3;
}
```

braces

vs

```
if ( i  == 0 )
    a = 2;
    b = 3;
```

executed even if i != 0

```
if ( i == 0)
    a = 2;
b = 3;
```

vs

```
if (i == 0) a = 2;
b = 3;
```

tabbing is helpful for readability. Many useful editors will help tabbing

Never do this:

```
if ( i  ==  0 );
a = 2;
```

The compiler itself ignores whitespace – it's just for readability

Programming in C
Operators:
=, +, -, *, /
% - modulus
& - bitwise AND
| - bitwise OR
^ - bitwise XOR
~ - bitwise NOT
<< - bitshift left
>> - bitshift right

Force a bit on:
a |= 2

Force a bit off:
a &= ~2

Flip a bit:
a ^= 2;

See Jean-Francois talk about these bitwise operators here:
https://www.youtube.com/watch?v=kwcjclunlug

Comparison:
==, <, >, != if (i < 3), if (i != 3)

&& - logical AND if (i == 1 && j == 2)
|| - logical OR if (i == 1 || j == 2)
! logical NOT

What about sin, cos, sqrt, etc?
These **are not** built in to the C language. However...

Programming in C

Libraries:
there are some “standard” libraries available that extend the operations you can easily use.

eg:
the math library gives access to functions like:
sin(x), cos(x), tan(x), sqrt(x), ln(x), log(x) etc...

To use math functions, you need to:
`#include <math.h>` at the top of the file, and also put:
-lm on the compilation command line.

Other libraries provide routines for string manipulations and other things...

These libraries tend to take up a substantial amount of flash and consume (precious) ram. You should try to avoid these on the MSP430 if at all possible!

Programming in C

```
int multiply_together(int x, int y)
{
    return x*y;
}
```

You can define other functions that can take arguments and return values.

```
...

y = multiply_together(4,8);
...
```

The function definition either needs to come in the file before you call it, or you need to supply a *function prototype* before you call it.

A prototype for this function would simply be:

```
int multiply_together(int x, int y);
```

Data types:

char, unsigned char – 8 bit integer (-128 to 127 or 0 to 255)

short, unsigned short (usually 16 bit integer, size on msp430 ?)

int, unsigned int – usually an integer of the native word size: 16 bits
(-32768 to 32767 or 0 to 65536)

long, unsigned long – 32 bit integer
(~ -2×10^9 to ~2×10^9 or 0 to ~4×10^9)

long long, unsigned long long – 64 bit integer
(~ -9×10^{18} to ~ 9×10^{18} or 0 to ~2×10^{19})

float – floating point number (32 bits)
(floating point operations are very expensive on a processor like the msp430 that lacks a dedicated fpu - avoid if possible).

Data Types:

In many compilers, can use types:
uint8_t/ int8_t (same as unsigned char/char)
uint16_t/int16_t (same as unsigned int/int on msp430)
uint32_t/int32_t (same as unsigned long/long on msp430)
uint64_t/int64_t

These are 'better' because you always know exactly how big they are.

To use these, add
#include <stdint.h>
at the top of the file.

- Indentation.
  Please use proper indentation of your C code
  to make it readable! Tabs of 3-4 spaces are generally best.

- There are tools that can help. Many text editors can help you indent properly.

- For Mac: install “indent” using macports.

See http://www.cprogramming.com/tutorial/style.html

for more details than you care about, see: http://en.wikipedia.org/wiki/Indent_style

```
if (i == 0){
    do thing 1;
    do thing 2;
    do thing 3;
}
else{
    do other 1;
    do other 2;
    do other 3;
}
```

vs:

```
if (i == 0)
{do thing 1;
do thing 2;
do thing 3;}
else{do other 1;
do other 2;
do other 3;}
```

```
if (i == 0){do thing 1;
do thing 2;
do thing 3;
}
else{do other 1;
do other 2;
do other 3;
}
```

Mixing C and Assembly code:

in a C program you can:

asm(“assembler text”);

For gcc, see: https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html

This can be useful for sections of code that need to be as fast as possible!

But must be done with care to make sure that you that you don't violate the compiler's assumptions about registers used!

Some Resources for C programming:

Operators

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Table

Operator Precedence:
http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Operator_precedence

C Library reference guide:
http://www.acm.uiuc.edu/webmonkeys/book/c_guide/

Textbook: Introduction to Embedded Systems Using Microcontrollers and the MSP430
http://webcat2.library.ubc.ca/vwebv/holdingsInfo?bibId=7372090

Some MSP430 examples:
http://dbindner.freeshell.org/msp430/#_increasing_the_clock_speed

Using peripherals

READING THE DATASHEET IS ESSENTIAL!
for this, Chapter 22 - ADC10

```
#include "msp430.h"

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON;         // ADC10ON
  ADC10CTL1 = INCH_1;                       // input A1
  ADC10AE0 |= 0x02;                         // PA.1 ADC option select
  P1DIR |= 0x01 ;                           // Set P1.0 to output direction

  for (;;)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    while (ADC10CTL1 & ADC10BUSY);          // ADC10BUSY?

    if (ADC10MEM < 0x2FF)
      P1OUT &= ~0x01;                       // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                        // Set P1.0 LED on

    unsigned i;
    for (i = 0xFFFF; i > 0; i--);           // Delay
  }
}
```

Using peripherals

```
#include "msp430.h"

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON;         // ADC10ON
  ADC10CTL1 = INCH_1;                       // input A1
  ADC10AE0 |= 0x02;                         // PA.1 ADC option select
  P1DIR |= 0x01 ;                           // Set P1.0 to output direction

  for (;;)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    while (ADC10CTL1 & ADC10BUSY);          // ADC10BUSY?

    if (ADC10MEM < 0x2FF)
      P1OUT &= ~0x01;                       // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                        // Set P1.0 LED on

    unsigned i;
    for (i = 0xFFFF; i > 0; i--);           // Delay
  }
}
```

```
ADC10CTL0 = ADC10SHT_2 + ADC10ON;       // ADC10ON
```


22.3.1 ADC10CTL0, ADC10 Control Register 0

15	14	13	12	11	10	9	8
SREFx			ADC10SHTx		ADC10SR	REFOUT	REFBURST
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ENC = 0

SREFx Bits 15-13 Select reference.
- 000 $V_{R+} = V_{CC}$ and $V_{R-} = V_{SS}$
- 001 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{SS}$
- 010 $V_{R+} = V_{eREF+}$ and $V_{R-} = V_{SS}$. Devices with V_{eREF+} only.
- 011 V_{R+} = Buffered V_{eREF+} and $V_{R-} = V_{SS}$. Devices with V_{eREF+} pin only.
- 100 $V_{R+} = V_{CC}$ and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with V_{eREF-} pin only.
- 101 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.
- 110 $V_{R+} = V_{eREF+}$ and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.
- 111 V_{R+} = Buffered V_{eREF+} and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.

ADC10SHTx Bits 12-11 ADC10 sample-and-hold time
- 00 4 × ADC10CLKs
- 01 8 × ADC10CLKs
- 10 16 × ADC10CLKs
- 11 64 × ADC10CLKs

ADC10SR Bit 10 ADC10 sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer.
- 0 Reference buffer supports up to ~200 ksps
- 1 Reference buffer supports up to ~50 ksps

REFOUT Bit 9 Reference output
- 0 Reference output off
- 1 Reference output on. Devices with V_{eREF+} / V_{REF+} pin only.

REFBURST Bit 8 Reference burst.
- 0 Reference buffer on continuously
- 1 Reference buffer on only during sample-and-conversion

MSC Bit 7 Multiple sample and conversion. Valid only for sequence or repeated modes.
- 0 The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion.
- 1 The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed

REF2_5V Bit 6 Reference-generator voltage. REFON must also be set.
- 0 1.5 V
- 1 2.5 V

REFON Bit 5 Reference generator on
- 0 Reference off
- 1 Reference on

ADC10ON Bit 4 ADC10 on
- 0 ADC10 off
- 1 ADC10 on

ADC10IE Bit 3 ADC10 interrupt enable
- 0 Interrupt disabled
- 1 Interrupt enabled

ADC10IFG Bit 2 ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed.
- 0 No interrupt pending
- 1 Interrupt pending

ENC Bit 1 Enable conversion
- 0 ADC10 disabled
- 1 ADC10 enabled

ADC10SC Bit 0 Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically.
- 0 No sample-and-conversion start
- 1 Start sample-and-conversion

```
ADC10CTL0 = ADC10SHT_2 + ADC10ON;       // ADC10ON
```


22.3.1 ADC10CTL0, ADC10 Control Register 0

15	14	13	12	11	10	9	8
SREFx			ADC10SHTx		ADC10SR	REFOUT	REFBURST
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ENC = 0

SREFx Bits 15-13 Select reference.
- 000 $V_{R+} = V_{CC}$ and $V_{R-} = V_{SS}$
- 001 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{SS}$
- 010 $V_{R+} = V_{eREF+}$ and $V_{R-} = V_{SS}$. Devices with V_{eREF+} only.
- 011 V_{R+} = Buffered V_{eREF+} and $V_{R-} = V_{SS}$. Devices with V_{eREF+} pin only.
- 100 $V_{R+} = V_{CC}$ and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with V_{eREF-} pin only.
- 101 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.
- 110 $V_{R+} = V_{eREF+}$ and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.
- 111 V_{R+} = Buffered V_{eREF+} and $V_{R-} = V_{REF-}/V_{eREF-}$. Devices with $V_{eREF+/-}$ pins only.

ADC10SHTx Bits 12-11 ADC10 sample-and-hold time
- 00 4 × ADC10CLKs
- 01 8 × ADC10CLKs
- 10 16 × ADC10CLKs
- 11 64 × ADC10CLKs

ADC10SR Bit 10 ADC10 sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer.
- 0 Reference buffer supports up to ~200 ksps
- 1 Reference buffer supports up to ~50 ksps

REFOUT Bit 9 Reference output
- 0 Reference output off
- 1 Reference output on. Devices with V_{eREF+} / V_{REF+} pin only.

REFBURST Bit 8 Reference burst.
- 0 Reference buffer on continuously
- 1 Reference buffer on only during sample-and-conversion

MSC Bit 7 Multiple sample and conversion. Valid only for sequence or repeated modes.
- 0 The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion.
- 1 The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed

REF2_5V Bit 6 Reference-generator voltage. REFON must also be set.
- 0 1.5 V
- 1 2.5 V

REFON Bit 5 Reference generator on
- 0 Reference off
- 1 Reference on

ADC10ON Bit 4 ADC10 on
- 0 ADC10 off
- 1 ADC10 on

ADC10IE Bit 3 ADC10 interrupt enable
- 0 Interrupt disabled
- 1 Interrupt enabled

ADC10IFG Bit 2 ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed.
- 0 No interrupt pending
- 1 Interrupt pending

ENC Bit 1 Enable conversion
- 0 ADC10 disabled
- 1 ADC10 enabled

ADC10SC Bit 0 Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically.
- 0 No sample-and-conversion start
- 1 Start sample-and-conversion

If the register descriptions don't have enough information then:

1) Look back earlier in the ADC10 chapter for more description of the operation.

2) Check the text book for an alternative description.

3) Ask a classmate or one of us!

Using peripherals

```
#include "msp430.h"

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON;         // ADC10ON
  ADC10CTL1 = INCH_1;                       // input A1
  ADC10AE0 |= 0x02;                         // PA.1 ADC option select
  P1DIR |= 0x01 ;                           // Set P1.0 to output direction

  for (;;)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    while (ADC10CTL1 & ADC10BUSY);          // ADC10BUSY?

    if (ADC10MEM < 0x2FF)
      P1OUT &= ~0x01;                       // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                        // Set P1.0 LED on

    unsigned i;
    for (i = 0xFFFF; i > 0; i--);           // Delay
  }
}
```

```
ADC10CTL1 = INCH_1;                    // input A1
```


22.3.2 ADC10CTL1, ADC10 Control Register 1

15	14	13	12	11	10	9	8
INCHx				SHSx		ADC10DF	ISSH
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC10DIVx			ADC10SSELx		CONSEQx		ADC10BUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-0

Can be modified only when ENC = 0

INCHx Bits 15-12 Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions. Only available ADC channels should be selected. See device specific datasheet.

- 0000 A0
- 0001 A1
- 0010 A2
- 0011 A3
- 0100 A4
- 0101 A5
- 0110 A6
- 0111 A7
- 1000 V_{eREF+}
- 1001 V_{REF-}/V_{eREF-}
- 1010 Temperature sensor
- 1011 (V_{CC} - V_{SS}) / 2
- 1100 (V_{CC} - V_{SS}) / 2, A12 on MSP430F22xx devices
- 1101 (V_{CC} - V_{SS}) / 2, A13 on MSP430F22xx devices
- 1110 (V_{CC} - V_{SS}) / 2, A14 on MSP430F22xx devices
- 1111 (V_{CC} - V_{SS}) / 2, A15 on MSP430F22xx devices

- 00 ADC10SC bit
- 01 Timer_A.OUT1 (1)
- 10 Timer_A.OUT0 (1)
- 11 Timer_A.OUT2 (Timer_A.OUT1 on MSP430F20x0, MSP430G2x31, and MSP430G2x30 devices) (1)

ADC10DF Bit 9 ADC10 data format

- 0 Straight binary
- 1 2s complement

ISSH Bit 8 Invert signal sample-and-hold

- 0 The sample-input signal is not inverted.
- 1 The sample-input signal is inverted.

ADC10DIVx Bits 7-5 ADC10 clock divider

- 000 /1
- 001 /2
- 010 /3
- 011 /4
- 100 /5
- 101 /6
- 110 /7
- 111 /8

ADC10SSELx Bits 4-3 ADC10 clock source select

- 00 ADC10OSC
- 01 ACLK
- 10 MCLK
- 11 SMCLK

(1) Timer triggers are from Timer0_Ax if more than one timer module exists on the device.

CONSEQx Bits 2-1 Conversion sequence mode select

- 00 Single-channel-single-conversion
- 01 Sequence-of-channels
- 10 Repeat-single-channel
- 11 Repeat-sequence-of-channels

ADC10BUSY Bit 0 ADC10 busy. This bit indicates an active sample or conversion operation

- 0 No operation is active.
- 1 A sequence, sample, or conversion is active.

```
ADC10CTL1 = INCH_1;                    // input A1
```


22.3.2 ADC10CTL1, ADC10 Control Register 1

15	14	13	12	11	10	9	8
INCHx				SHSx		ADC10DF	ISSH
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC10DIVx			ADC10SSELx		CONSEQx		ADC10BUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-0

Can be modified only when ENC = 0

INCHx Bits 15-12 Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions. Only available ADC channels should be selected. See device specific datasheet

- 0000 A0
- 0001 A1
- 0010 A2
- 0011 A3
- 0100 A4
- 0101 A5
- 0110 A6
- 0111 A7
- 1000 V_{eREF+}
- 1001 V_{REF-}/V_{eREF-}
- 1010 Temperature sensor
- 1011 (V_{CC} - V_{SS}) / 2
- 1100 (V_{CC} - V_{SS}) / 2, A12 on MSP430F22xx devices
- 1101 (V_{CC} - V_{SS}) / 2, A13 on MSP430F22xx devices
- 1110 (V_{CC} - V_{SS}) / 2, A14 on MSP430F22xx devices
- 1111 (V_{CC} - V_{SS}) / 2, A15 on MSP430F22xx devices

- 00 ADC10SC bit
- 01 Timer_A.OUT1 (1)
- 10 Timer_A.OUT0 (1)
- 11 Timer_A.OUT2 (Timer_A.OUT1 on MSP430F20x0, MSP430G2x31, and MSP430G2x30 devices) (1)

ADC10DF Bit 9 ADC10 data format
- 0 Straight binary
- 1 2s complement

ISSH Bit 8 Invert signal sample-and-hold
- 0 The sample-input signal is not inverted.
- 1 The sample-input signal is inverted.

ADC10DIVx Bits 7-5 ADC10 clock divider
- 000 /1
- 001 /2
- 010 /3
- 011 /4
- 100 /5
- 101 /6
- 110 /7
- 111 /8

ADC10SSELx Bits 4-3 ADC10 clock source select
- 00 ADC10OSC
- 01 ACLK
- 10 MCLK
- 11 SMCLK

(1) Timer triggers are from Timer0_Ax if more than one timer module exists on the device.

Table 2. Terminal Functions

TERMINAL NAME	TERMINAL NO. 14 N, PW	TERMINAL NO. 16 RSA	I/O	DESCRIPTIO
P1.0/ TA0CLK/ ACLK/ A0	2	1	I/O	General-purpose digital I/O pin Timer0_A, clock signal TACLK input ACLK signal output ADC10 analog input A0 (1)
P1.1/ TA0.0/ A1	3	2	I/O	General-purpose digital I/O pin Timer0_A, capture: CCI0A input, compare: Out0 outp ADC10 analog input A1 (1)
P1.2/ TA0.1/ A2	4	3	I/O	General-purpose digital I/O pin Timer0_A, capture: CCI1A input, compare: Out1 outp ADC10 analog input A2 (1)
P1.3/ ADC10CLK/ A3/ VREF-/VEREF	5	4	I/O	General-purpose digital I/O pin ADC10, conversion clock output (1) ADC10 analog input A3 (1) ADC10 negative reference voltage (1)
P1.4/ SMCLK/ A4/ VREF+/VEREF+/ TCK	6	5	I/O	General-purpose digital I/O pin SMCLK signal output ADC10 analog input A4 (1) ADC10 positive reference voltage (1) JTAG test clock, input terminal for device programmi
P1.5/ TA0.0/ A5/ SCLK/ TMS	7	6	I/O	General-purpose digital I/O pin Timer0_A, compare: Out0 output ADC10 analog input A5 (1) USI: clock input in I2C mode; clock input/output in SI JTAG test mode select, input terminal for device prog
P1.6/ TA0.1/ A6/ SDO/ SCL/ TDI/TCLK	8	7	I/O	General-purpose digital I/O pin Timer0_A, capture: CCI1A input, compare: Out1 outp ADC10 analog input A6 (1) USI: Data output in SPI mode USI: I2C clock in I2C mode JTAG test data input or test clock input during progra
P1.7/ A7/ SDI/ SDA/ TDO/TDI (2)	9	8	I/O	General-purpose digital I/O pin ADC10 analog input A7 (1) USI: Data input in SPI mode USI: I2C data in I2C mode JTAG test data output terminal or test data input duri

CONSEQx Bits 2-1 Conversion sequence mode select
- 00 Single-channel-single-conversion
- 01 Sequence-of-channels
- 10 Repeat-single-channel
- 11 Repeat-sequence-of-channels

ADC10BUSY Bit 0 ADC10 busy. This bit indicates an active sample or conversion operation
- 0 No operation is active.
- 1 A sequence, sample, or conversion is active.

DVSS	14	13, 14	NA	Ground reference

```
ADC10AE0 |= 0x02;                    // PA.1 ADC option select
```

22.3.3 ADC10AE0, Analog (Input) Enable Control Register 0

7	6	5	4	3	2	1	0
ADC10AE0x							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC10AE0x	Bits 7-0	ADC10 analog enable. These bits enable the corresponding pin for analog input. BIT0 corresponds to A0, BIT1 corresponds to A1, etc. The analog enable bit of not implemented channels should not be programmed to 1.
	0	Analog input disabled
	1	Analog input enabled

Using peripherals

```
#include "msp430.h"

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON;         // ADC10ON
  ADC10CTL1 = INCH_1;                       // input A1
  ADC10AE0 |= 0x02;                         // PA.1 ADC option select
  P1DIR |= 0x01 ;                           // Set P1.0 to output direction

  for (;;)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    while (ADC10CTL1 & ADC10BUSY);          // ADC10BUSY?

    if (ADC10MEM < 0x2FF)
      P1OUT &= ~0x01;                       // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                        // Set P1.0 LED on

    unsigned i;
    for (i = 0xFFFF; i > 0; i--);           // Delay
  }
}
```