Today's plan:

- Announcements
- Python and graphical user interface (GUI).
- Timers on the MSP430
- Sensors

Lecture test
during the first lecture after the midterm break (Feb 25)

I will bring to class or describe a MSP430 based system doing something. It will only use hardware covered in the manual (excluding part 9). You will have to specify all the connections and write a program in C which will run this system as shown.

You will be using your computer with all the programs you have written or tested as well as any notes or texts and the lab manual and lectures. You will submit your program as a pdf file on Canvas when the time is up (no later). 10% penalty per minute!

Its all on the honor system, you will be expected to switch off any communication programs including email, texting and so on, on your computer and not to communicate with anybody or AI.

Announcement:

Projects: you should start planning. You will need to provide a short written description (about 1/2 page) of what you plan to build. In it, describe (possibly in points):

1) the function of the project, 2) an outline of 'how it will work,' both in terms of user interface and in terms of what hardware does what. 3) A list of what kinds of parts you will require. 4) How you plan to acquire parts. Submit on Canvas before your lab 6 section.

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), two if one of them was used in the labs 1-6. Your project may (but is not required) to communicate with a host computer for display or user interaction.

Due before lab 6 (week of February 24, 2025)

Python and GUI's

For part 8 you will need Python installed and example temperature programs tested.

We used Python in Phys 219 so you are familiar with it and have it installed on your computers.

Python programs are interpreted, not compiled which make them faster to change but the execution is slower and it takes more memory.

They have the same format on windows, mac and linux

Remember that in python white spaces and indenting matters (unlike C where it's just for readability)

.

C vs Python:

Python programs generally consume much more memory and execute much more slowly than an equivalent program in C.

For our applications Python programs are fast enough.

Why are we using python?

- It is much easier to make a non-trivial program (with GUI) work cross-platform in python than C.

- Interacting with system hardware (eg serial port) is more straightforward in Python.

To make your python programs run tolerably fast:

If you are manipulating arrays of numbers,
1) always use numpy arrays
2) never iterate over the array if you can avoid it (and you can almost always avoid it!)

eg:

```
import numpy as np   # is similar to an include file, but way more powerful.
a = np.arange(0,50,1) # a is an array of 50 elements: 0, 1, 2 ... 49
b=a * 0.2                                   # b is an array 0, 0.2, 0.4, ... 0.98
```

As compared to:

```
a =  range(0,50,1)
b=[]
for i in range(50):
        b[i] = a[i]*0.2
```

Remember that in python # is a symbol for comment like // in C

```
import numpy as np
...
with serial.Serial(port,9600,timeout = 0.050) as ser
#sometimes serial has trouble unless you change the boud rate
# with timeout=0, read returns immediately, even if no data
# with timeout=.05, ser.read will wait for up to 50 ms for a byte to appear
# from the serial port, if there isn't one waiting.
...

#port = "/dev/ttyACM0"  #for Linux
#port = "COM3" #For Windows Com5 or Com7 Find out which is connected to Launchpad
#port = "/dev/tty.uart-XXXX" #For Mac

...
```

Python program List_All_coms.py is now included with temperature programs.
It works on all 3 systems.

Python resources:

There are tons of python resources on the web.

Some useful starting points:

Beginners guide to python:
https://wiki.python.org/moin/BeginnersGuide

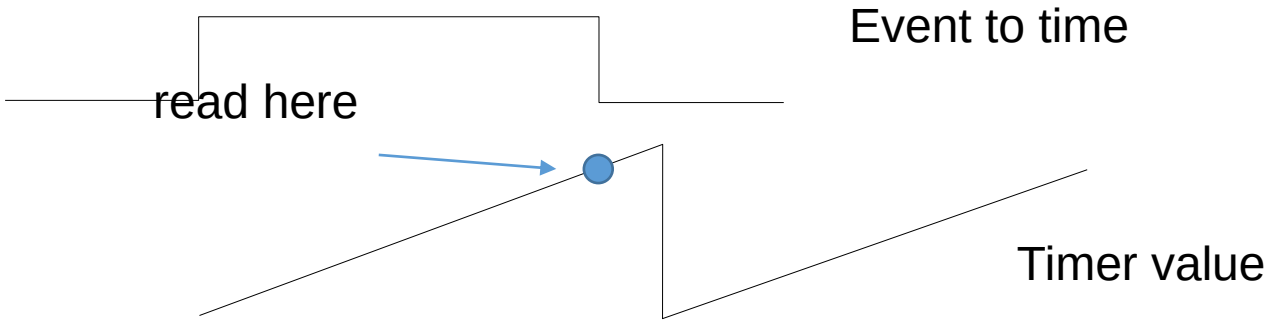
numpy:
http://www.numpy.org/

Matplotlib:
https://matplotlib.org/stable/users/index.html

Event Timing

The MSP430F5529 has many timer units that count without CPU intervention. See Chapter 17 in the family reference manual
We used one of these timers to generate PWM signals on an output

One possibility to time an event is to:
1) start the event to time
2) Reset the timer value to 0 for example register TA0R. To start the timer set it up in TAxCTR register. It starts when MC bit is changed from 0 to 2 (continuous)
3) enter a loop that continuously checks to see if the event is finished
4) read the new timer value (overflows?)



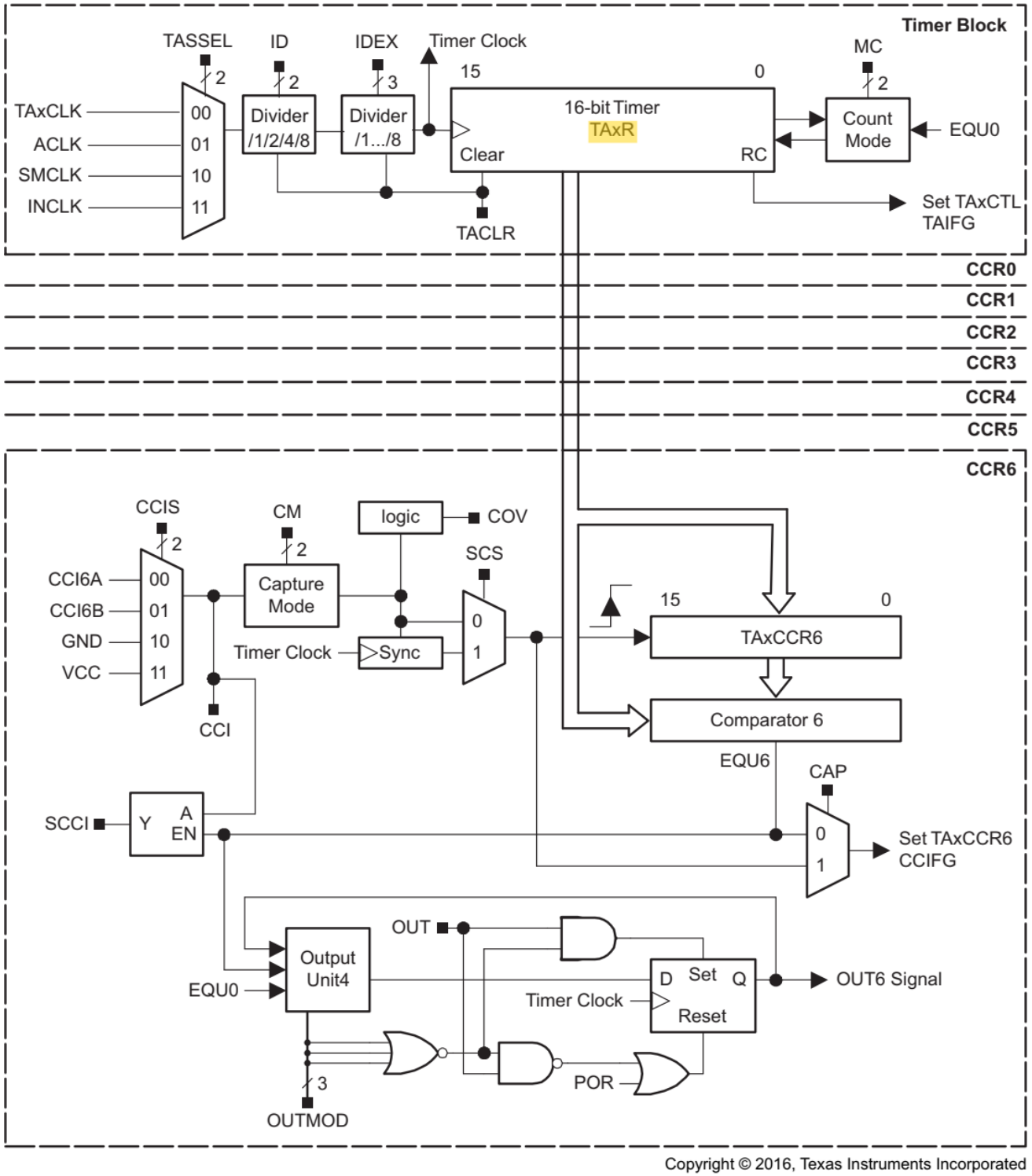


Figure 17-1. Timer_A Block Diagram

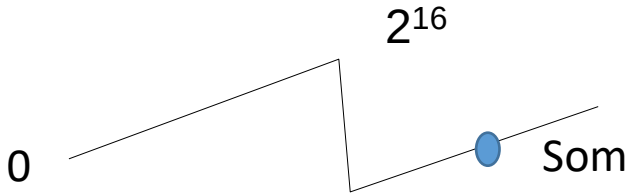
Example part of the code

```
unsigned int time;
TA0CTL = TASSEL_2 + MC_2 + ID_0;
                    // Timer A control set to SMCLK, MC_2 count up, no division –
                    //keep 1.048MHz,
                      //start the event here
TAOR=0;              //Set timer counter to 0
                     // checking for the event to finish possibly with a while loop

time = TA0R;   //read the counter

// as long as there is no overflow, this gives the right answer
// timer must be configured to count all the way to 0xFFFF and not reset to 0 at TA0CCR0
//like when we used it for PWM)
//
```

Event Timing - overflows

The timing registers are 16 bits long and run at approximately 1 MHz by default if fed by SM
means that any time longer than 2^{16} microseconds will create an overflow – the counter value go
zero and start again.

Event to time



Some number smaller than the time we wanted to measure

Timer value

Event Timing – Dealing with overflows

Two strategies:

1) Guarantee that the event to be timed is never longer than one timer period (2^16 counts). How? Hardware prescale so that the counter counts slowly enough (set clock divider with IDx bits in TAxCTL)

PRO: might be easiest solution.
CON: might limit resolution.

2) Keep track of overflows (interrupts, or count overflows)

PRO: get the full resolution possible
CON: more complicated code needed to track overflows

17.3.1 TAxCTL Register

Timer_Ax Control Register

Figure 17-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Table 17-4. TAxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MC = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit clears TAR, the clock divider logic (the divider setting remains unchanged), and the count direction. The TACLR bit is automatically reset and is always read as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending

General Advice:

Get pieces of a complicated system working one by one.

Ensure that each new piece doesn't break any old pieces.

When you combine programs make sure that all the setups are in the beginning. Avoid set up commands in the loops unless needed

Start with the simplest way, then if you have time and/or a good reason, move to a better way.

For example try to get the distance sensor working with 1 byte accuracy and than may be try for 2 bytes

The debugger, voltmeter and oscilloscope can be very helpful!

Sensors and Actuators

Available sensors:

- Optical sensors
- Temperature
- Humidity
- Magnetic field
- Pressure
- Distance
- Position and bend
- Accelerometer
- Gyroscope
- Alcohol
- GPS
- Infrared
- Infrared camera sensor
- Heart beat and blood oxygenetion

Actuators

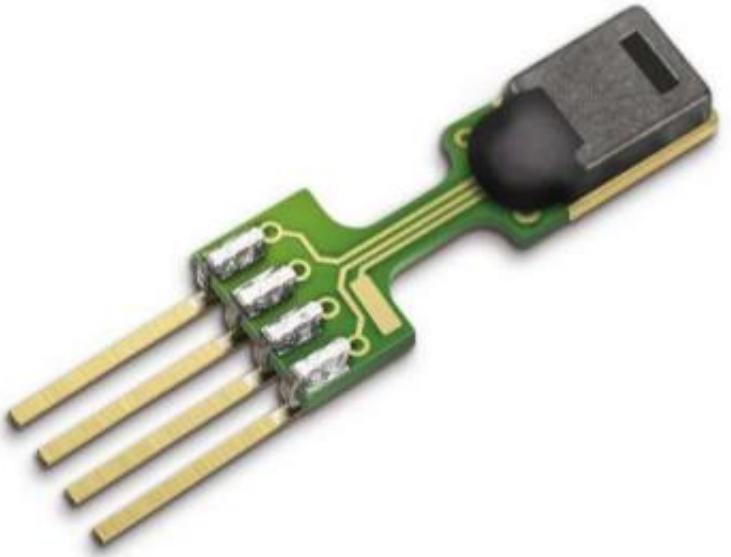
- Servo motors
- Stepper motors
- DC motors
- Relays,
- Solenoids
- “Robot” platforms

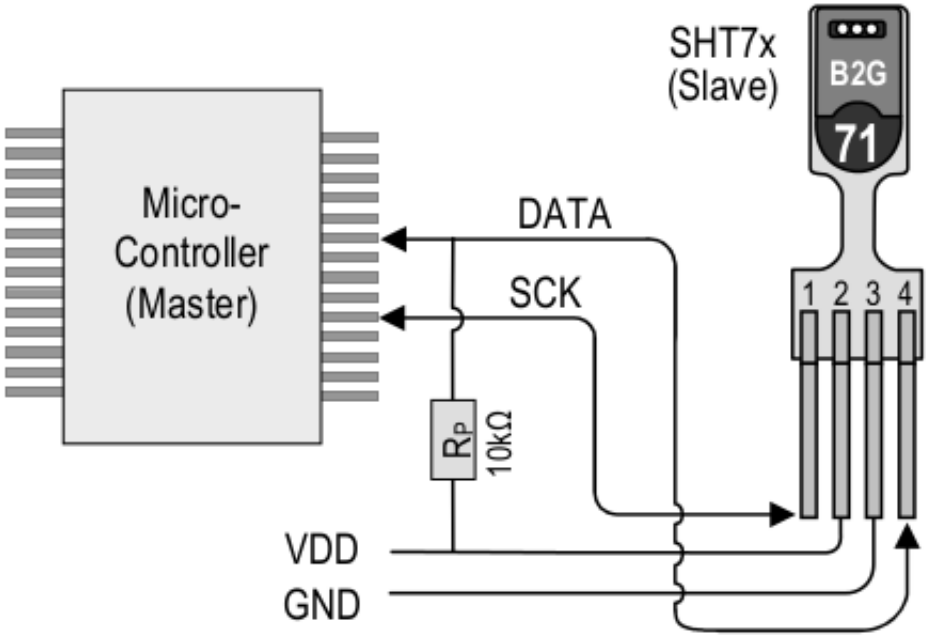
MQ analog gas sensors

1. MQ-2 Smoke Sensor
2. MQ-3 Alcohol Sensor
3. MQ-4 Methane Sensor
4. MQ-5 LPG Natural Gas City Gas Sensor
5. MQ-6 isobutane propane sensor
6. MQ-7 Carbon Monoxide Sensor Module
7. MQ-8 hydrogen sensor
8. MQ-9 Carbon Monoxide Combustible Gas Sensor
9. MQ-135 air quality detection sensor

SHT75 Temperature and Humidity Sensor

- Fully Calibrated
- Digital output
- Low power consumption
- Excellent long term stability
- two-wire serial interface.
- Accuracy +/- 0.5C



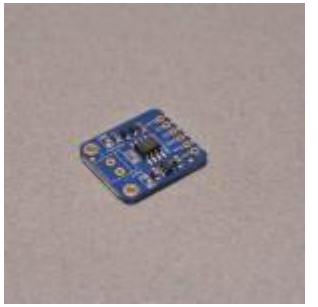


DS18B20 Temperature only

-Inexpensive,
- somewhat complicated interface
- high resolution 0.0625 degrees
- accuracy (+/- 0.5C)
- easy to multiplex many sensors

Also:

- Thermocouples
- Thermistors
- IR no contact sensor

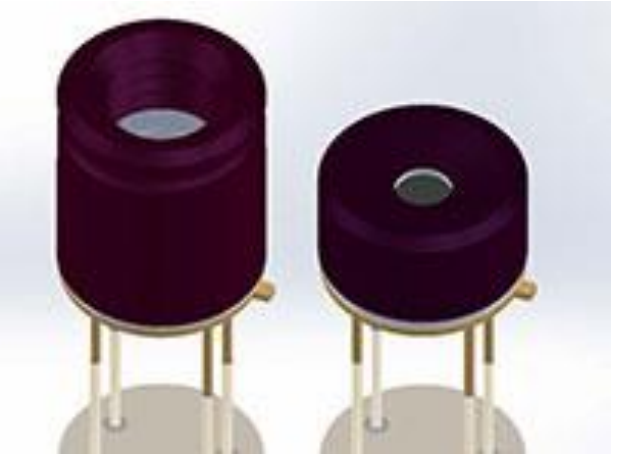


MLX90614:

Infrared camera sensor

MLX90640 FIR Sensor

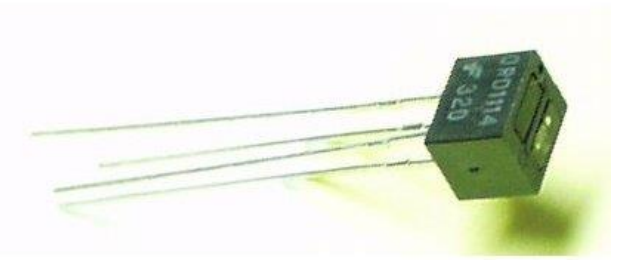
- Small size, low cost 32 pixel x 24 pixel IR array
- Easy to integrate
- Industry-standard 4 lead TO39 package
- Factory calibrated
- I^2C compatible digital interface
- Programmable frame rate 0.5 Hz to 32 Hz
- 3 V supply voltage
- 2 FOV options: 55° x 35° and 110° x 75°



Distance Sensors:

- Optical:

short range QRD1114,
medium range GP2D12



- Ultrasonic