EEL 4742 – Embedded Systems
Module 2 – Intro to MSP430

Hadi Kamali

Department of Electrical and Computer Engineering (**ECE**)
University of Central Florida

Office Location/phone: HEC435 – (407) 823-0764
webpage: https://www.ece.ucf.edu/~kamali/
e-mail: kamali@ucf.edu

HAVEN Research Group
https://haven.ece.ucf.edu/

Adopted from previous UCF EEL4742C by Zakhia Abichar & Zhishan Guo

A recap on Basics of Bit Masking & Bit Field

- Four different operations can be performed
  - Set bit, clear bit, toggle bit, check the status of a bit
- Use of Bit Fields for better readiness

Data = 0b 0 0 0 0 0 0 0 0
bit fields A B C D

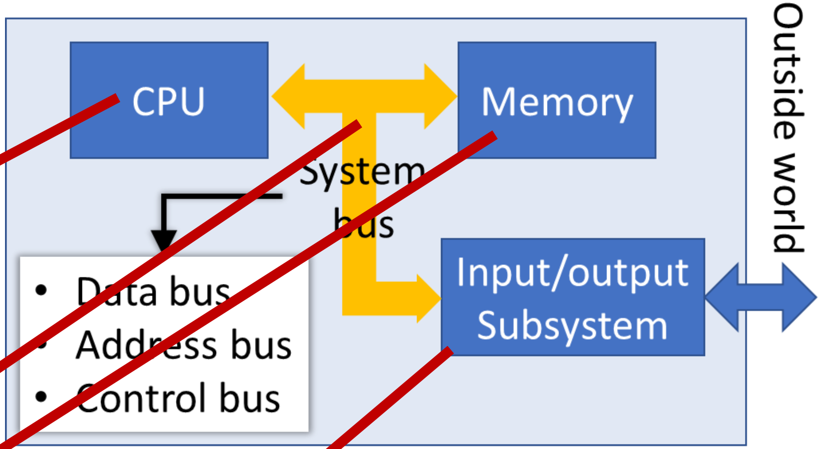
- Lab 1 Overview
  - Flashing LEDs
  - Set the frequency of flashing
    - Simple delay functions
      - Nested loop
      - Long variables
      - _delay() function

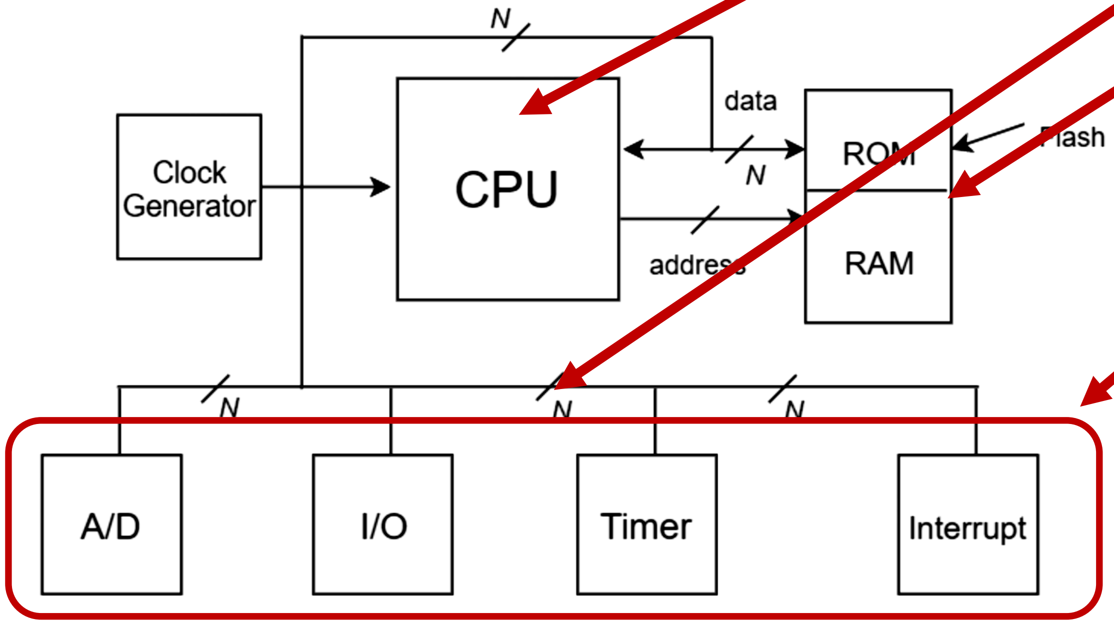
Some of you already did lab1? Any feedback? It should have been straightforward…

Microcomputer

- The minimal set of hardware components that connect together to form a computing system
  - CPU
  - Memory
  - Input/output subsystem
  - System bus

Memory: *stores data and programs,*
A/D: *converts analog signals to digital,*
Timer: *control time's events,*
I/O: *digital in / out,*
Interrupts: *to control program execution.*

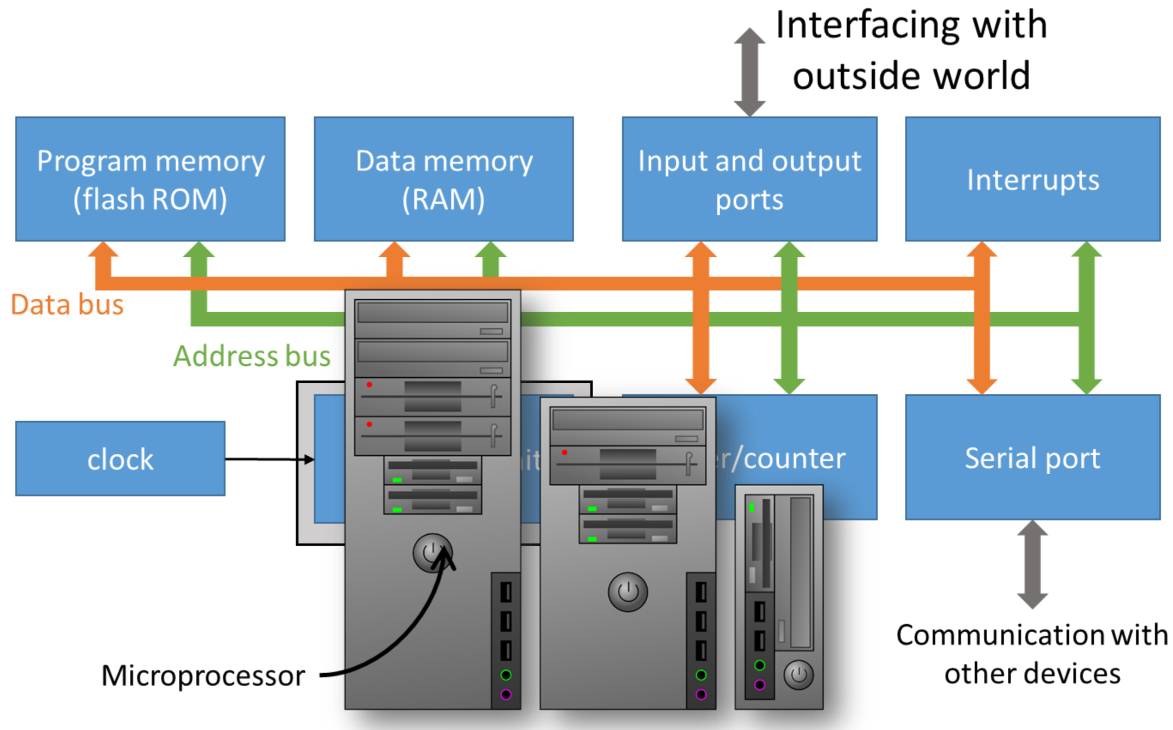




Input/output subsystem is also called the ***peripheral*** *subsystem. It includes all components that support and interact with the CPU and memory in an embedded system*

Microcomputer: Microprocessors (MPU)

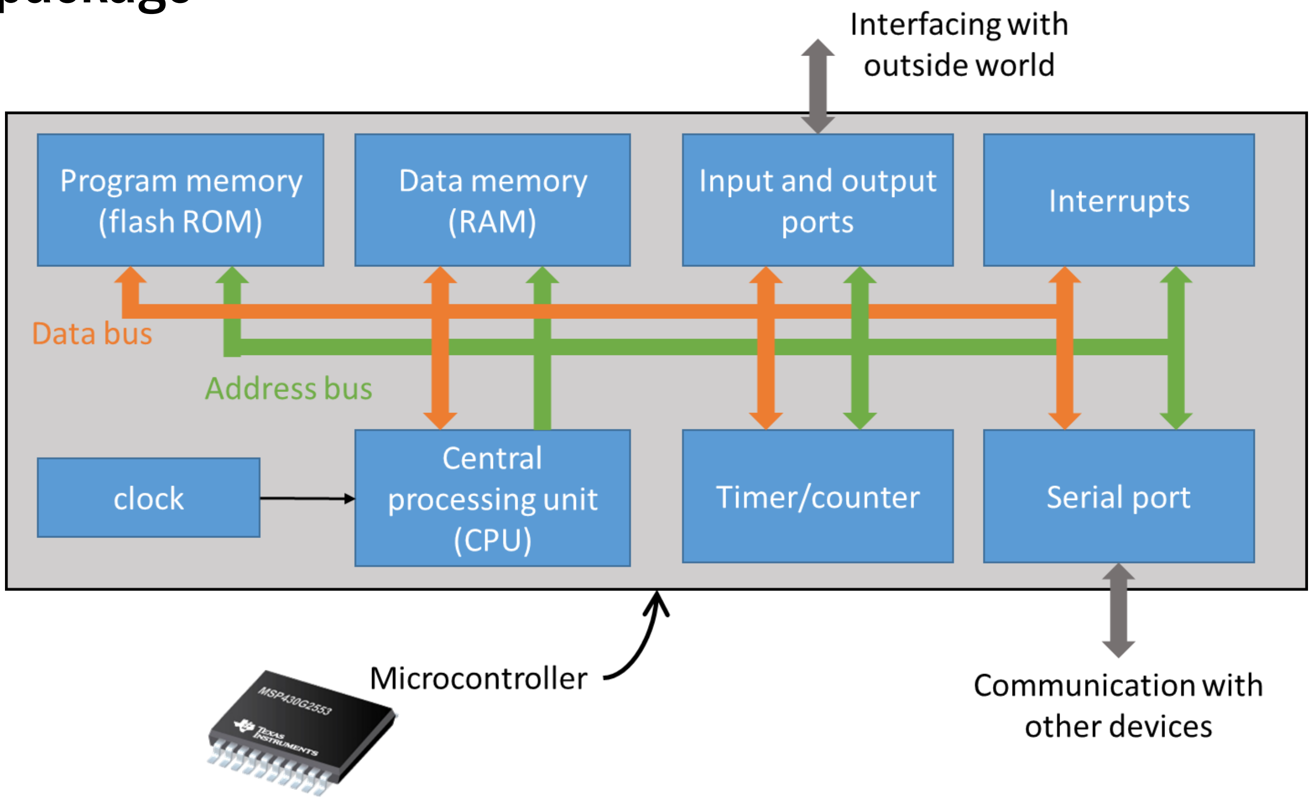
- When it comes to microprocessors, it is more disaggregated.
  - Less integration is happening in a single chip!
  - The microprocessor chip itself contains only the CPU and everything else is connected externally.
  - A very good example of this: PC. In a PC, everything is connected external to the CPU motherboard.



*It is **multiple chips** connected on a baseboard (e.g., PC mainboard where all hardware are connected to)!*

Microcomputer: Microcontrollers (MCU/µC)

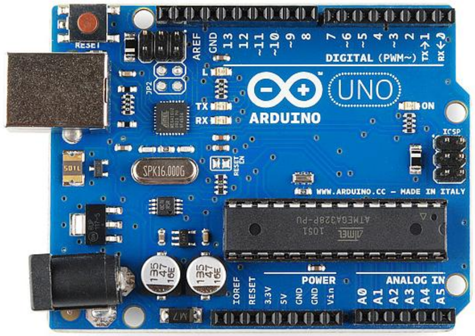
- When it comes to microcontrollers, it is more integrated into a single chip.
  - More integration is happening in a single chip!
  - Some peripherals comes as part of the chip package along with the CPU and memory.
  - Naturally, from development point of view, this is much simpler to use than microprocessors, since we don't have to have additional circuitry to connect the CPU with the peripherals.



*It is **Single chip** where all CPU, memory, peripherals packaged in the IC.*

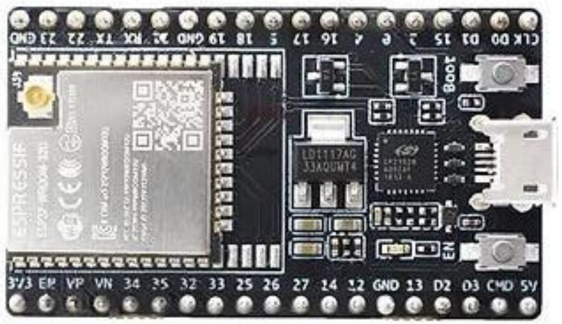
Microcomputer: Microcontrollers (MCU/µC)

- **Some of well-known Microcontrollers around us...**

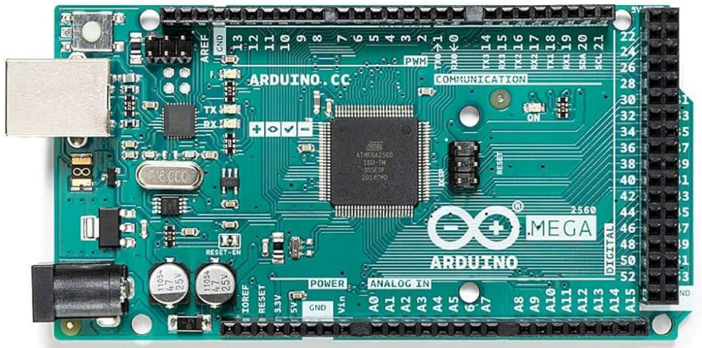


Arduino Uno (ATmega328P)
easy to expand projects

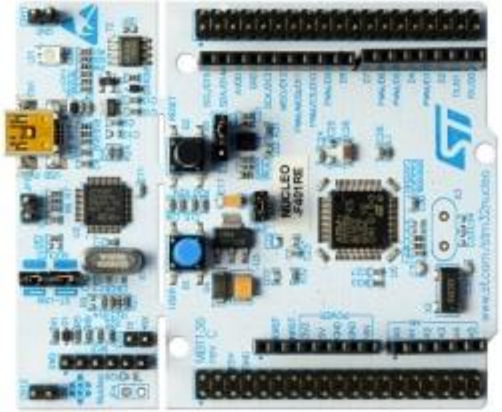
Rasberry PI Pico (RP2040)
w/ programmable I/O (custom)



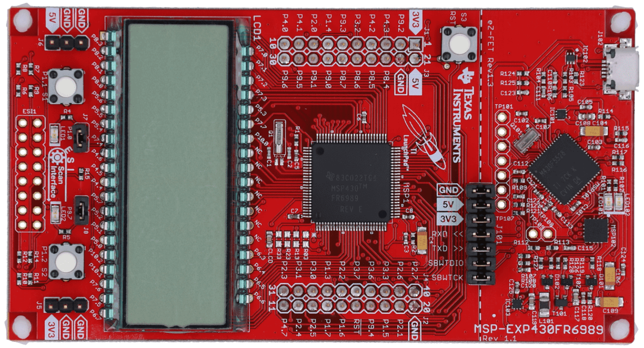
ESP32 Dev Board (ESP32)
dual-core processor and integrated Wi-Fi and Bluetooth



Arduino MEGA (ATmega2560)
large number of input/output (I/O) pins



Nucleo F401RE (ARM Cortex-M)
Complex with lots of peripherals



TI Launchpad (MSP430)
ultra-low power consumption

MCU vs. MPU

- Some main differences between microcontrollers and microprocessors

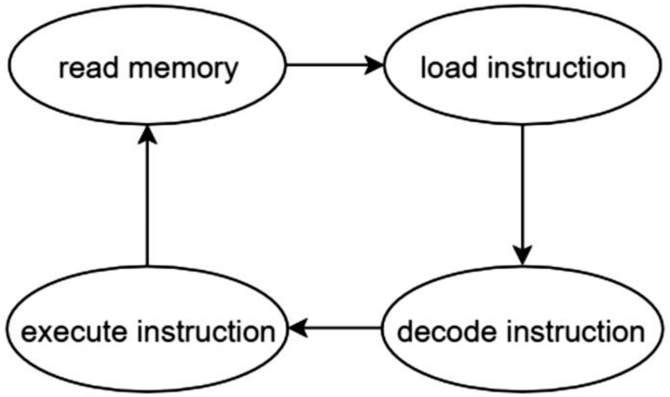
Microprocessor	Microcontroller
*CPU, RAM, ROM and peripherals are **separate** components*	*CPU, RAM, ROM and peripherals are **integrated** into a single chip*
***Flexibility** in choosing the memory size and type of peripherals*	*The on-chip RAM, ROM is fixed. Peripherals are **fixed***
*Suitable for **computation intensive** applications*	*Suitable for **control-oriented** applications*
***High** processing power*	***Low** processing power*
***High** power consumption*	***Low** power consumption*
Expensive	*Inexpensive*
Typically 32/64 bit	*Typically 8/16 bit*
*Instruction set focuses on **processing-intensive operations***	*Instruction set focuses on **control and bit-level operations***

Features of Typical Microcontroller

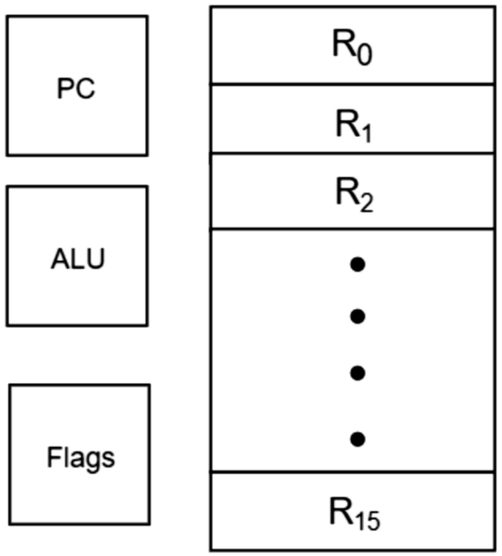
- Central Processing Unit (CPU)
  - Arithmetic Logic Unit (ALU)
    - the brain of the CPU where arithmetic and logic operations takes place.
  - Special registers such as program counter (PC), stack pointer (SP) and status register (SR)
  - General purpose registers
  - Instruction decoder
  - Other supporting components to handle resets, interrupts, frequency of operation, bus interface logic, etc.

Program counter is used to keep track of the address of the next instruction to execute

A composition of registers, ALU, state machine, and flags.



Flags give the status of the processor
N: negative flag
C: carry flag
Z: zero flag
V: overflow flag



Features of Typical Microcontroller

- Memory
  - Program memory
    - Contains the list of instructions to execute.
    - PC is the index required to be used for reading this memory one by one.
  - Data memory
    - To store data for processing (e.g., image files, sensor outputs, etc.)
    - Used during run-time to store variables (intermediate calculation)

Features of Typical Microcontroller

- Memory --- Read-only memory (ROM)
  - Non-volatile memory (NVM)
    - Retains contents when power is removed
  - Stores the program
    - Instructions to be executed by the CPU
  - Stores the bootloader
    - Any preliminary part of boot up required for MCU (like OS initialization in RTOS)
  - Most common type of memory used in modern microcontrollers is the flash memory
  - Contents of a flash memory can be both programmed and erased electronically
    - e.g., EEPROM

Features of Typical Microcontroller

- Memory --- Random-access memory (RAM)
  - Volatile memory
    - Contents will be lost, when power is OFF!
  - Used to store temporary variables in a program
    - x=5;
    - y=6;
    - z:=x+y;
    - z:=z+x;

Features of Typical Microcontroller

- Most common peripherals
  - Timer
    - Watchdog timer
  - Real-time clock (RTC)
  - Communication interfaces
  - Analog-to-Digital converter (ADC)
  - General purpose input/output (GPIO)

Features of Typical Microcontroller

- Most common peripherals
  - **Timer**
    - Watchdog timer
  - Real-time clock (RTC)
  - Communication interfaces
  - Analog-to-Digital converter (ADC)
  - General purpose input/output (GPIO)

*Timer, as the name suggest, **counts time** based on the clock signal.*

*They can **count up** or **down**, from a user-set initial value.*

*Once it reaches its maximum value, it **interrupts** the CPU.*

Timer size	Maximum value
8-bit	*(2^8) 255*
16-bit	*(2^{16}) 65,535*
32-bit	*(2^{32}) 4,294,967,295*

Features of Typical Microcontroller

- Most common peripherals
  - Timer
    - <u>Watchdog timer</u>
  - Real-time clock (RTC)
  - Communication interfaces
  - Analog-to-Digital converter (ADC)
  - General purpose input/output (GPIO)

A special timer, used to keep the system in check.

*Used to protect the system against software failure stuck in an **unintended infinite loop**.*

If the timer reaches maximum value, it triggers a reset. So, the program has to continually restart the watchdog timer to prevent it from reaching the maximum value.

When watchdog timer has no reset, then it means the software is stuck somewhere due to an unprecedented issue.

*For any freeze of MPCs, a human user can restart it. But in MCUs, as there is no one monitoring the system, humans can't manually reset the system in case of any error. In such cases, a **<u>watchdog timer is used to restart the system</u>**.*

Features of Typical Microcontroller

- Most common peripherals
  - Timer
    - Watchdog timer
  - **Real-time clock (RTC)**
  - Communication interfaces
  - Analog-to-Digital converter (ADC)
  - General purpose input/output (GPIO)

*Used to **track the time** of current time and date.*

The "timer" depends on the clock signal, which may not accurately track time. However, RTC is far more accurate than a normal timer.

RTC is needed in applications where time is important (e.g., data logging in an IoT application, or clocks in appliances, or programmable alarms).

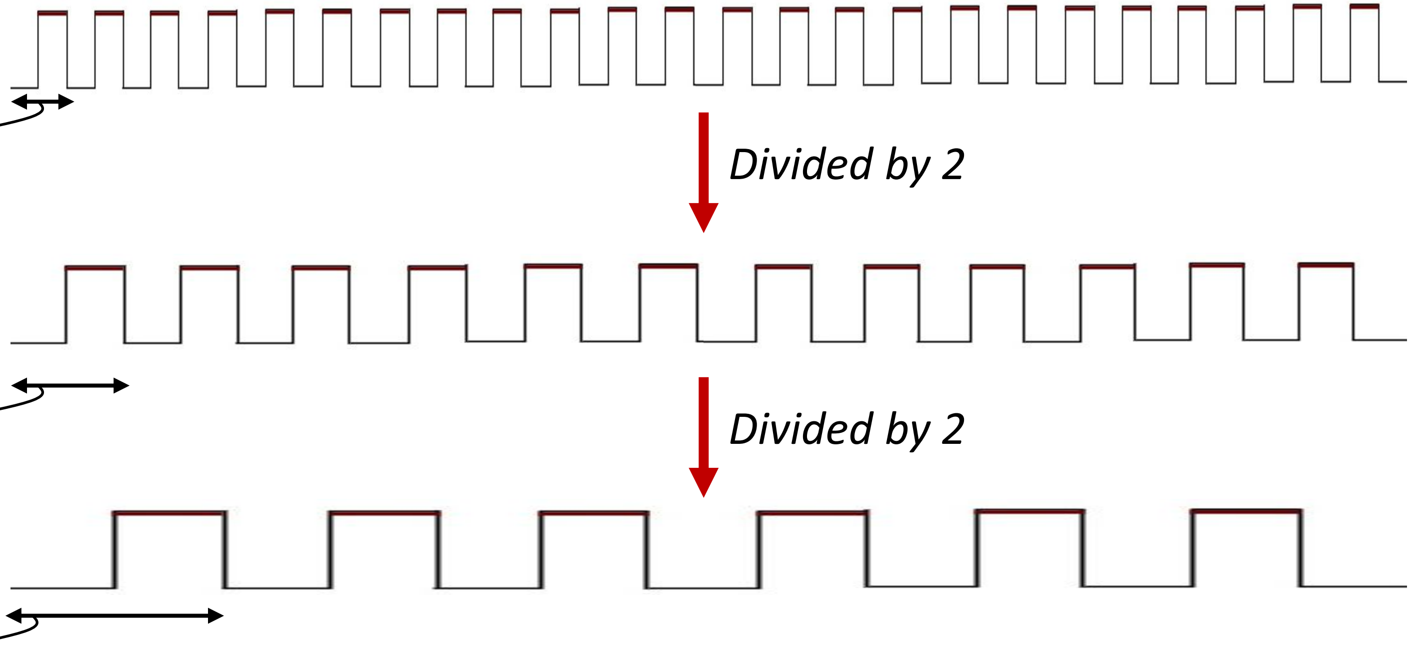
wake the microcontroller from a low-power state at a specific time or after a certain period.

*RTC typically runs on a separate, low-power clock source, i.e., **a 32.768 kHz crystal oscillator**, and can maintain time with very little energy, often powered by a small battery or capacitor.*

Why?

Features of Typical Microcontroller

- Most common peripherals
  - **Real-time clock (RTC)**



$$\frac{1}{2^{15}\ Hz} \approx 0.00003\ s$$

$$\frac{1}{2^{14}\ Hz} \approx 0.00006\ s$$

$$\frac{1}{2^{13}\ Hz} \approx 0.00012\ s$$

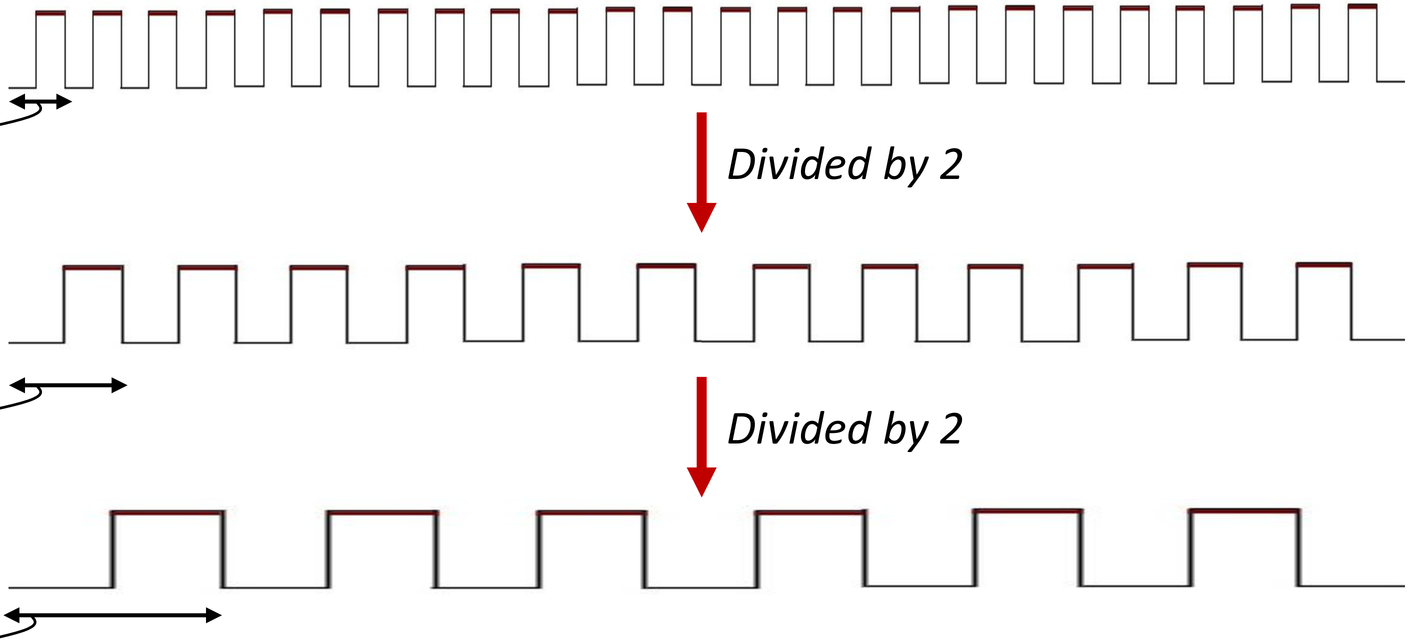
32.768 kHz = 32,768 Hz = 2^{15} Hz

RTC typically runs on a separate, low-power clock source, i.e., ***a 32.768 kHz crystal oscillator****, and can maintain time with very little energy, often powered by a small battery or capacitor.*

Features of Typical Microcontroller

- Most common peripherals
  - **Real-time clock (RTC)**

$$\frac{1}{2^{15}\ Hz} \approx 0.00003\ s$$



$$\frac{1}{2^{14}\ Hz} \approx 0.00006\ s$$

$$\frac{1}{2^{13}\ Hz} \approx 0.00012\ s$$

If we do this 15 times,
Frequency of output signal
Will be 1 Hz

Each clock period is 1 second!

32.768 kHz = 32,768 Hz = 2^{15} Hz

*RTC typically runs on a separate, low-power clock source, i.e., **a 32.768 kHz crystal oscillator**, and can maintain time with very little energy, often powered by a small battery or capacitor.*

Features of Typical Microcontroller

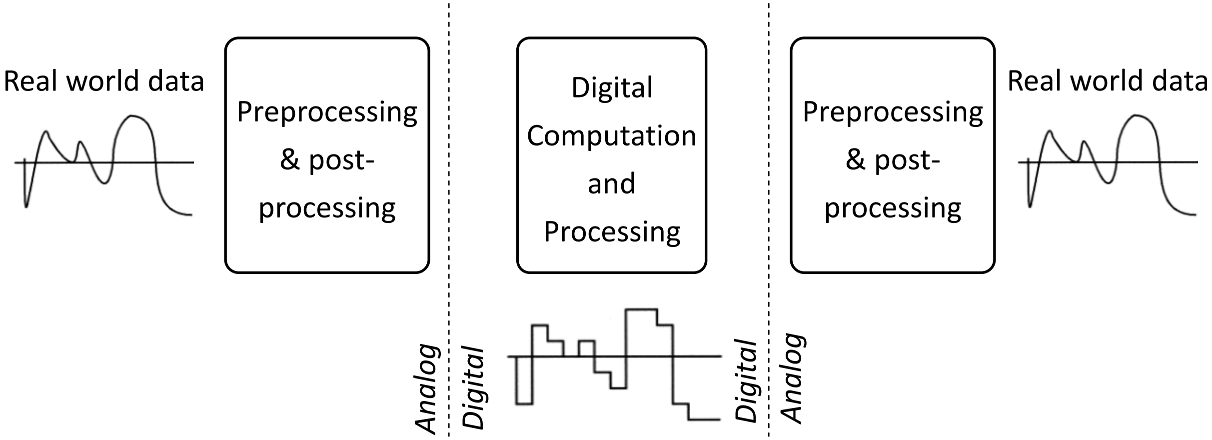
- Most common peripherals
  - Timer
    - Watchdog timer
  - Real-time clock (RTC)
  - **<u>Communication interfaces</u>**
  - Analog-to-Digital converter (ADC)
  - General purpose input/output (GPIO)

Microcontrollers support a wide choice of communication protocols:

- *Serial communication UART*
- *Serial peripheral interface (SPI)*
  - *Data transfer with storage such as SD cards*
- *Inter-integrated circuit (I2C)*
  - *Communication with temperature sensor*
- *Universal serial bus (USB)*
- *Controller area network (CAN)*
  - *Communication network in automobiles*

Features of Typical Microcontroller

- Most common peripherals
  - Timer
    - Watchdog timer
  - Real-time clock (RTC)
  - Communication interfaces
  - **Analog-to-Digital converter (ADC)**
  - General purpose input/output (GPIO)



Analog-to-digital converter (ADC)

- *Used to convert analog signals to digital values.*
- *e.g., a sensor's output varies between 0-5V. This is an analog signal. The ADC is then used to map the analog voltage to a digital value (0 - 65535).*

Digital-to-analog converter (DAC)

- *Used to convert digital values to analog signals.*
- *e.g., Playing digital audio through speakers.*
- *e.g., digital control signals into analog voltages that drive actuators like motors.*

Features of Typical Microcontroller

- Most common peripherals
  - Timer
    - Watchdog timer
  - Real-time clock (RTC)
  - Communication interfaces
  - Analog-to-Digital converter (ADC)
  - **General purpose input/output (GPIO)**

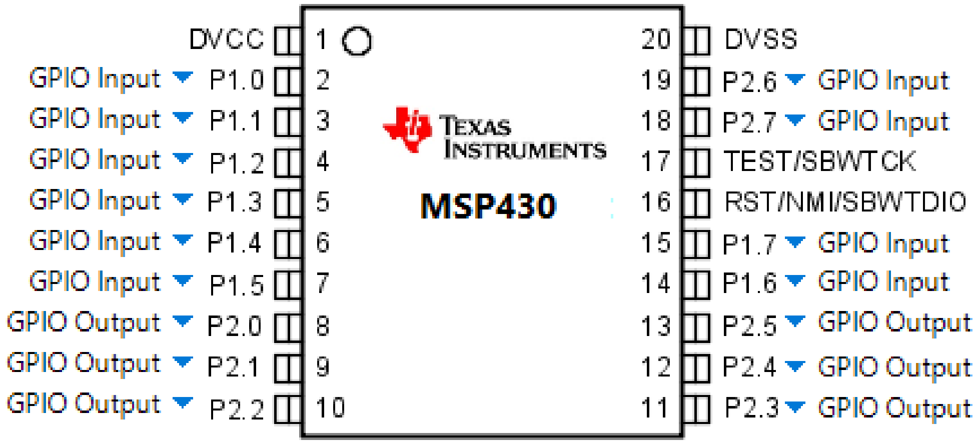
Microcontrollers can read or write values on its GPIO pins.

GPIO pins, typically are grouped into ports (P).

Each port has 8 pins.

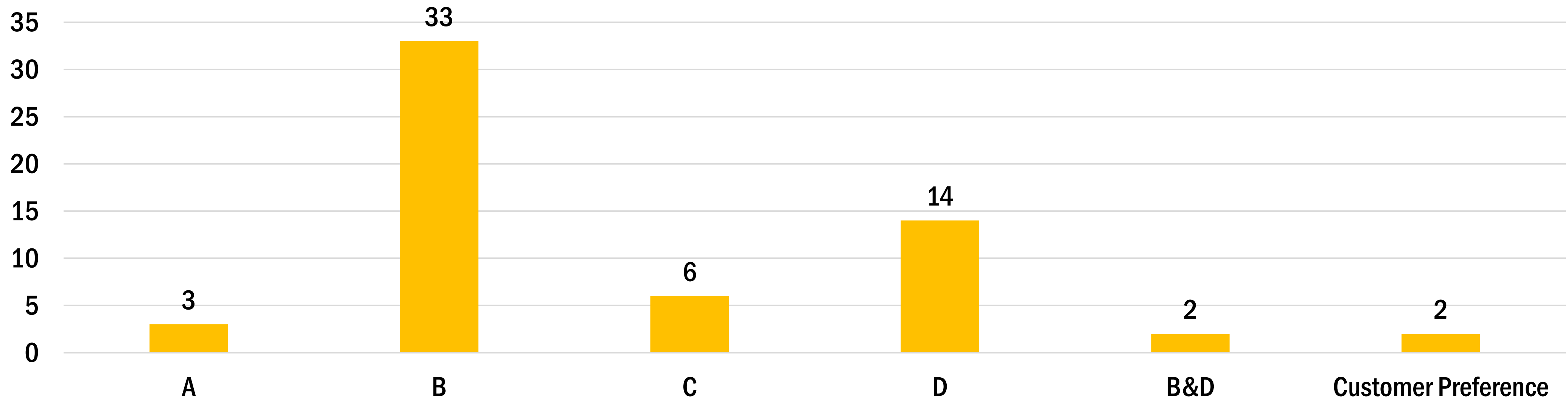
P1.0 denotes pin 0 in Port 1
P2.5 denotes pin 5 in Port 2

Port 1 (P1):
Commonly used for basic I/O operations, such as reading buttons or controlling LEDs. Supports interrupt functionality, which allows Port 1 pins to trigger an interrupt service routine (ISR) on a specified edge (rising or falling).
Port 2 (P2):
Similar to Port 1 in functionality, with 8 pins that can be individually configured. Also supports interrupts, making it useful for handling multiple external inputs that require attention from the microcontroller.



Survey-like Question

- A microcontroller that monitor and adjust home temperature, control the lighting, and manage security sensors in real-time. Most critical factor at designing?
- A) Accuracy for temperature. B) Response time to a security breach.
- C) Complexity of the lighting control. D) Energy consumption of the microcontroller.



Survey-like Question

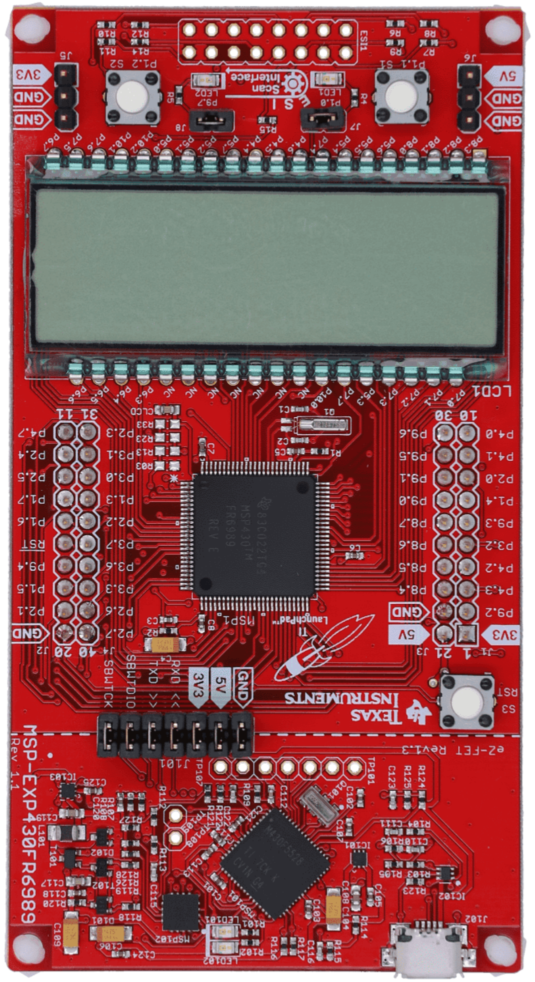
- A microcontroller that monitor and adjust home temperature, control the lighting, and manage security sensors in real-time. Most critical factor at designing?
- A) Accuracy for temperature. B) Response time to a security breach.
- C) Complexity of the lighting control. D) Energy consumption of the microcontroller.

- ChatGPT?!?
  - Splitting the controller >>> one for security (safety-critical) and one for the rest
    - Isolation of security
    - Fault containment
    - Simpler software design
    - Scalability

MSP430 Launchpad

- MSP430 General Info
  - Used in the lab: MSP430FR6989
  - MSP: Mixed Signal Processor (analog & digital)
  - ‘F’ or ‘G’: Type of memory (Flash)
  - ‘FR’: FRAM (ferroelectric nonvolatile RAM)
    - For Ultra low power computation/processing
  - ‘FG’: Flash medical
  - Number 6989 starts with ‘6’ means it belongs to the FR6xx series
    - Different versions with different configurations

  - The Basic Launchpad’s chip: MSP430G2553
  - Number 2553 starts with ‘2’ means it belongs to the x2xx series

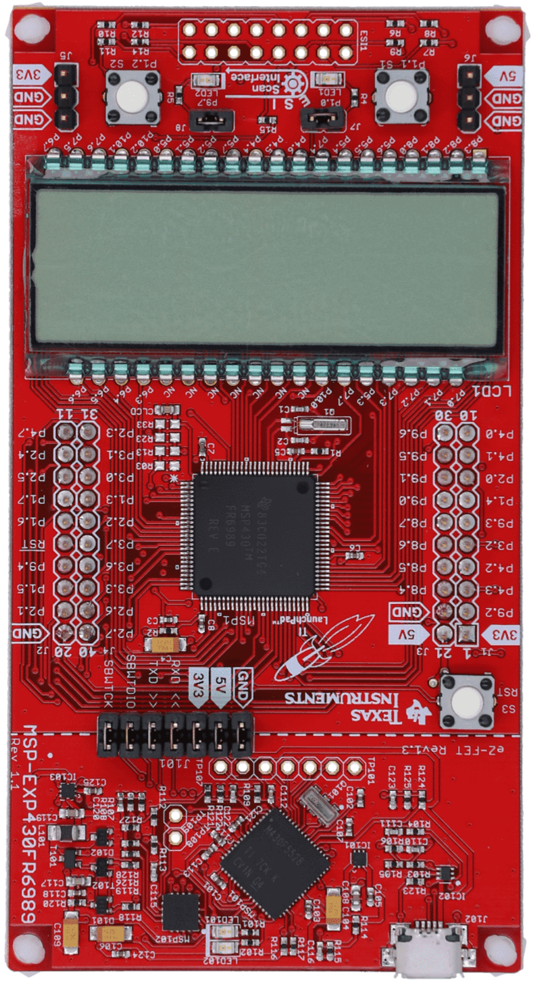


MSP430FR6989 development board

MSP430 Launchpad

- MSP430 General Info
  - Core: MSP430 16-bit RISC architecture.
  - Memory
    - Most MSP430 have a 16-bit address
      - up to 2^{16} = 64 KB memory
    - MSP430X variant has a 20-bit address
      - up to 2^{20} = 1 MB memory

    Different versions with different memory sizes:
    128 KB of non-volatile FRAM (Ferroelectric RAM), 2 KB of SRAM.
  - Clock Frequency: up to 16-MHz Clock
  - Peripherals
    - 12-bit ADC.
    - Comparators, Op-Amps, and DACs.
    - Serial communication interfaces including I2C, SPI, and UART.



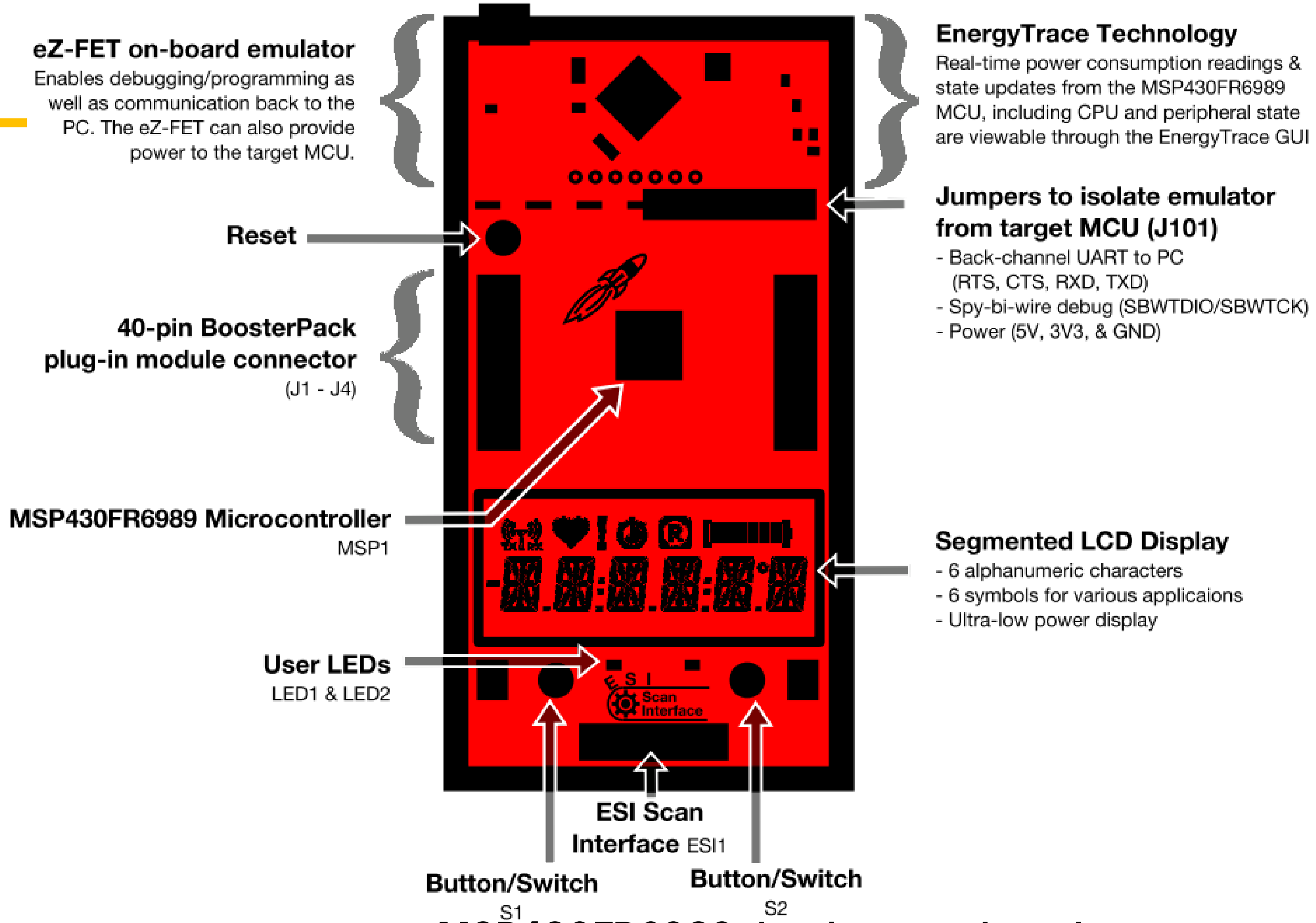
MSP430FR6989 development board

MSP430 Launchpad

- MSP430 features

USB Connectivity

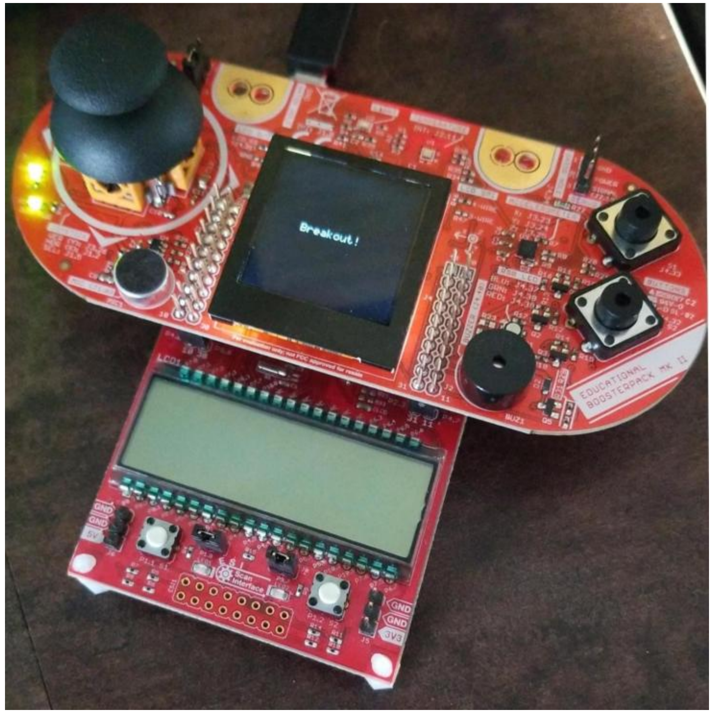
- *Power to the MSP430 board*
- *Communication link for programming*
- *On-board debugging*
- *Firmware upgradability*



MSP430FR6989 development board

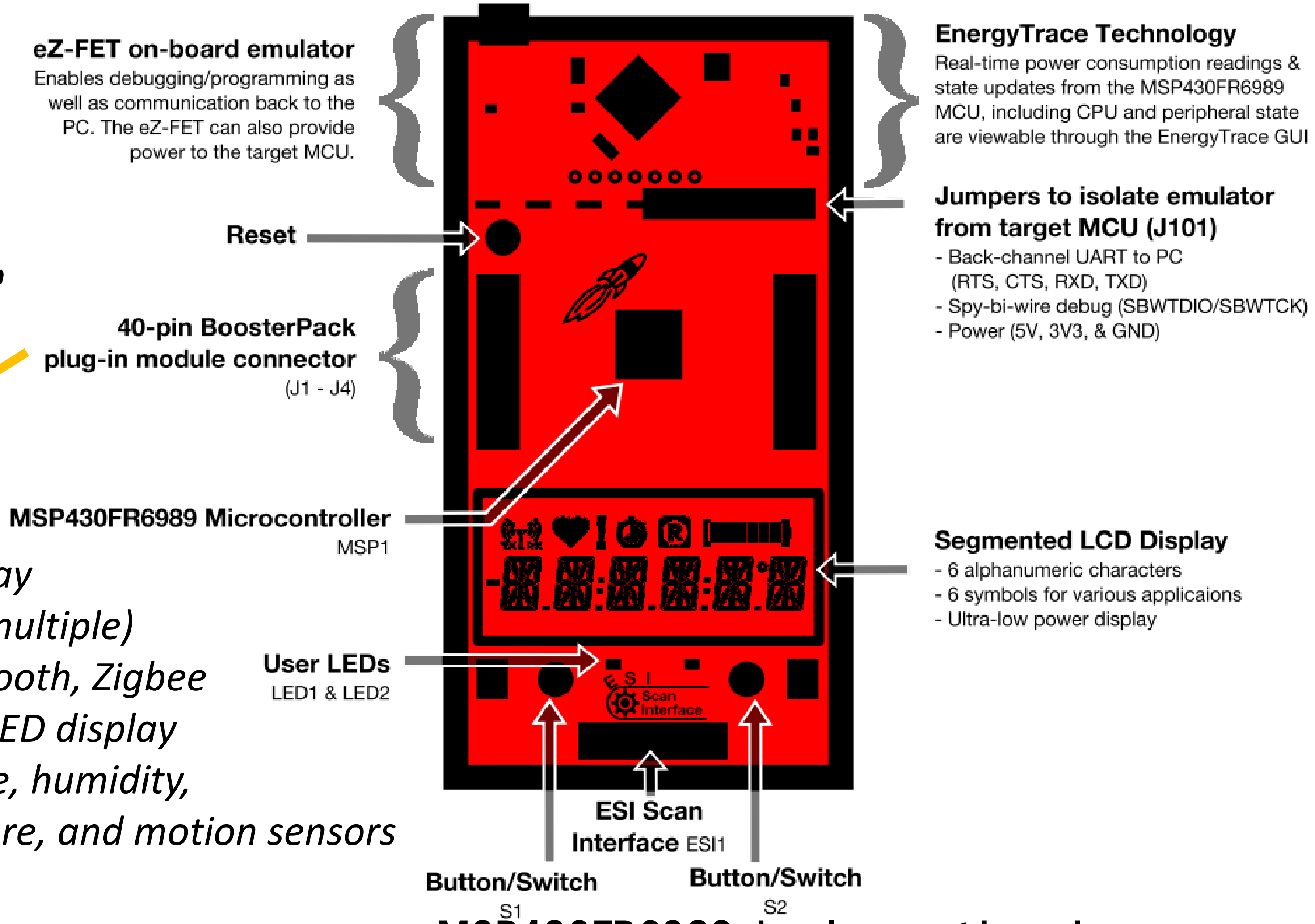
MSP430 Launchpad

- MSP430 features



Sensors, display, wireless, etc.

- *Plug-and-play*
- *Stackable (multiple)*
- *Wi-Fi, Bluetooth, Zigbee*
- *LCD and OLED display*
- *temperature, humidity, light, pressure, and motion sensors*

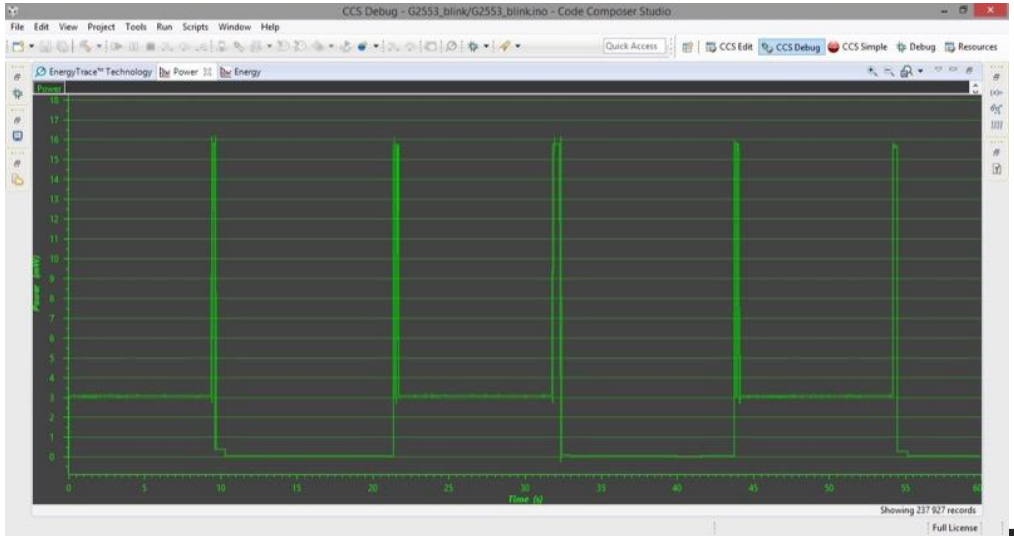


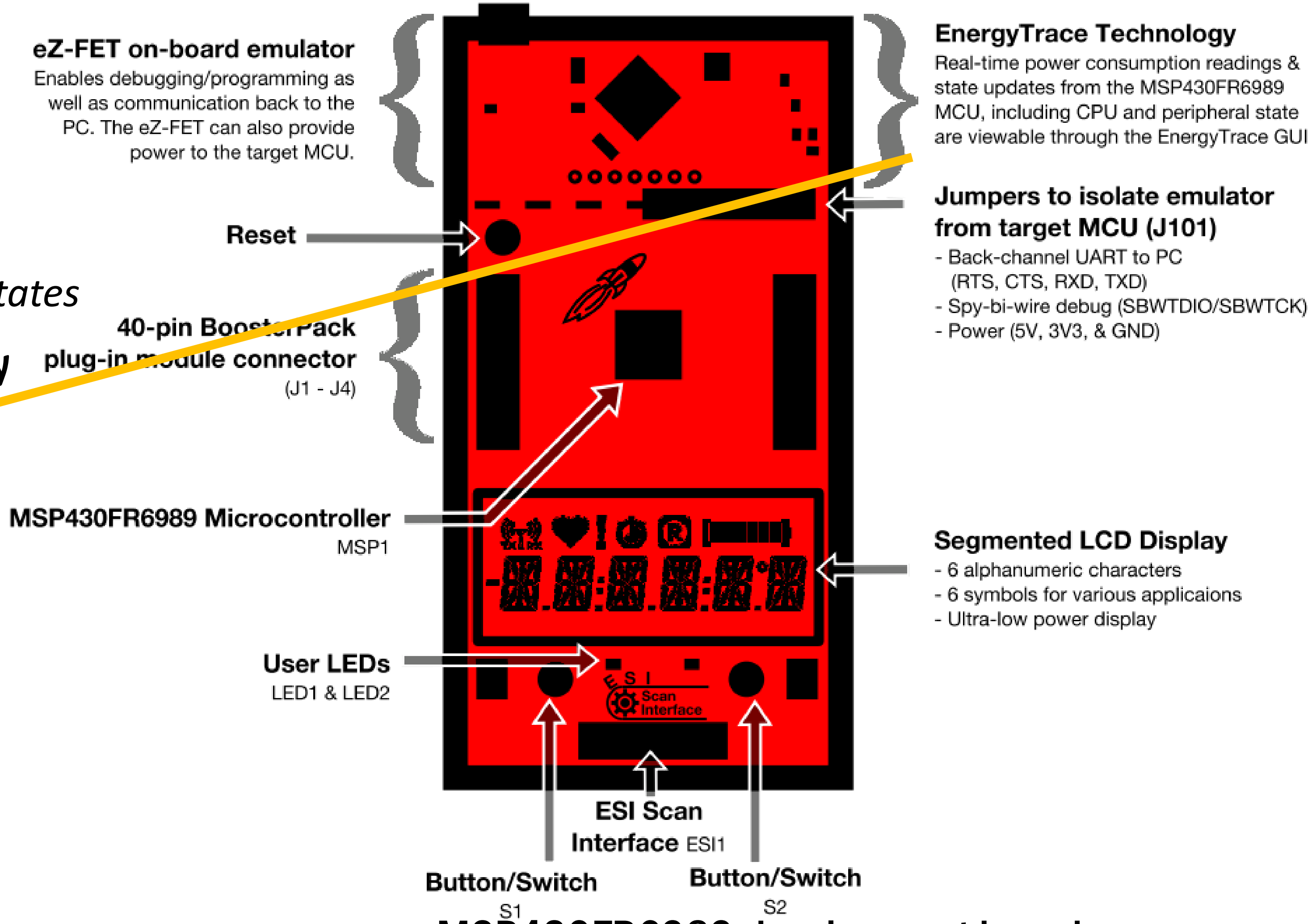
MSP430FR6989 development board

MSP430 Launchpad

- **MSP430 features**
  - *Through eZ-FET component*
  - *Based on voltage drop on resistors*
  - *Current calculation down to nanoAmp*
  - *EnergyTrace, with current monitoring,*
  - *EnergyTrace+, with current monitoring and CPU states*
  - *EnergyTrace++ , with current monitoring CPU states and peripheral states.*

Tracking Energy





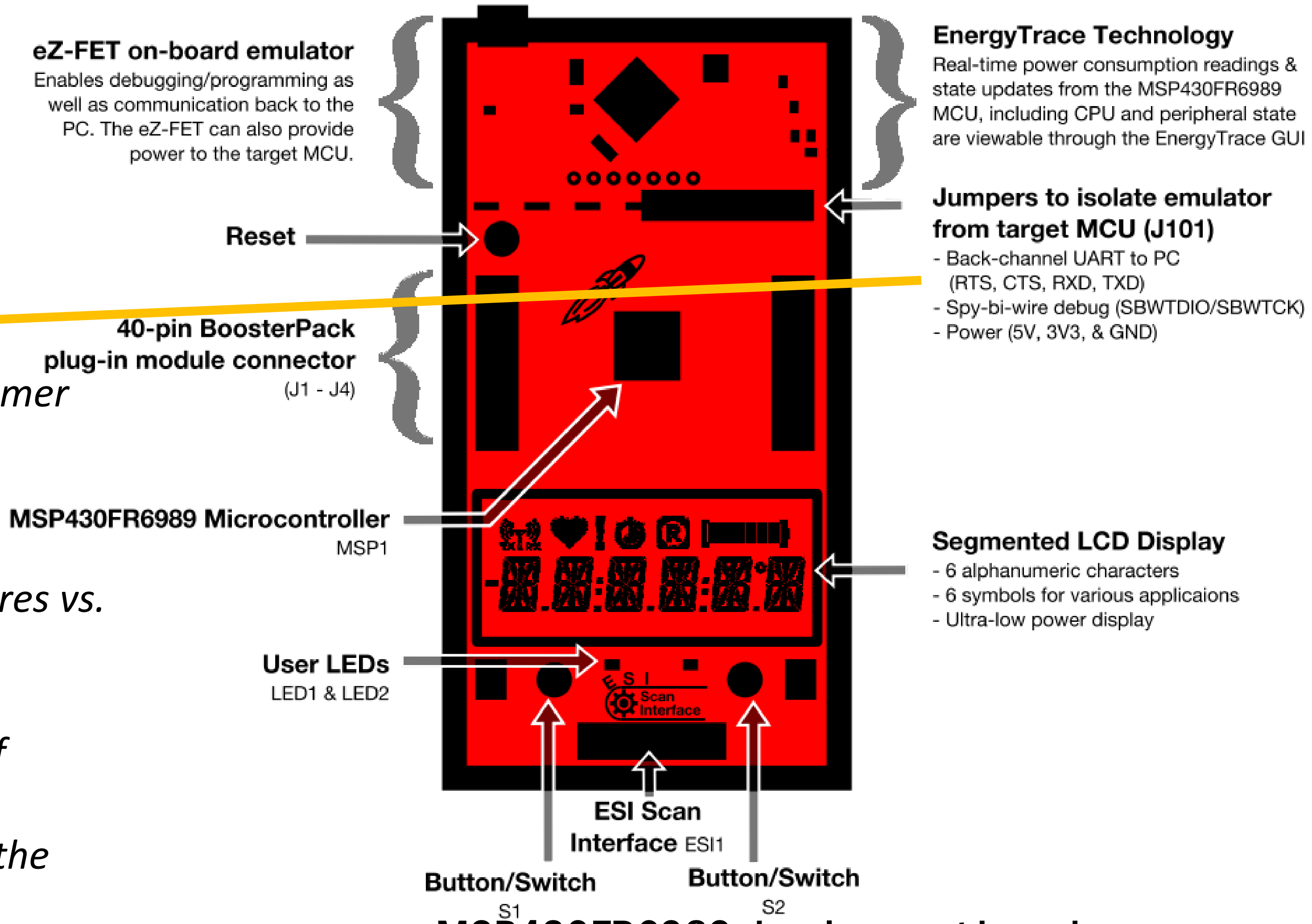
MSP430FR6989 development board

MSP430 Launchpad

- MSP430 features

Debugging, power, communication, peripheral control

- *(jumper) on-board or external debugger/programmer*
- *(jumper) Connecting/disconnecting peripherals*
- *(jumper) enabling/disabling EnergyTrace (or use of external power analysis*
- *(Spy) alternative to the traditional JTAG (only 2 wires vs. JTAG that requires 4 to 5 wires*
- *(Spy) Compatible with the existing JTAG interface*
- *(Spy) can be used to program the flash memory of MSP430 microcontrollers*
- *Integrated with Code Composer Studio (CCS) and the eZ-FET debugger found on many MSP430*



MSP430FR6989 development board

MSP430 Launchpad

- MSP430 features

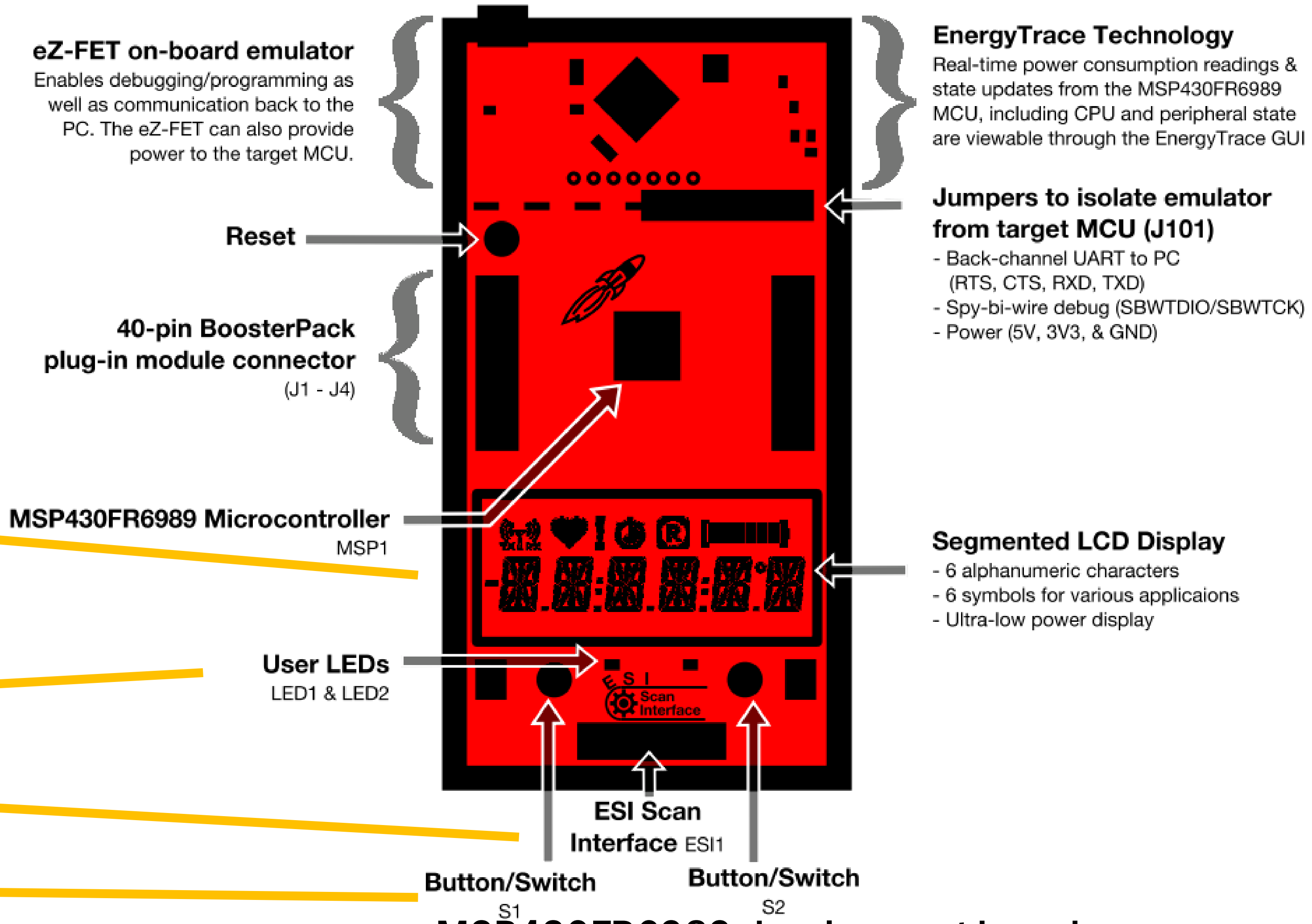
- *part of TI's EnergyTrace™ Technology*
- *scan and measure the power consumption of the MSP430 microcontroller*
- *Integrated with TI's Code Composer Studio (CCS)*

Segmented LCD display (ultra low power) general-purpose outputs

General-purpose outputs

Energy scanning interface

General-purpose inputs

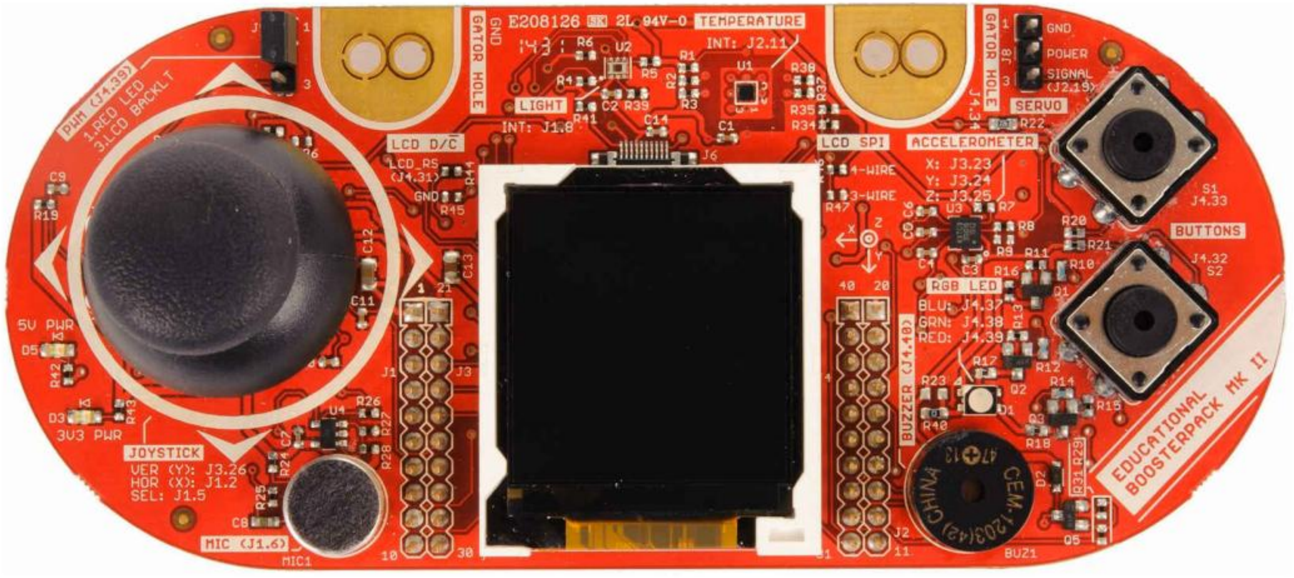


MSP430FR6989 development board

BOOSTXL-EDUMKII BoosterPack

- Bunch of sensors and peripherals allowing one to get benefit from them while using MSP430 launchpad
  - Temperature Sensor
  - Light Sensor
  - Accelerometer
  - LCD Display

  - Capacitive Touch Buttons
  - Push Buttons and LEDs
  - Joystick

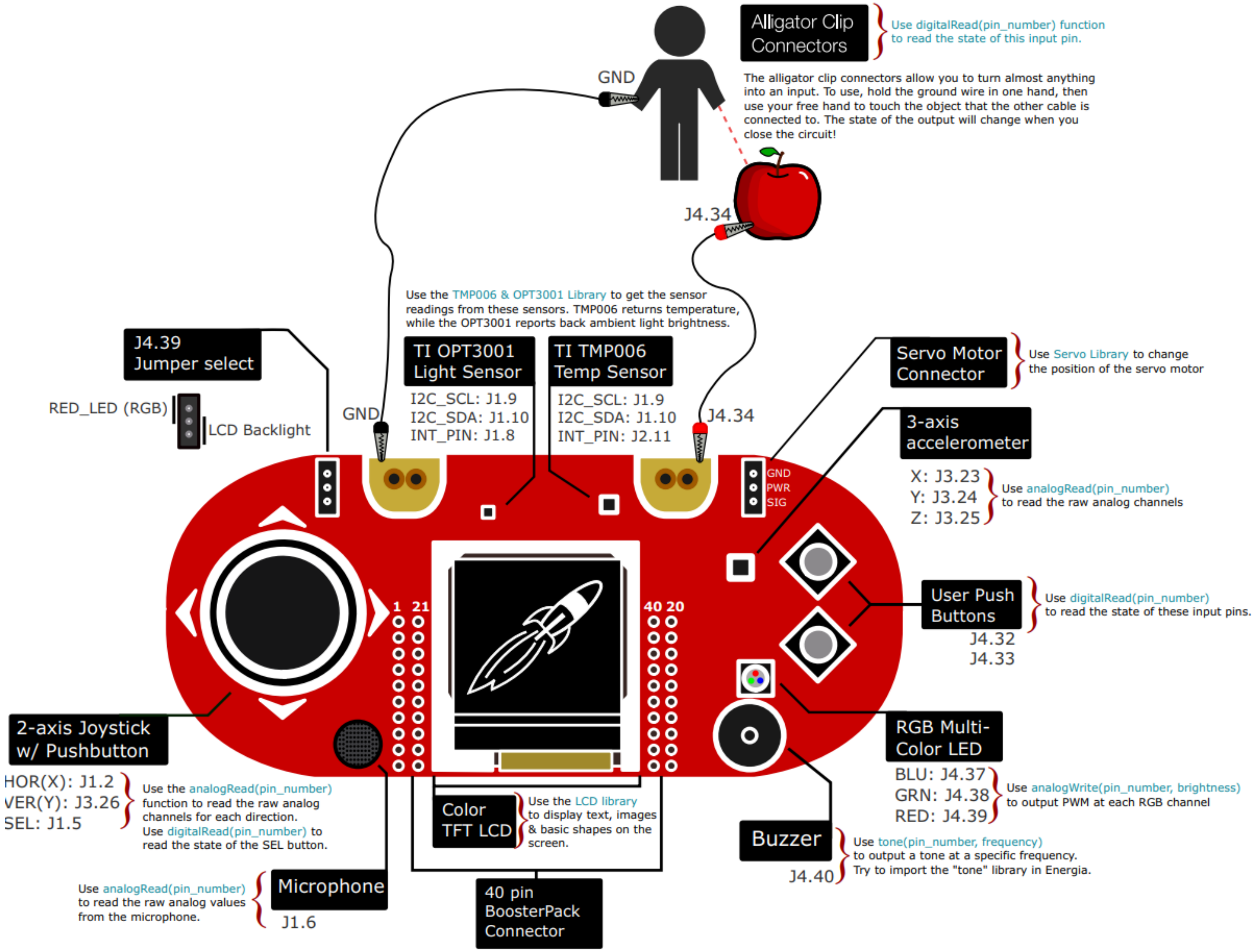


BOOSTXL-EDUMKII BoosterPack Plug-in Module

BOOSTXL-EDUMKII BoosterPack

- Bunch of sensors and peripherals allowing one to get benefit from them while using MSP430 launchpad
  - Temperature Sensor
  - Light Sensor
  - Accelerometer
  - LCD Display

  - Capacitive Touch Buttons
  - Push Buttons and LEDs
  - Joystick



Documentation (How to find)

- Where do I find the information for each question below?

- I’m using the G2553 chip, I want to know how the timer works??

- I have a chip, I heard that it could have multiple independent timers, how many timers does it have??

- Would the chip burn out if it receives a signal of 6V??

- I’m using a board, to which pin is the LED connected??

Family user’s guide? *Chip’s data sheet?* *LaunchPad board’s datasheet?*

Documentation (How to find)

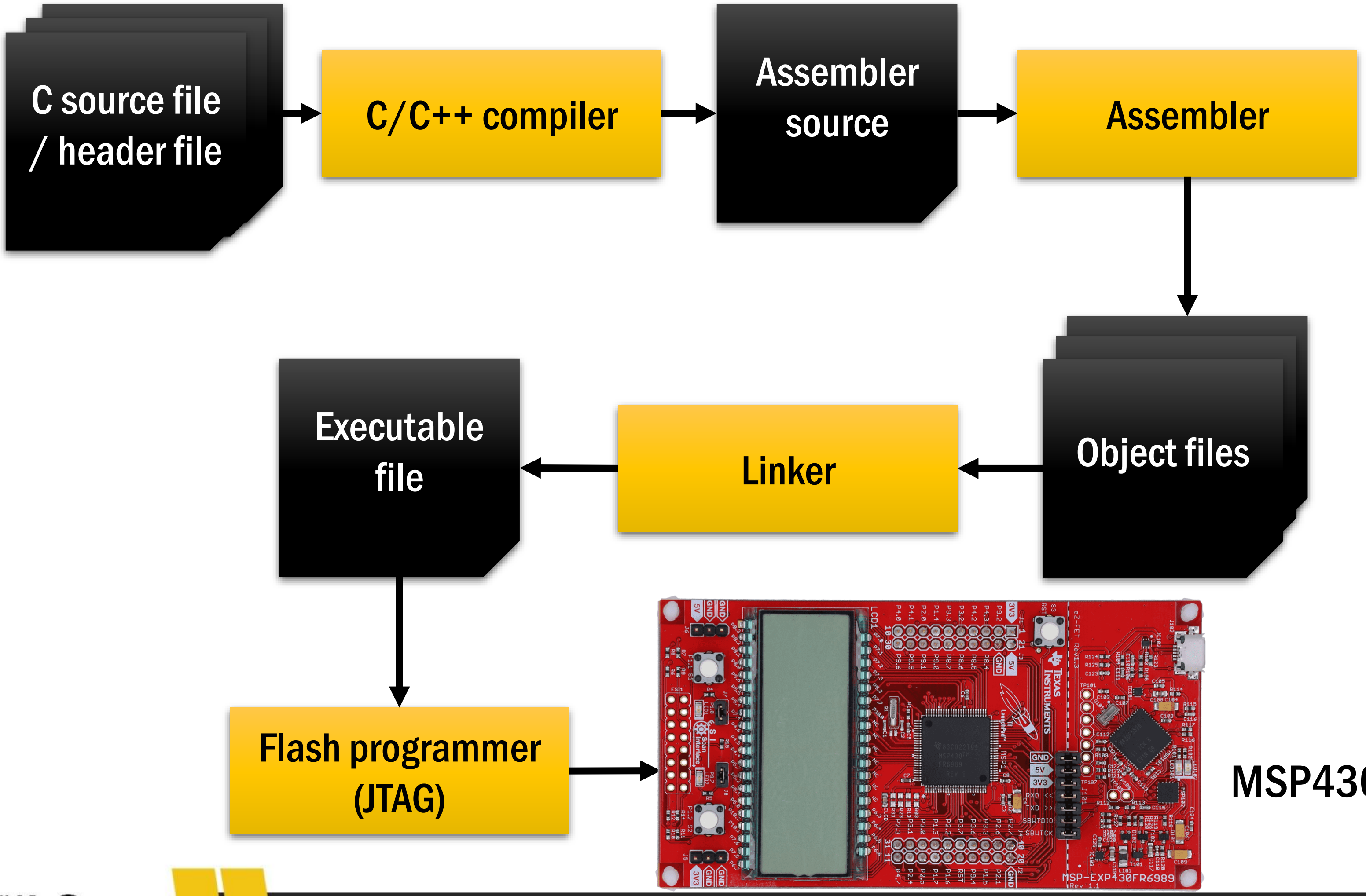
- Where do I find the information for each question below?

- I'm using the G2553 chip, I want to know how the timer works??
  - Look in the x2xx Family User's Guide. All the x2xx chips have the same timer (described in the common doc).
- I have a chip, I heard that it could have multiple independent timers, how many timers does it have??
  - Look in the chip's data sheet, a chip may have a few independent timers to make it more versatile to use.
- Would the chip burn out if it receives a signal of 6V??
  - Look in the chip's data sheet, a chip may have a few independent timers to make it more versatile to use.
- I'm using a board, to which pin is the LED connected??
  - Look in the evaluation board's data sheet.

Family user's guide? *Chip's data sheet?* *LaunchPad board's datasheet?*

Flow of Development in MSP430

- Development Process in MSP430



MSP430FR6989 development board

Flow of Development in MSP430

- Development Process in MSP430
  - Editor for developing the codes.

```c
#include <msp430x11x1.h>

void main(void){
    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer
    P2DIR = 0x18;               // Set P2.3, P2.4 in output mode
    P2OUT = 0x08;               // Set P2.3 High and P2.4 Low
    for(;;){                    // Infinite Loop
    }
}
```

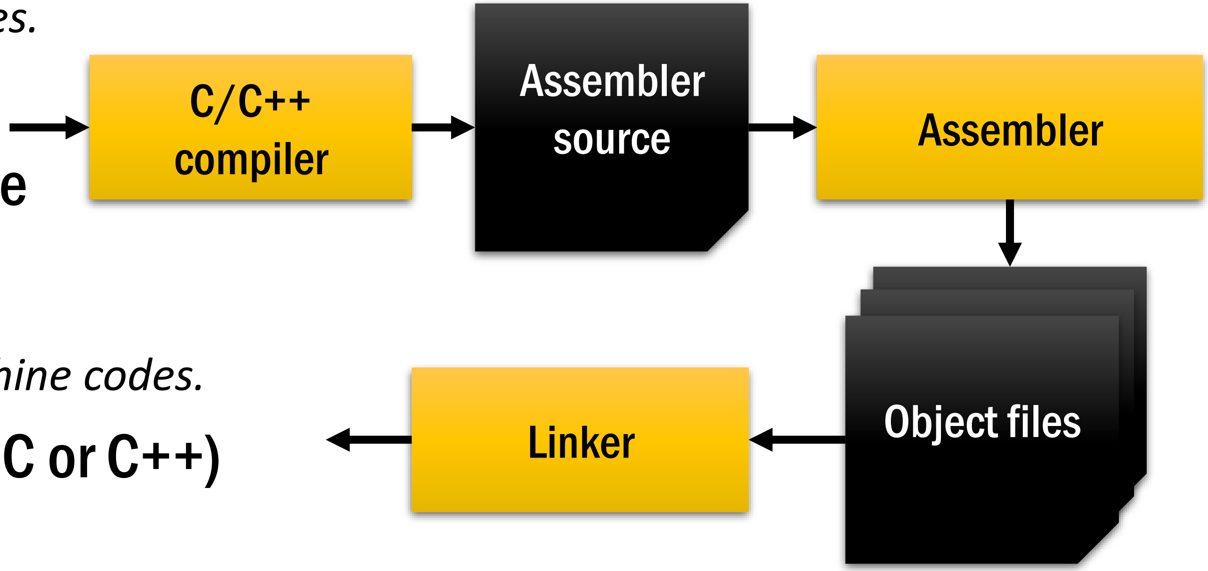
C source file / header file

Each chip has a zip file of code examples posted on TI's webpage of that chip!

  - Used to write and edit programs.
  - syntax highlighting, auto indenting, ability to search definitions in header files are some features that are expected of an ideal editor

Flow of Development in MSP430

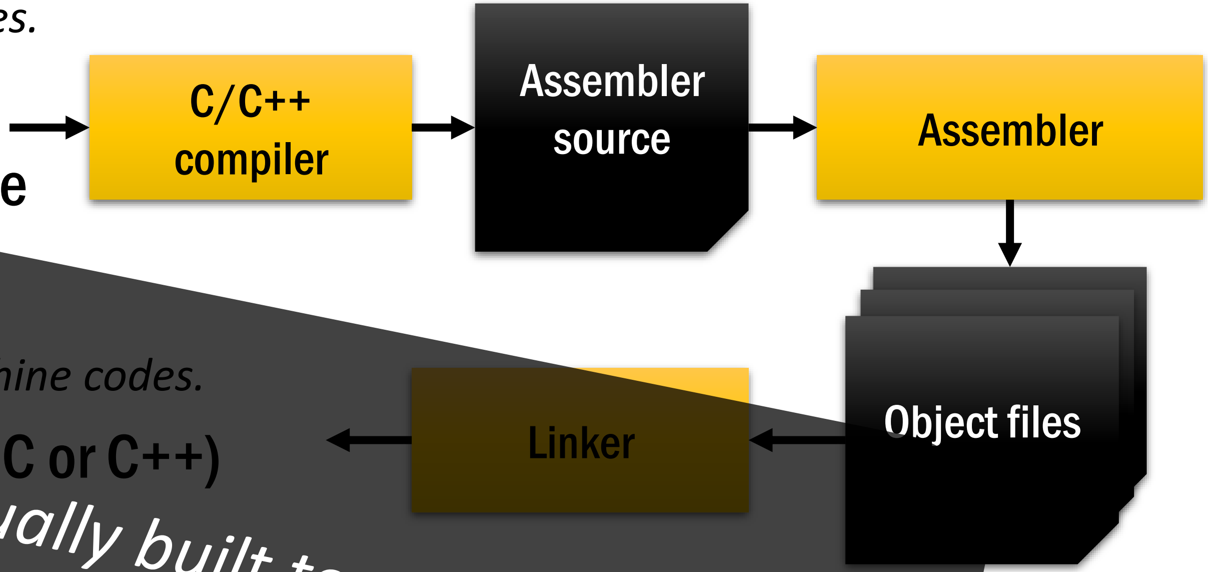
- **Compiling** *Translates high-level code to assembly languages.*
  - Checks errors
  - Compiles the code and produces the executable file
  - Has the capability to optimize the code



- **Assembling** *Converts assembly languages to objects of machine codes.*
  - translates high-level programming languages (like C or C++) into assembly code or machine code
  - understands and processes the instruction set of the MSP430 microcontroller
  - generates object files containing the machine code, along with additional data like symbol tables and relocation information.
- **Linking** *Converts objects of machine codes into executable.* ***reflects the memory map***
  - combines multiple object files (which are the output of the assembler) into a single executable file

Flow of Development in MSP430

- **Compiling** *Translates high-level code to assembly languages.*
  - Checks errors
  - Compiles the code and produces the executable file
  - Has the capability to optimize the code

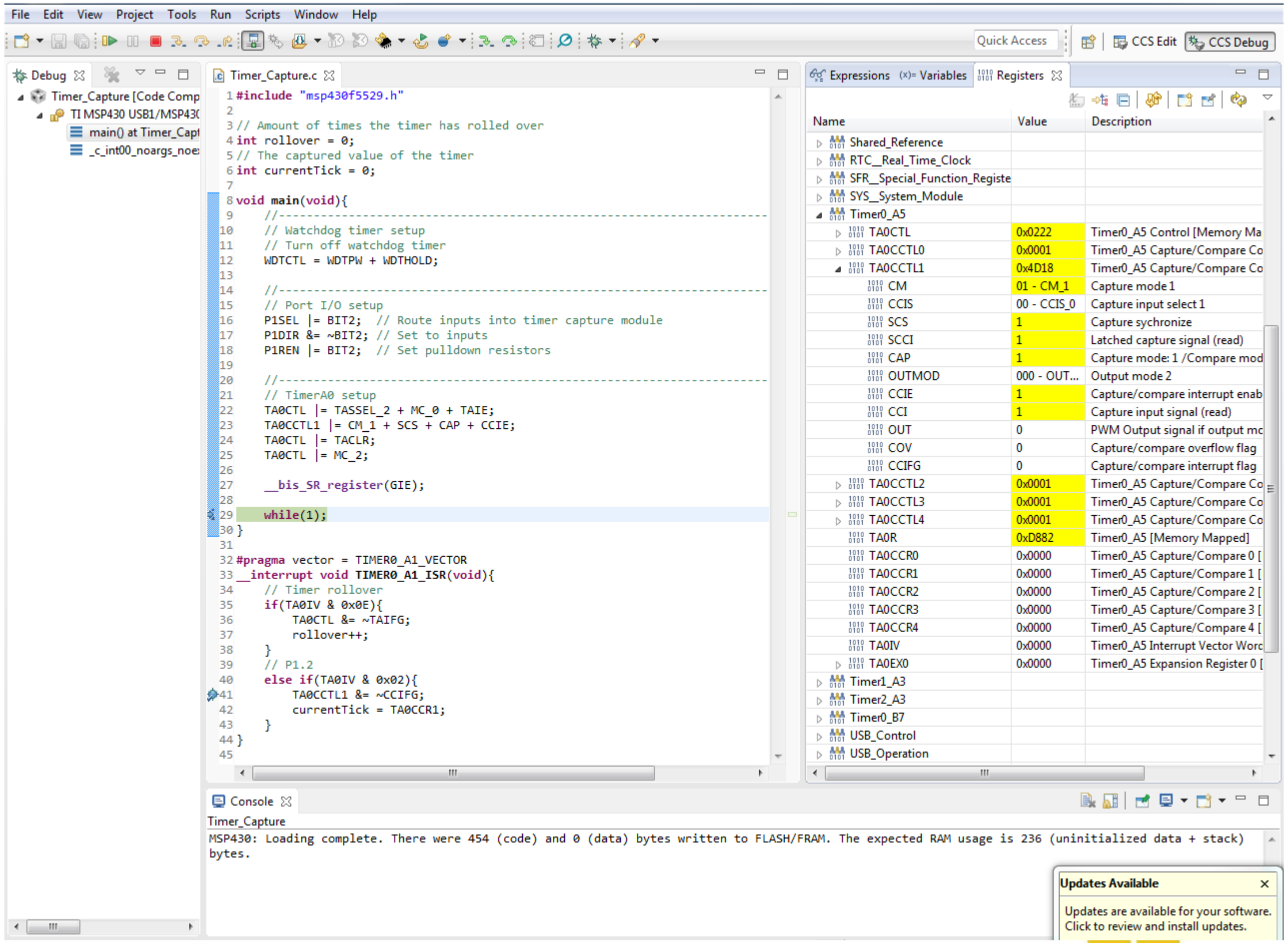


- **Assembling** *Converts assembly languages to objects of machine codes.*
  - translates high-level programming languages (like C or C++) into assembly code or machine code
  - understands and processes the instruction set of the MSP430 microcontroller
  - generates object files containing the machine code, along with additional data like symbol tables and relocation information.
- **Linking** *Converts objects of machine codes into executable.*
  - combines multiple object files (which are the output of the assembler) into a single executable file

*All the above three are usually built together in the **“integrated development environment”** (IDE).*

Code Composer Studio (CCS)

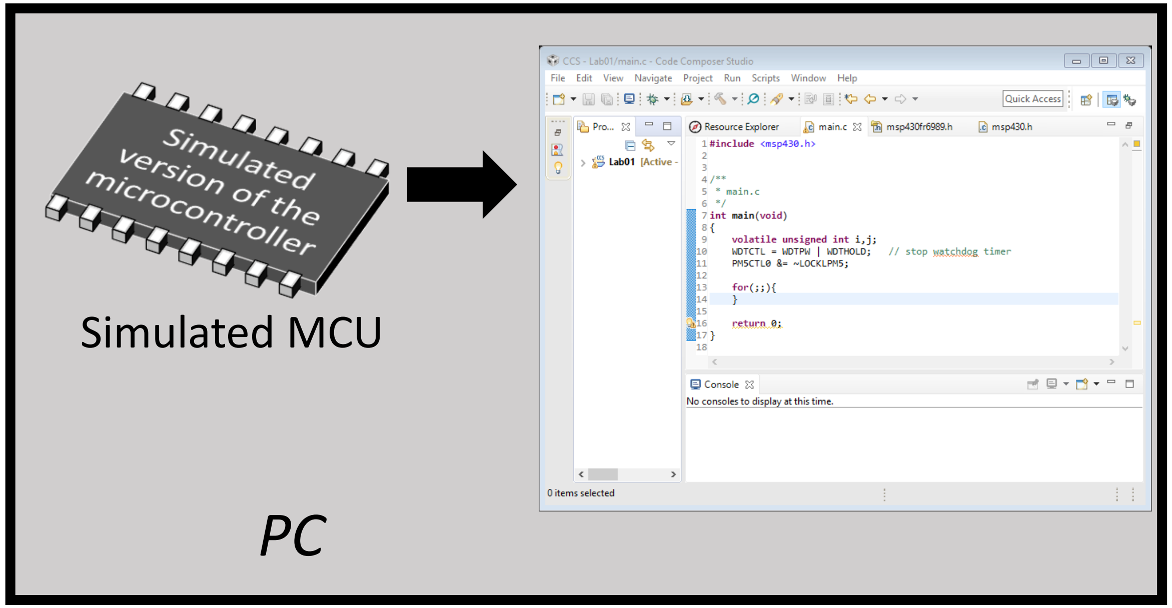
- The name of IDE for MSP430
  - Code Composer Studio (CCS)
  - provided by the manufacturer, Texas Instruments
  - Compiler
  - Assembler
  - Linker
  - Debug using eZ-FET
  - Real time EnergyTrace

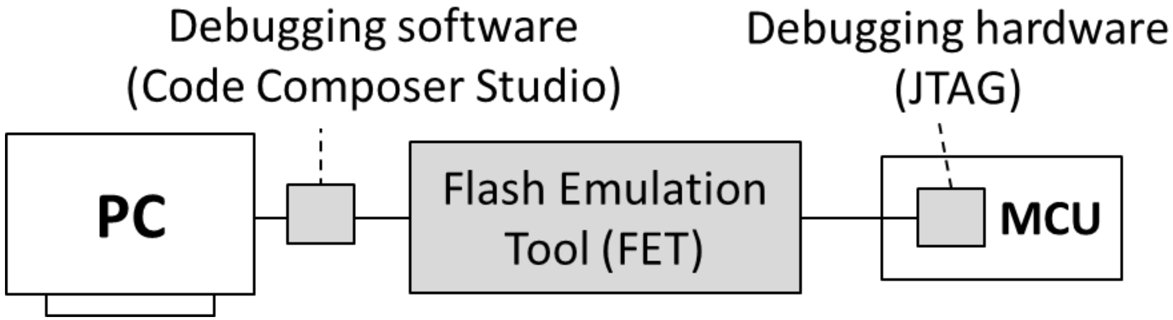


Emulation and Debugging

- the execution of the program on the MCU on a PC
- Debugging directly from the HOST PC
- The debugging software and hardware work together
- The FET is an intermediary since PCs usually don't support JTAG directly
- No simulation → it's emulation

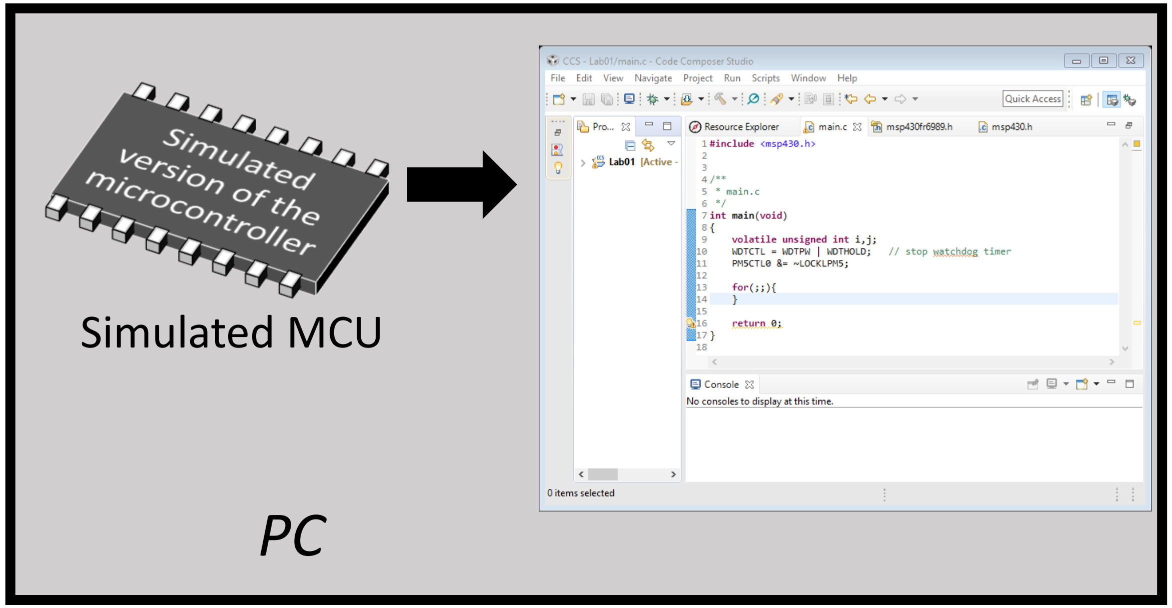
runs on actual hardware
(Through JTAG)

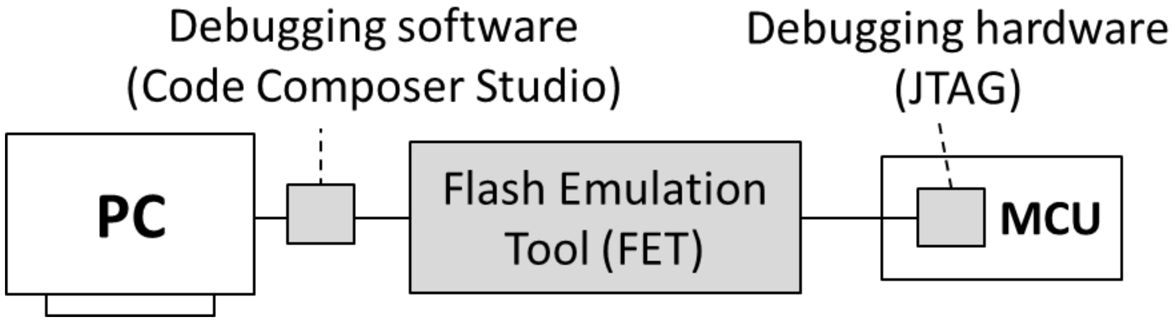




Emulation and Debugging

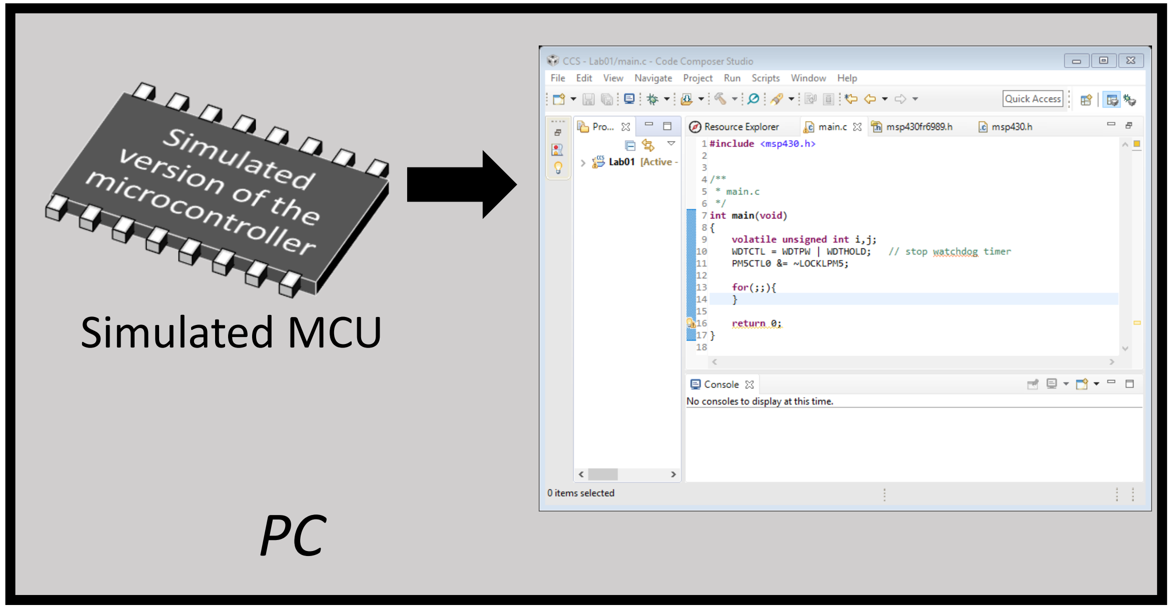
- Joint Task Action Group (JTAG)
  - It’s a hardware module on the chip used for programming and debugging
  - Standard JTAG has 4 pins (known as 4-wire JTAG)
  - Simpler version is 2-wire JTAG; known as Spy-Bi-Wire (SBW); uses fewer pins but is slower!
  - JTAG provides a set of commands that read/write to/from memory/CPU
  - Supports programming and debugging

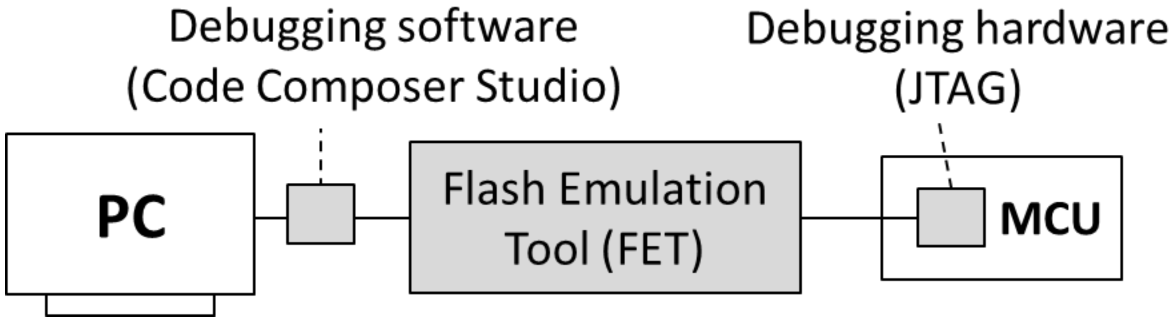




Emulation and Debugging

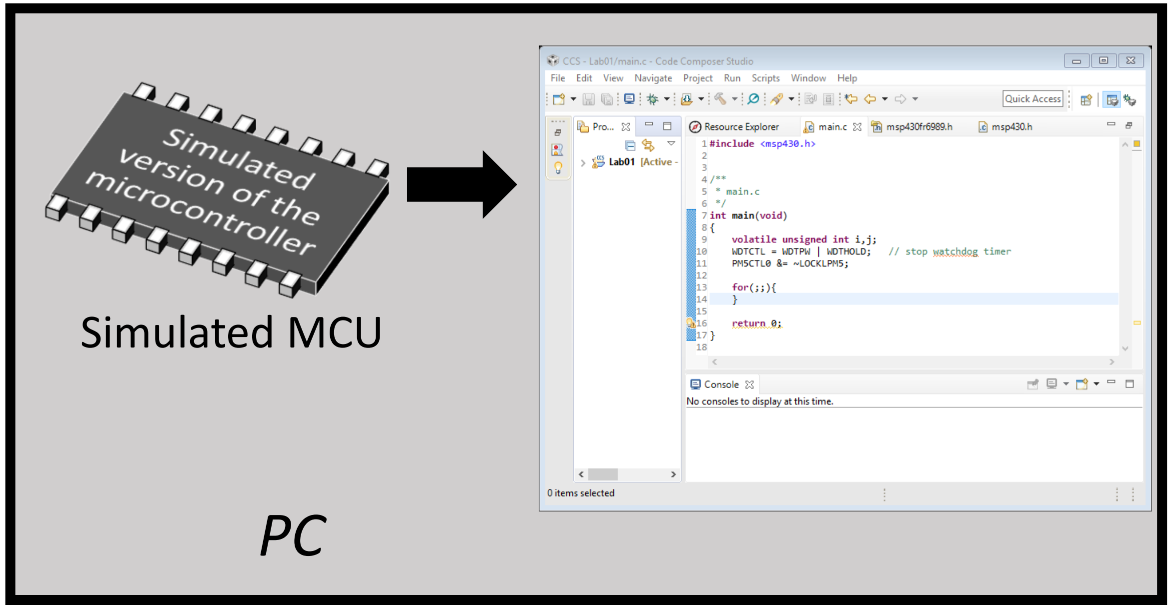
- Joint Task Action Group (JTAG)
  - Is JTAG a security vulnerability?

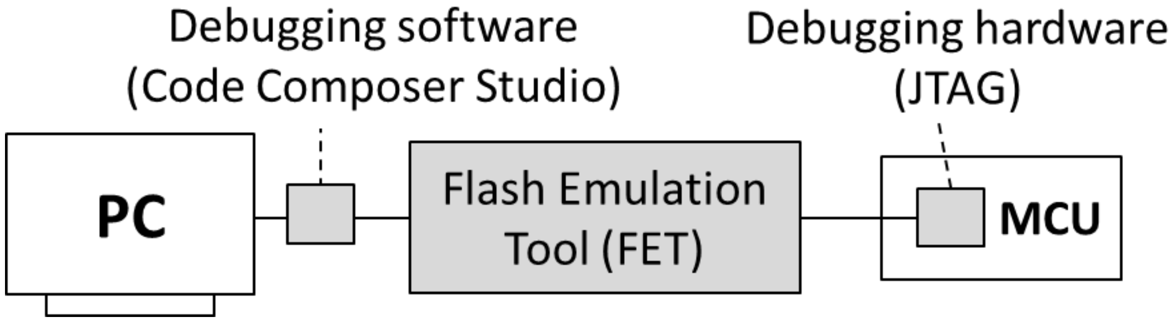




Emulation and Debugging

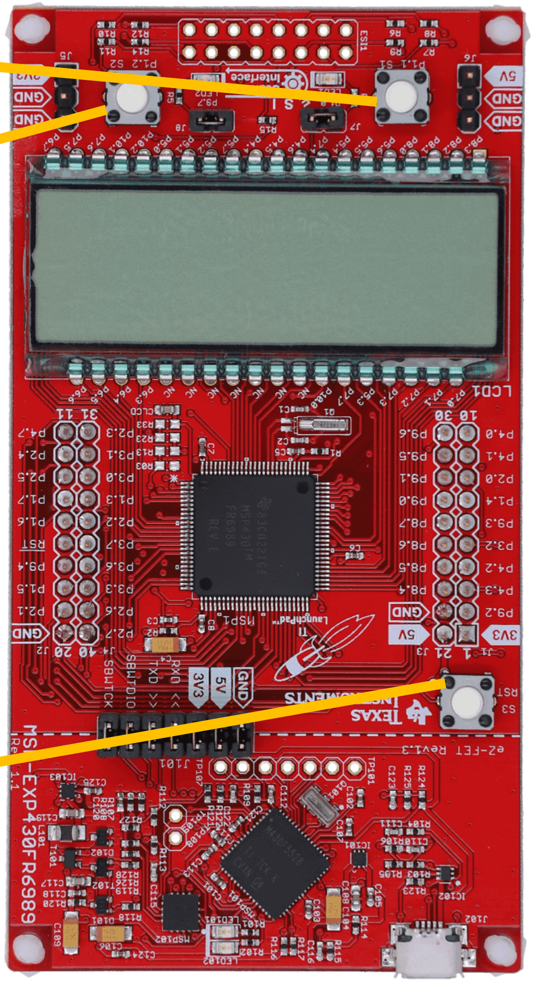
- Joint Task Action Group (JTAG)
  - Is JTAG a security vulnerability?
  - Yes. An intruder can tap to the JTAG pins and put the chip in debug mode
  - They can steal the code or data, inject code or modify the data
  - JTAG has a security fuse that can be blown after programming is done
  - This disables JTAG permanently before the product is deployed





Push Buttons for Lab 2

- **S1: Button connected to Port 1.1**
  - Port 1, Pin 1 (P1.1)
  - It usually pulls the pin low (active low) (i.e., it connects it to ground)
- **S2: Button connected to Port 1.2**
  - Port 1, Pin 2 (P1.1)
  - It usually pulls the pin low (active low) (i.e., it connects it to ground)

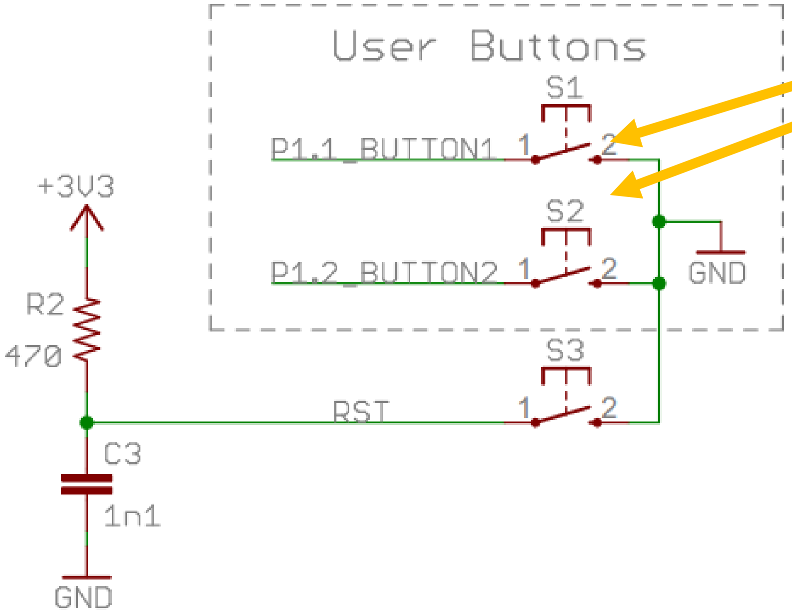


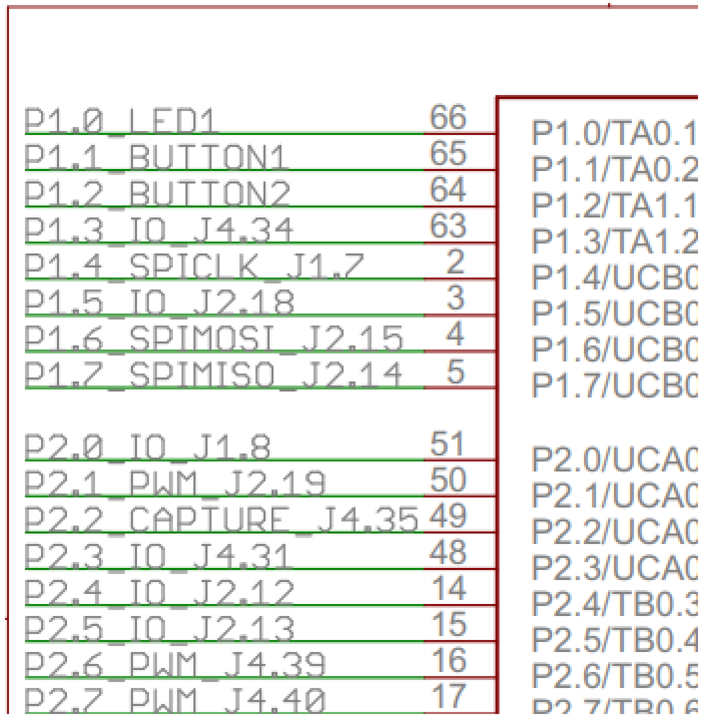
S1 and S2 often used for tasks like resetting a counter, toggling an LED, or triggering an interrupt service routine (ISR).

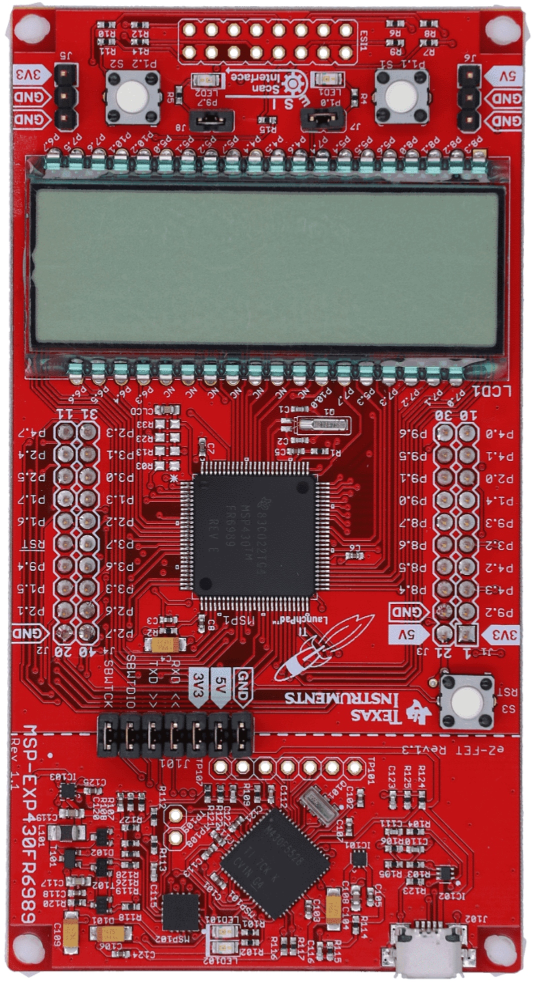
- **S3: Button connected to the reset (RST) pin**

Push Buttons for Lab 2

- All three with common RC network

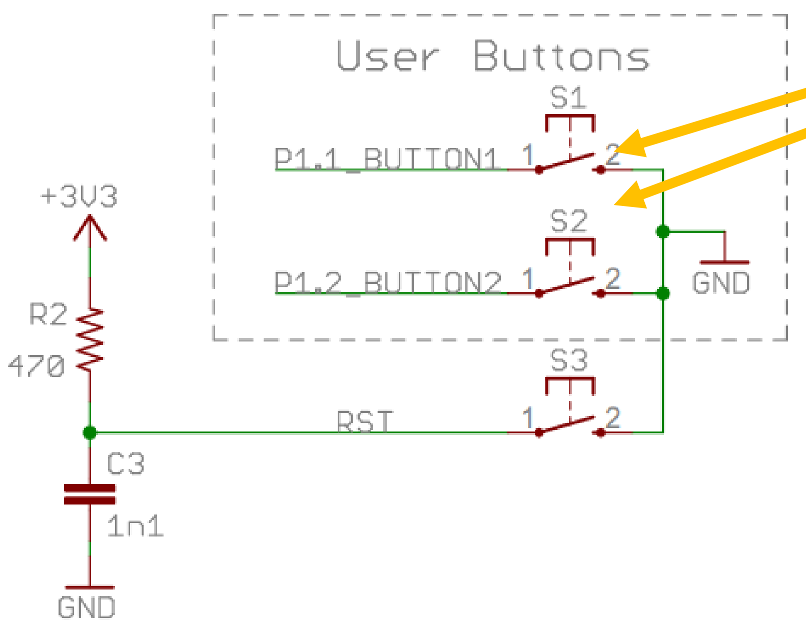


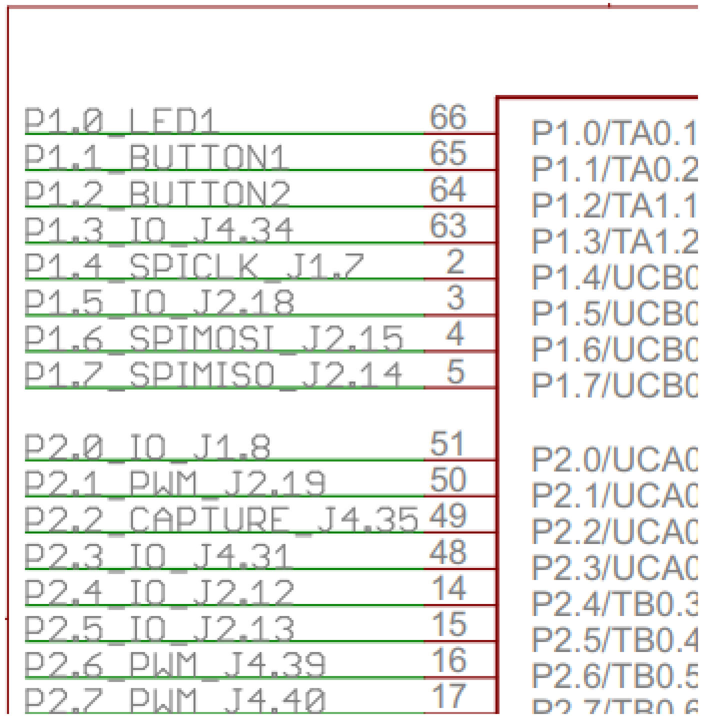


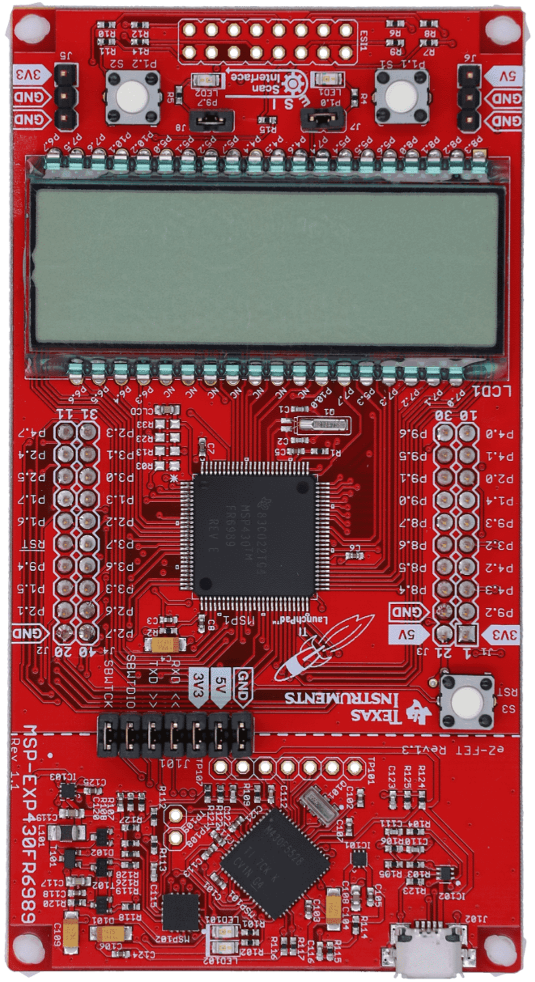


Push Buttons for Lab 2

- Connected to GND for Active low mode.







P1DIR, P1IN and P1OUT → To work with P1

P1REN → To enable/disable resistor for pull-up/pull-down network.

Once port is an input, P1OUT is used to configure the resistor as pull-up or pull-down.

Push Buttons for Lab 2

- **Writing a code that turns the RED LED when button S1 pushed.**

```
#include <msp430fr6989.h>
#define redLED BIT0                          // Red LED at P1.0
#define greenLED BIT7                        // Green LED at P9.7
#define BUT1 BIT1                            // Button S1 at P1.1
void main(void) {
        WDTCTL = WDTPW | WDTHOLD;            // Stop the Watchdog timer
        PM5CTL0 &= ~LOCKLPM5;                // Enable the GPIO pins

        // Configure and initialize LEDs
        P1DIR |= redLED;                     // Direct pin as output
        P9DIR |= greenLED;                   // Direct pin as output
        P1OUT &= ~redLED;                    // Turn LED Off
        P9OUT &= ~greenLED;                  // Turn LED Off

// Configure buttons
        P1DIR &= ~BUT1; // Direct pin as input
```

P1DIR, P1IN and P1OUT → To work with P1
P1REN → To enable/disable resistor for pull-up/pull-down network.
Once port is an input, P1OUT is used to configure the resistor as pull-up or pull-down.

Push Buttons for Lab 2

- **Writing a code that turns the RED LED when button S1 pushed.**

```
#include <msp430fr6989.h>
#define redLED BIT0                        // Red LED at P1.0
#define greenLED BIT7                      // Green LED at P9.7
#define BUT1 BIT1                          // Button S1 at P1.1
void main(void) {
        WDTCTL = WDTPW | WDTHOLD;          // Stop the Watchdog timer
        PM5CTL0 &= ~LOCKLPM5;              // Enable the GPIO pins

        // Configure and initialize LEDs
        P1DIR |= redLED;                   // Direct pin as output
        P9DIR |= greenLED;                 // Direct pin as output
        P1OUT &= ~redLED;                  // Turn LED Off
        P9OUT &= ~greenLED;                // Turn LED Off

// Configure buttons
        P1DIR &= ~BUT1; // Direct pin as input
```

```
// Configure buttons
P1DIR &= ~BUT1;          // Direct pin as input
P1REN |= BIT1;           // Enable built-in resistor (for the specific bit)
P1OUT |= BIT1            // Set the pull-up resistor (active low button to pull it low)

for(;;) {
        if (!(P1IN & BIT1)) {  // Check if the button S1 is pressed
            P1OUT |= BIT0;     // Turn on the red LED (P1.0)
        } else {
            P1OUT &= BIT0;     // Turn off the red LED (P1.0)
```

P1DIR, P1IN and P1OUT → To work with P1

P1REN → To enable/disable resistor for pull-up/pull-down network.

Once port is an input, P1OUT is used to configure the resistor as pull-up or pull-down.

Push Buttons for Lab 2

- **Writing a code that turns the RED LED when button S1 pushed.**

```
#include <msp430fr6989.h>
#define redLED BIT0                          // Red LED at P1.0
#define greenLED BIT7                        // Green LED at P9.7
#define BUT1 BIT1                            // Button S1 at P1.1
void main(void) {
        WDTCTL = WDTPW | WDTHOLD;            // Stop the Watchdog timer
        PM5CTL0 &= ~LOCKLPM5;                // Enable the GPIO pins

        // Configure and initialize LEDs
        P1DIR |= redLED;                     // Direct pin as output
        P9DIR |= greenLED;                   // Direct pin as output
        P1OUT &= ~redLED;                    // Turn LED Off
        P9OUT &= ~greenLED;                  // Turn LED Off

// Configure buttons
        P1DIR &= ~BUT1; // Direct pin as input
```

```
// Configure buttons
P1DIR &= ~BUT1;          // Direct pin as input
P1REN |= BIT1;           // Enable built-in resistor (for the specific bit)
P1OUT |= BIT1            // Set the pull-up resistor (active low button to pull it low)

for(;;) {
        if (!(P1IN & BIT1)) {  // Check if the button S1 is pressed
           P1OUT |= BIT0;  // Turn on the red LED (P1.0)
        } else {
           P1OUT &= BIT0; // Turn off the red LED (P1.0)
```

Is this the corrected code?
Needs to be debugged?

P1DIR, P1IN and P1OUT → To work with P1
P1REN → To enable/disable resistor for pull-up/pull-down network.
Once port is an input, P1OUT is used to configure the resistor as pull-up or pull-down.

Push Buttons for Lab 2

- **Writing a code that turns the RED LED when button S1 pushed.**

```
#include <msp430fr6989.h>
#define redLED BIT0                          // Red LED at P1.0
#define greenLED BIT7                        // Green LED at P9.7
#define BUT1 BIT1                            // Button S1 at P1.1
void main(void) {
        WDTCTL = WDTPW | WDTHOLD;            // Stop the Watchdog timer
        PM5CTL0 &= ~LOCKLPM5;                // Enable the GPIO pins

        // Configure and initialize LEDs
        P1DIR |= redLED;                     // Direct pin as output
        P9DIR |= greenLED;                   // Direct pin as output
        P1OUT &= ~redLED;                    // Turn LED Off
        P9OUT &= ~greenLED;                  // Turn LED Off

// Configure buttons
        P1DIR &= ~BUT1; // Direct pin as input
```

```
// Configure buttons
P1DIR &= ~BUT1;          // Direct pin as input
P1REN |= BIT1;           // Enable built-in resistor (for the specific bit)
P1OUT |= BIT1            // Set the pull-up resistor (active low button to pull it low)

for(;;) {
        if (!(P1IN & BIT1)) {  // Check if the button S1 is pressed
            P1OUT |= BIT0;  // Turn on the red LED (P1.0)
        } else {
            P1OUT &= BIT0; // Turn off the red LED (P1.0)
```

Now how to do it using two Push buttons each for one LED?

- ***S1 for redLED***
- ***S2 for greenLED***

P1DIR, P1IN and P1OUT → To work with P1

P1REN → To enable/disable resistor for pull-up/pull-down network.

Once port is an input, P1OUT is used to configure the resistor as pull-up or pull-down.

Push Buttons for Lab 2

- Writing a code that turns the RED LED when button S1 pushed.

```
#include <msp430fr6989.h>
#define redLED BIT0                          // Red LED at P1.0
#define greenLED BIT7                        // Green LED at P9.7
#define BUT1 BIT1                            // Button S1 at P1.1
void main(void) {
        WDTCTL = WDTPW | WDTHOLD;            // Stop the Watchdog timer
        PM5CTL0 &= ~LOCKLPM5;                // Enable the GPIO pins

        // Configure and initialize LEDs
        P1DIR |= redLED;                     // Direct pin as output
        P9DIR |= greenLED;                   // Direct pin as output
        P1OUT &= ~redLED;                    // Turn LED Off
        P9OUT &= ~greenLED;                  // Turn LED Off

// Configure buttons
        P1DIR &= ~BUT1; // Direct pin as input
```

```
// Configure buttons
P1DIR &= ~BUT1;          // Direct pin as input
P1REN |= BIT1;           // Enable built-in resistor (for the specific bit)
P1OUT |= BIT1            // Set the pull-up resistor (active low button to pull it low)

for(;;) {
        if (!(P1IN & BIT1)) {  // Check if the button S1 is pressed
            P1OUT |= BIT0;     // Turn on the red LED (P1.0)
        } else {
            P1OUT &= BIT0;     // Turn off the red LED (P1.0)
```

Now how to do it using two Push buttons each for one LED?

- ***S1 | S2 for redLED***
- ***S1 ^ S2 for greenLED***

P1DIR, P1IN and P1OUT → To work with P1

P1REN → To enable/disable resistor for pull-up/pull-down network.

Once port is an input, P1OUT is used to configure the resistor as pull-up or pull-down.

Thank You!

Questions?

Email: kamali@ucf.edu
UCF HEC 435 (407) 823 – 0764
https://www.ece.ucf.edu/~kamali/

HAVEN Research Group
https://haven.ece.ucf.edu/