EEE4742C – Embedded Systems
Module 1 – Intro to Embedded Systems

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/

What is Embedded Systems

- Embedded Systems are everywhere
  - In every second of our lives
- Def 1: Any computing system w/ tightly coupling of HW and SW for a specific function
- Def 2: Any system with an IC in it (computer)
- Def 3: Any system with invisible data computation/monitoring

Network equipment / audio systems / computers / video systems / Appliances / Gaming Systems

- What main products:
  - Microprocessors and Microcontrollers

What is Embedded Systems

- Embedded Systems are everywhere
  - In every second of our lives
- In the home: microwave oven, fridge, oven, TV, wireless router, camera, alarm clock, calculator, electronic door lock
- In the car: engine control unit, infotainment system, remote control
- In the classroom: system that controls lights, projection screen and projector
- In the buildings: fire alarm system, elevator, door access, AC system
- In the city: vending machine, ATM machine, parking meter, toll collector.

What is Embedded Systems

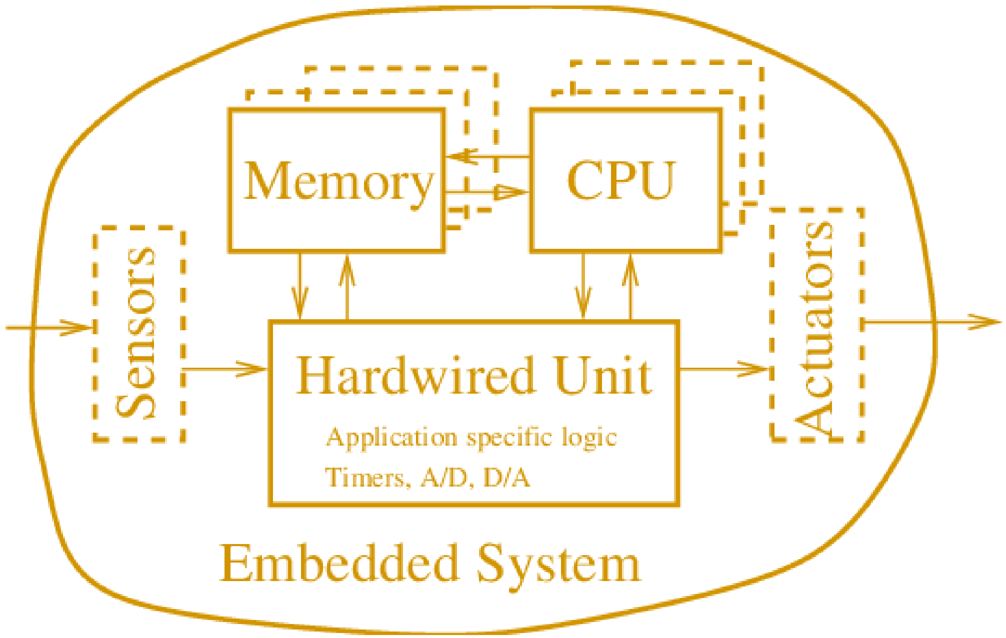
- Embedded Systems are everywhere
  - In every second of our lives
- We can say anything sounds smart opposed to the computer inside
  - How does it do such smart action?

    Hidden in the plain sight........
  - Necessarily no operating system e.g., Windows or MacOS
  - No possibility to upgrade it by user

Network equipment / audio systems / computers / video systems / Appliances / Gaming Systems

What is Embedded Systems

- Regardless of the function, embedded systems has two major components:
  - HW
    - The processing unit (CPU)
    - The storage unit (Memory)
  - SW
    - Software program (e.g., firmware), stored in the memory

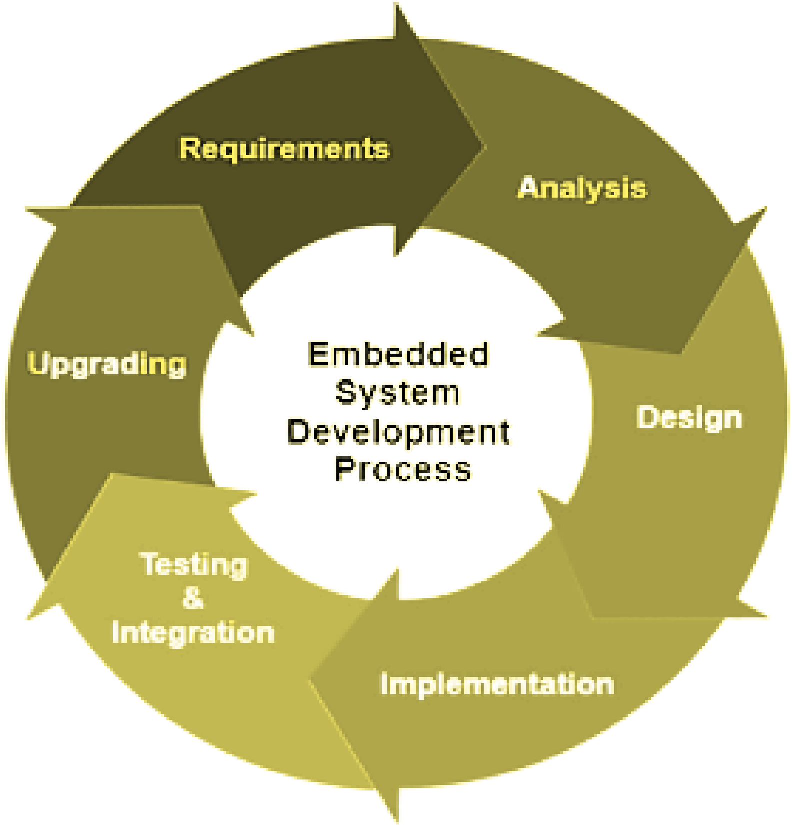


- From the embedded engineer point of view:
  - The interaction between these components require the utmost attention
    - For cost, performance, functionality, time to market, etc.
    - Trade-off: like reduce the cost as long as performance is met.
    - Trade-off: like we do not need quad-core CPUs in the microwave.

What is Embedded Systems

- Constraints of Embedded Systems
    - Resource-constrained (Small size / Small energy)
    - Environmental process variations
        - At high temperature,
        - With power fluctuation,
        - Dealing with water
        - RF interference
        - Under physical pressure

- Widely used in safety critical applications
    - Must function correctly (0.999999 reliability)



History of Embedded Systems

- 1960s: First Embedded Systems
  - developed by Charles Stark Draper and his team
    - at the MIT Instrumentation Laboratory
  - used in the Apollo spacecraft
  - Processor
    - arch.: 16bit word length, 15 bits data, 1 bit parity
    - Freq.: 2.048 MHz (160K Instruction/second)
  - Memory
    - RAM: 2KB (temporary) and ROM: 36KB (navigation control)
  - Using Assembly language
  - “Executive” RTOS
  - High reliability for Apollo missions

Apollo Guidance Computer (AGC) – HW and Interface (1966)

History of Embedded Systems

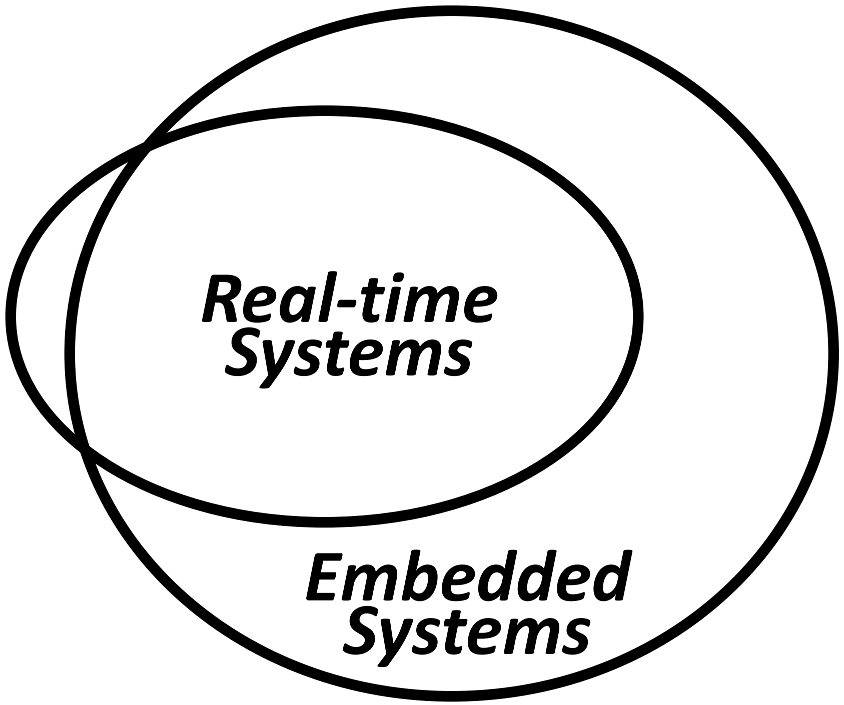
- 1960s: First Embedded Systems
  - developed by Charles Stark Draper and his team
    - at the MIT Instrumentation Laboratory
  - used in the Apollo spacecraft
  - Processor
    - arch.: 16bit word length, 15 bits data, 1 bit parity
    - Freq.: 2.048 MHz (160K instruction/second)
  - Memory
    - RAM: 2KB (temporary) and ROM: 36KB (navigation control)
  - Using Assembly language
  - “Executive” RTOS
  - High reliability for Apollo missions

For comparison, Intel 4004, the first commercial microprocessor was released in 1971. Texas Instruments TMS1000, the first microcontroller was released in 1974. (4-bit CPUs at the oldest edition)

Apollo Guidance Computer (AGC) – HW and Interface (1966)

Real-time Systems

- Real-time systems are a special type of embedded systems
  - Specific function but should be done at a specific time!
    - Tasks should be done on time for the system to work properly.
    - E.g., airplanes, Apollo Guidance Computer (AGC), car’s engine, etc.
- Different forms of time constrains for real-time systems
  - Soft: Missing a few threshold should be fine
  - Hard: Catastrophe occurs if a threshold is violated (0.99999 reliability).
- The OS is responsible for thread management (to meet the timing).
- Mostly safety-critical
  - Airplanes, cars, etc.



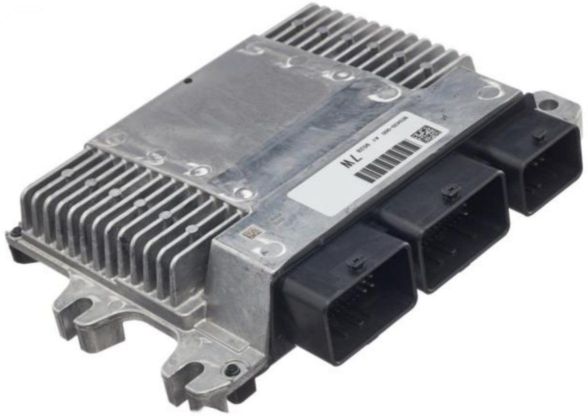
History of Embedded Systems

- 1970s: Growth in Consumer Electronics
  - e.g., calculators, digital watches, and video game consoles.

The Pocket Calculator Race (1972)

- 1980s: Automotive and Industrial Applications
  - e.g., Electronic Control Units (ECUs) for tasks such as engine control, anti-lock braking systems (ABS).
  - e.g., automation and control in manufacturing processes (robotics).

Nissan Engine Control Module (ECM) 1980



- 1990s: Integration with the Internet and Telecommunications
  - e.g., TCP/IP enabled embedded systems, Cell phones.

History of Embedded Systems

- 2000s: Expand to daily life
  - e.g., appliances and smart homes
- 2010s: The rise of internet-of-things (IoTs)
  - Always connected IoT devices for smart usages.
- 2020s: AI advances and Edge Computing
  - Smart Phones, smart watches, implants for smart usages.

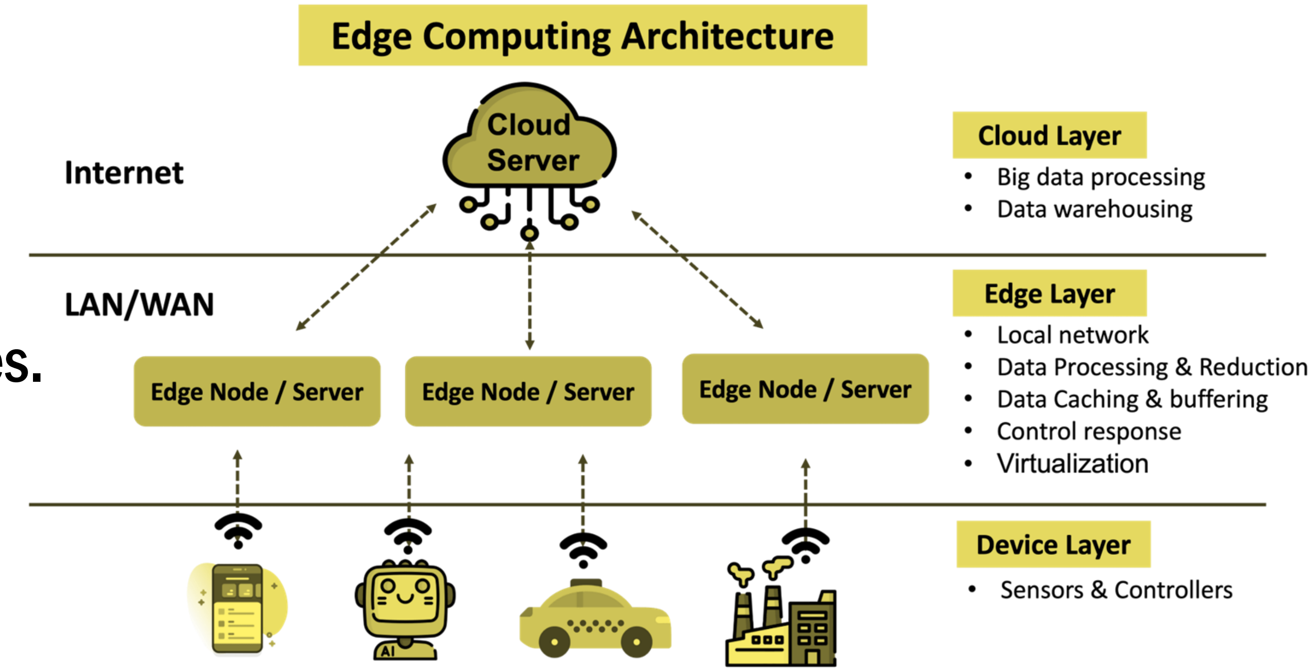


Figure : Edge computing architecture overview
Source : The research team

Survey of Embedded Engineers

- To which field does your embedded project belong?
  - Industrial controls & automation
  - Consumer electronics
  - Communication / networks
  - Medical
  - Automotive
  - Military
  - Computer peripherals (mouse, keyboard)
  - Video / imaging
  - Transportation (airport, bus, taxi)
  - Security, audio, electronic instruments…

Which of these showed up in the responses?

Survey of Embedded Engineers

- To which field does your embedded project belong?
  - Industrial controls & automation 33%
  - Consumer electronics 23%
  - Communication / networks 23%
  - Medical 15%
  - Automotive 15%
  - Military 15%
  - Computer peripherals (mouse, keyboard) 11%
  - Video / imaging 8%
  - Transportation (airport, bus, taxi)
  - Security, audio, electronic instruments…

Survey of Embedded Engineers

- Resource allocation
  - Software
  - Hardware

- Programming language
  - C
  - C++
  - Assembly language
  - Java

What percentage corresponds to each answer choice?

- Do embedded projects use an Operating System (OS)?
  - Yes
  - No

Survey of Embedded Engineers

- Resource allocation
  - Software 60%
  - Hardware 40%

- Programming language
  - C 60%
  - C++ 20%
  - Assembly language 5%
  - Java 2%

- Do embedded projects use an Operating System (OS)?
  - Yes 70%
  - No 30%

Survey of Embedded Engineers

- Main processor in the embedded project
  - 64-bit CPU
  - 32-bit CPU
  - 16-bit CPU
  - 8-bit CPU

What percentage corresponds to each answer choice?

- CPU clock rate
  - 10-99 MHz
  - 100-250 MHz
  - 250-999 MHz
  - 1 GHz
  - 2+ GHz

Survey of Embedded Engineers

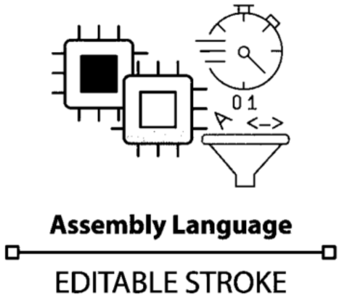
- Main processor in the embedded project
  - 64-bit CPU 6%
  - 32-bit CPU 62%
  - 16-bit CPU 16%
  - 8-bit CPU 13%

What percentage corresponds to each answer choice?

- CPU clock rate
  - 10-99 MHz 40%
  - 100-250 MHz 16%
  - 250-999 MHz 22%
  - 1 GHz 13%
  - 2+ GHz 4%

Embedded system programming

- Assembly language
    - Specific to the target processor
    - Efficient – instructions specific to the processor
- C
    - High-level language
    - Portable
    - Independent of processor
- C++
    - Object-oriented programming
    - Data abstraction



C for embedded systems

- Some things to consider:
- The smallest variable is typically, 8-bits in C language
  - Bit fields can be used to define smaller variable [Out of scope]

- Operations on the variables affect the entire variable
  - e.g. Logical operators (&&, ||, !), → Vector Operation
  - Arithmetic operators (+,-,*,/), and even → int Operation (Vector)
  - Bitwise operators (&, |, <<, >>, ~, ^) → Bit Operation

C for embedded systems

- Consider this:
  - An 8-bit register P1OUT
  - 1 – Turn ON LED
  - 0 – Turn OFF LED

Register P1OUT

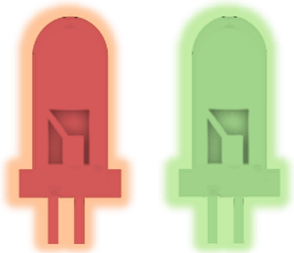
7	6	5	4	3	2	1	0
0	0	0	**1**	0	0	0	0

C code to turn ON the LED?

```
P1OUT = 0x10;         // 00010000b  <- Binary
```

C for embedded systems

- Consider a more complicated case:
  - An 8-bit register P1OUT
  - 1 – Turn ON LED
  - 0 – Turn OFF LED



Register P1OUT

7	6	5	4	3	2	1	0
0	0	0	**1**	**1**	0	0	0

C code to turn ON the LED at BIT 4, without disturbing the LED at BIT 3?

~~`P1OUT = 0x10;         // 00010000b  <- Binary`~~

```
P1OUT = 0x18;         // 00011000b  <- Binary
```

This is vector-based byte modification (with consideration of previous value) (HARD to modify → Needs Always pre-check → Better Manipulation is needed)

Bit Manipulation

- Bit manipulation is the process of performing logical operations on bit sequences in order to reach a desired result.
- Bit manipulation is a fundamental technique that is used in embedded programming to write readable code.

- Two useful bit manipulation techniques are
  - Bit masking
  - Bit fields

Bit Manipulation

- Used to manipulate the contents of registers
- Sometimes, rather than modifying the entire byte, specific bits are required to be modified.
- Consider the same example again

	7	6	5	4	3	2	1	0
Register P1OUT	0	0	0	0	1	0	0	0
Mask	0	0	0	1	0	0	0	0
P1OUT | Mask *OR operation*	0	0	0	1	1	0	0	0

Bit masking

- Four different operations can be performed
  - Set bit
  - Clear bit
  - Toggle bit
  - Check the status of a bit

OR - |

X	Y	X | Y
0	0	
0	1	
1	0	
1	1	

Set operation

AND - &

X	Y	X & Y
0	0	
0	1	
1	0	
1	1	

Clear operation

XOR - ^

X	Y	X ^ Y
0	0	
0	1	
1	0	
1	1	

Toggle operation

Bit masking

- Four different operations can be performed
  - Set bit
  - Clear bit
  - Toggle bit
  - Check the status of a bit

OR - |

X	Y	X | Y
0	0	0
0	1	1
1	0	1
1	1	1

Set operation

AND - &

X	Y	X & Y
0	0	0
0	1	0
1	0	0
1	1	1

Clear operation

XOR - ^

X	Y	X ^ Y
0	0	0
0	1	1
1	0	1
1	1	0

Toggle operation

Pre-defined Bit masking in MSP430

- The least-significant bit is called bit 0, and it can be represented in a hexadecimal mask as 0x01.
- The most-significant bit in a byte is called bit 7, and it can be represented in a hexadecimal mask as 0x80.
- MSP430 has some pre-defined masks in the header file.

BIT0	*0x0001*	***BIT4***	*0x0010*	***BIT8***	*0x0100*	***BITC***	*0x1000*
BIT1	*0x0002*	***BIT5***	*0x0020*	***BIT9***	*0x0200*	***BITD***	*0x2000*
BIT2	*0x0004*	***BIT6***	*0x0040*	***BITA***	*0x0400*	***BITE***	*0x4000*
BIT3	*0x0008*	***BIT7***	*0x0080*	***BITB***	*0x0800*	***BITF***	*0x8000*

Set bit in MSP430 (based on Bit Masking)

- Let's say we need to set the bit 4 in an 8-bit variable data.
- We can use the mask BIT4 for this operation.
  - data = data | BIT4;
- In general,
  - data = data | mask;

BIT0	*0x0001*	*BIT4*	*0x0010*	*BIT8*	*0x0100*	*BITC*	*0x1000*
BIT1	*0x0002*	*BIT5*	*0x0020*	*BIT9*	*0x0200*	*BITD*	*0x2000*
BIT2	*0x0004*	*BIT6*	*0x0040*	*BITA*	*0x0400*	*BITE*	*0x4000*
BIT3	*0x0008*	*BIT7*	*0x0080*	*BITB*	*0x0800*	*BITF*	*0x8000*

Clear bit in MSP430 (based on Bit Masking)

- Let's say we need to clear the bit 5 in an 8-bit variable data.
- We can use the mask BIT5 for this operation.

	7	6	5	4	3	2	1	0
data	0	0	1	1	1	0	0	0
BIT5	0	0	1	0	0	0	0	0
~BIT5	1	1	0	1	1	1	1	1
data & ~BIT5 (*AND operation*)	0	0	0	1	1	0	0	0

Clear bit in MSP430 (based on Bit Masking)

- Let's say we need to clear the bit 5 in an 8-bit variable data.
- We can use the mask BIT5 for this operation.
  - data = data & ~BIT5;
- In general,
  - data = data & ~mask;

BIT0	*0x0001*	***BIT4***	*0x0010*	***BIT8***	*0x0100*	***BITC***	*0x1000*
BIT1	*0x0002*	***BIT5***	*0x0020*	***BIT9***	*0x0200*	***BITD***	*0x2000*
BIT2	*0x0004*	***BIT6***	*0x0040*	***BITA***	*0x0400*	***BITE***	*0x4000*
BIT3	*0x0008*	***BIT7***	*0x0080*	***BITB***	*0x0800*	***BITF***	*0x8000*

Toggle bit in MSP430 (based on Bit Masking)

- Let's say we need to toggle the bit 5 in an 8-bit variable data.
- We can use the mask BIT5 for this operation.

	7	6	5	4	3	2	1	0
data	0	0	0	1	1	0	0	0
BIT5	0	0	1	0	0	0	0	0
~BIT5	1	1	0	1	1	1	1	1
data ^ BIT5 (*XOR operation*)	0	0	1	1	1	0	0	0

Toggle bit in MSP430 (based on Bit Masking)

- Let's say we need to toggle the bit 5 in an 8-bit variable data.
- We can use the mask BIT5 for this operation.
  - data = data ^ BIT5;
- In general,
  - data = data ^ mask;

BIT0	*0x0001*	***BIT4***	*0x0010*	***BIT8***	*0x0100*	***BITC***	*0x1000*
BIT1	*0x0002*	***BIT5***	*0x0020*	***BIT9***	*0x0200*	***BITD***	*0x2000*
BIT2	*0x0004*	***BIT6***	*0x0040*	***BITA***	*0x0400*	***BITE***	*0x4000*
BIT3	*0x0008*	***BIT7***	*0x0080*	***BITB***	*0x0800*	***BITF***	*0x8000*

- A simple use case of toggle is to blink an LED on and off.

Check bit in MSP430 (based on Bit Masking)

- Let's say we need to **check** the value of **bit 4** in an 8-bit variable data

	7	6	5	4	3	2	1	0
data	0	0	1	1	1	0	0	0
BIT4	0	0	0	1	0	0	0	0
data & BIT4	0	0	0	1	0	0	0	0

```
if ((data & BIT4)!=0)
      //bit 4 is 1
else
      //bit 4 is 0
```

```
if ((data & BIT4)==BIT4)
      //bit 4 is 1
else
      //bit 4 is 0
```

Bit Masking (Summary)

- Set bit / Clear bit / Toggle bit
  - data = data | mask;   data = data & ~mask;   data = data ^ mask;
- Check bit
  - ```
    if ((data & mask)!=0)
       { \\ bit is set }
    else
       { \\ bit is not set }
    ```

BIT0	*0x0001*	***BIT4***	*0x0010*	***BIT8***	*0x0100*	***BITC***	*0x1000*
BIT1	*0x0002*	***BIT5***	*0x0020*	***BIT9***	*0x0200*	***BITD***	*0x2000*
BIT2	*0x0004*	***BIT6***	*0x0040*	***BITA***	*0x0400*	***BITE***	*0x4000*
BIT3	*0x0008*	***BIT7***	*0x0080*	***BITB***	*0x0800*	***BITF***	*0x8000*
    ```

Exercise 1

- Set bit 1 and bit 4 bit of the register P1OUT simultaneously

	7	6	5	4	3	2	1	0
Register P1OUT	0	0	0	0	1	0	0	0

Exercise 1

- Set bit 1 and bit 4 bit of the register P1OUT simultaneously

Register P1OUT

7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0

- P1OUT = P1OUT | (BIT1 | BIT4)

7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	0

Exercise 2

- Clear bit 1 and bit 3 bit of the register P1OUT simultaneously

	7	6	5	4	3	2	1	0
Register P1OUT	0	0	0	0	1	0	0	1

Exercise 2

- Clear bit 1 and bit 3 bit of the register P1OUT simultaneously

Register P1OUT

7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1

- P1OUT = P1OUT & ~BIT1 & ~BIT3

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1

Exercise 2

- Clear bit 1 and bit 3 bit of the register P1OUT simultaneously

Register P1OUT

7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1

- P1OUT = P1OUT & ~BIT1 & ~BIT3

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1

- P1OUT = P1OUT & ~(BIT1 | BIT3)
  - The parenthesis cannot be removed directly. De Morgan's law can be applied.

Exercise 3

- If bit 1 of register P1IN is set, then set bit 3 of register P1OUT, else toggle bit 4 bit.

Exercise 3

- If bit 1 of register P1IN is set, then set bit 3 of register P1OUT, else toggle bit 4 bit.

```
If ((P1IN & BIT1)!=0) {
   P1OUT = P1OUT | BIT3
} else {
   P1OUT = P1OUT ^ BIT4
}
```

Bit Manipulation

- Bit manipulation is the process of performing logical operations on bit sequences in order to reach a desired result.
- Bit manipulation is a fundamental technique that is used in embedded programming to write readable code.
- Two useful bit manipulation techniques are
  - Bit masking
  - Bit fields

Bit Fields

- Bit filed is for clustering bits into customized groups of bits

Data = 0b 0 0 0 0 0 0 0 0

bit fields A B C D

- Let's say, we need A to be 10, B to be 110, C to be 01 and D to be 1
- Then, we could do

Data = Data | (BIT7 | BIT5 | BIT4 | BIT1 | BIT0)

- Any simpler way to do this?

Bit Fields

- Bit filed is for clustering bits into customized groups of bits

Data = 0b 0 0 0 0 0 0 0 0

bit fields A B C D

- Let's say, we need A to be 10, B to be 110, C to be 01 and D to be 1
- We can define masks for groups (fields)

A_0 = 00000000	B_0 = 00000000	C_0 = 00000000	D_0 = 00000000
A_1 = 01000000	B_1 = 00001000	C_1 = 00000010	D_1 = 00000001
A_2 = 10000000	B_2 = 00010000	C_2 = 00000100	
A_3 = 11000000	B_3 = 00011000	C_3 = 00000110	
	B_4 = 00100000		
	B_5 = 00101000		
	B_6 = 00110000		
	B_7 = 00111000		

Bit Fields

- Bit filed is for clustering bits into customized groups of bits

Data = 0b 0 0 0 0 0 0 0 0

bit fields A B C D

- Let's say, we need A to be 10, B to be 110, C to be 01 and D to be 1
- We can define masks for groups (fields)

A_0 = 00000000	B_0 = 00000000	C_0 = 00000000	D_0 = 00000000
A_1 = 01000000	B_1 = 00001000	C_1 = 00000010	D_1 = 00000001
A_2 = 10000000	B_2 = 00010000	C_2 = 00000100	
A_3 = 11000000	B_3 = 00011000	C_3 = 00000110	
	B_4 = 00100000		
	B_5 = 00101000		
	B_6 = 00110000		
	B_7 = 00111000		

- Then:

Data = Data | (A_2 | B_6 | C_1 | D_1)

Exercise

- In data, change C to 10.

Data = 0b 0 0 0 0 0 0 0 0

bit fields A B C D

Exercise

- In Data, change C to 10.

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

- The preliminary value is 01.
  - We cannot do >> Data = Data | (A_2 | B_6 | C_2 | D_1) ×
    - The C field becomes 11 instead of 10
- We need to clear C and then set it to the desired value.
  - Data = Data & ~C_3
  - Data = Data | (A_2 | B_6 | C_2 | D_1)

Some Hints from C Coding Options

- The masks are provided in the header files (.h). It is cleaner to use these predefined masks.
- We can use the assignment operators (&=, |=, ^=) from C language.

- Data = Data | (BIT7 | BIT5 | BIT4 | BIT1 | BIT0) can be replaced with
  - Data |= (BIT7 | BIT5 | BIT4 | BIT1 | BIT0)

- Data = Data & ~(BIT7 | BIT5 | BIT4 | BIT1 | BIT0) can be replaced with
  - Data &= ~(BIT7 | BIT5 | BIT4 | BIT1 | BIT0)

Some Hints from C Coding Options

- C language tends to be most popular in embedded systems programming
  - the benefits to developers outweigh the loss of program efficiency
- Bit masking can be used to manipulate individual bits in a register
  - Bit Set
  - Bit Clear
  - Bit Toggle
  - Bit Check
- Bit fields can be manipulated the same way as bit masking
- Bits within a bitfield can be individually set, tested, cleared, and toggled without affecting the state of the other bits outside the bitfield.

Prepare Yourself Now for Lab 1

- It is Flashing LED!

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- What Operations We need to Create this!

Prepare Yourself Now for Lab 1

- It is Flashing LED!

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- What Operations We need to Create this!
  - Set/clear → for the first time value
  - Toggle → for flashing
  - Loop → delay

Prepare Yourself Now for Lab 1

- It is Flashing LED!

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
        for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

- What Operations We need to Create this!
  - Set/clear → for the first time value
  - Toggle → for flashing
  - Loop → delay

Prepare Yourself Now for Lab 1

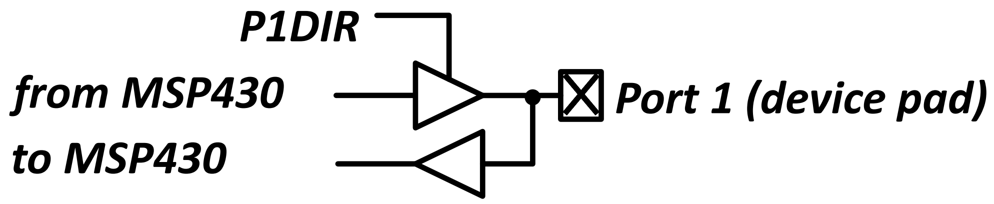
- It is Flashing LED!

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- P1DIR

Input Mode (P1DIR = 0): *The input buffer reads the voltage on the port pin. The output buffer is disconnected, so the pin is in a high-impedance state. The microcontroller can sense external signals but cannot drive the pin.*
Output Mode (P1DIR = 1): *The output buffer connects the data bus to the port pin, allowing the microcontroller to drive the pin high or low. The pin actively sources or sinks current to create a logic high or low signal.*




```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

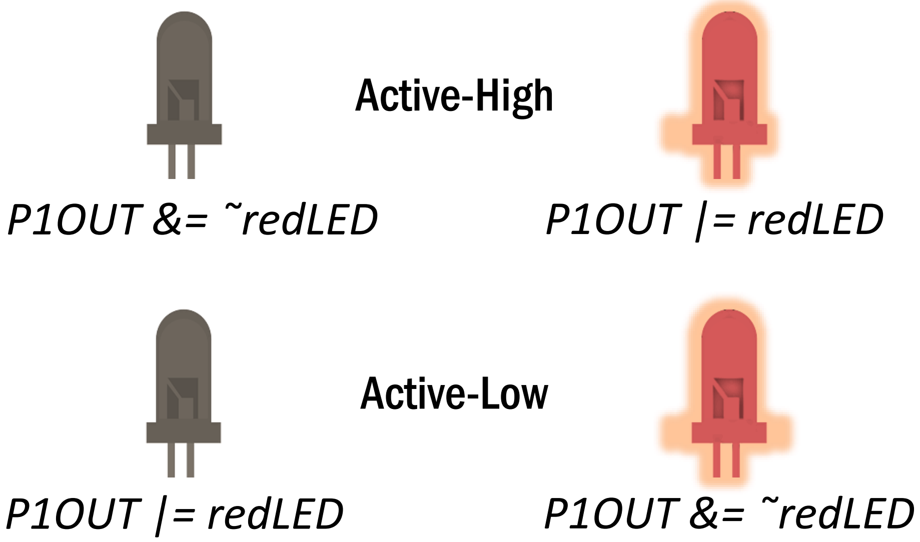
Prepare Yourself Now for Lab 1

- It is Flashing LED!

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- Active-High vs. Active-Low




```c
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- **It is Flashing LED!**

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

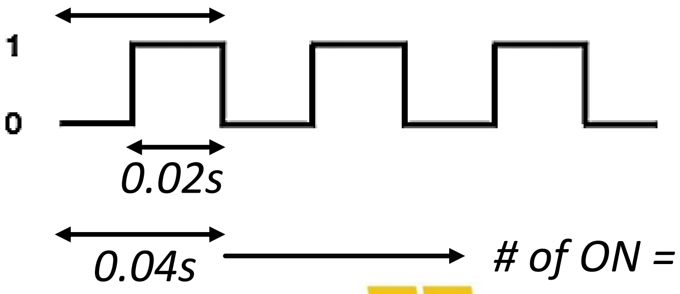
The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- **Creating Timing Cycles**
  - **Using Loops**

e.g., each loop counts take 1us. How many times ON per second?

$20{,}000 \times 10^{-6} = 2\times10^{4}\times10^{-6} = 2\times10^{-2}$ s → for each toggle

***clock period** = time of each flash (half cycle on and half cycle off)*



# of ON = 1s / 04s = ***25 times***

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- **It is Flashing LED!**

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- **Creating Timing Cycles**
  - **Using Loops**

e.g., it's a 10MHz Processor (1 CPS) How many times ON per second?

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- **It is Flashing LED!**

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

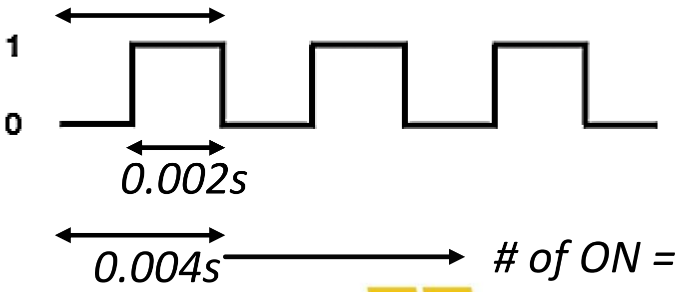
The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- **Creating Timing Cycles**
  - **Using Loops** *10,000,000 ops/s → 10^{-7}s per inst.*

e.g., it's a 10MHz Processor (1 CPS) How many times ON per second?

$20{,}000 \times 10^{-7} = 2\times10^{4}\times10^{-7} = 2\times10^{-3}$ s → for each toggle

***clock period** = time of each flash (half cycle on and half cycle off)*



default CPU clock is 1 MHz

*# of ON = 1s / 004s = **250 times***

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- **It is Flashing LED!**

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- **Creating Timing Cycles**
  - **Using Loops**

How to make it flashing faster?

How to make it flashing slower?

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- It is Flashing LED!

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- Creating Timing Cycles
    - Using Loops

*How to make it flashing faster? → **shallow loop (less than 20,000)***

*How to make it flashing slower? → **deep loop (more than 20,000)***

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
        for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- It is Flashing LED!

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- Creating Timing Cycles
    - Using Loops

What is the possible slowest loop here?

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<???; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- It is Flashing LED!

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- Creating Timing Cycles
  - Using Loops

What is the possible slowest loop here?
i is integer (32 bit but MSP considers it as 16 bit)
biggest possible value of i is 65,535!

How to make it slower?

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
        for(;;) {
                // Delay loop
                for(i=0; i<65535; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- **It is Flashing LED!**

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- **Creating Timing Cycles**
  - **Using Loops**

What is the possible slowest loop here?
i is integer (32 bit but MSP considers it as 16 bit)
biggest possible value of i is 65,535!

How to make it slower?
- *Nested loop*
- *Using 32-bit variables*
- *Delay cycle function;*

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<65535; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- It is Flashing LED!

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- Creating Timing Cycles
  - Using Loops

What is the possible slowest loop here?
i is integer (32 bit but MSP considers it as 16 bit)
biggest possible value of i is 65,535!

How to make it slower?

- ***Nested loop***
- *Using 32-bit variables*
- *Delay cycle function;*

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int I, j;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<65536; i++) {
                        for(j=0; j<16; i++) {}}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- **It is Flashing LED!**

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- **Creating Timing Cycles**
    - **Using Loops**

What is the possible slowest loop here?
i is integer (32 bit but MSP considers it as 16 bit)
biggest possible value of i is 65,535!

How to make it slower?
- *Nested loop*
- ***Using 32-bit variables***
- *Delay cycle function;*

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile uint32_t i; // 32 bit integer
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<1048576; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- **It is Flashing LED!**

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- **Creating Timing Cycles**
  - **Using Loops**

What is the possible slowest loop here?
i is integer (32 bit but MSP considers it as 16 bit)
biggest possible value of i is 65,535!

How to make it slower?

- *Nested loop*
- *Using 32-bit variables*
- ***Delay cycle function;***

default CPU clock is 1 MHz

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay cycle
                _delay_cycles(10000)
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

Prepare Yourself Now for Lab 1

- It is Flashing LED!

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

- How the code should be changed for these patterns?

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/