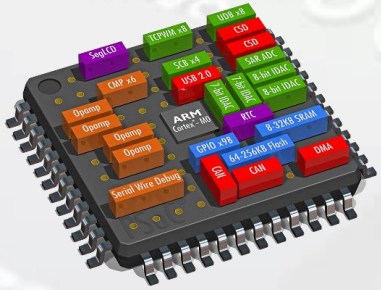
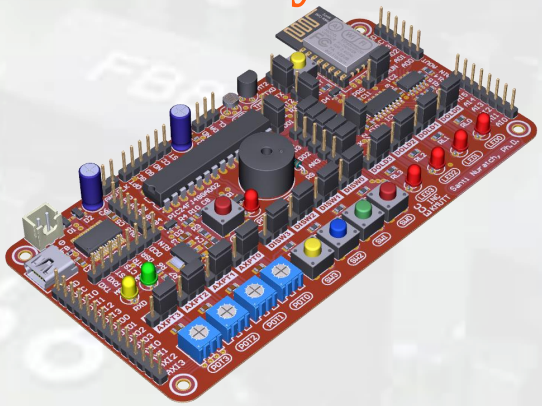
Week #02 – Tools and C Programming

INC 352:

Embedded Systems and Industrial Automation Applications Laboratory





Asst.Prof.Dr.Santi Nuratch

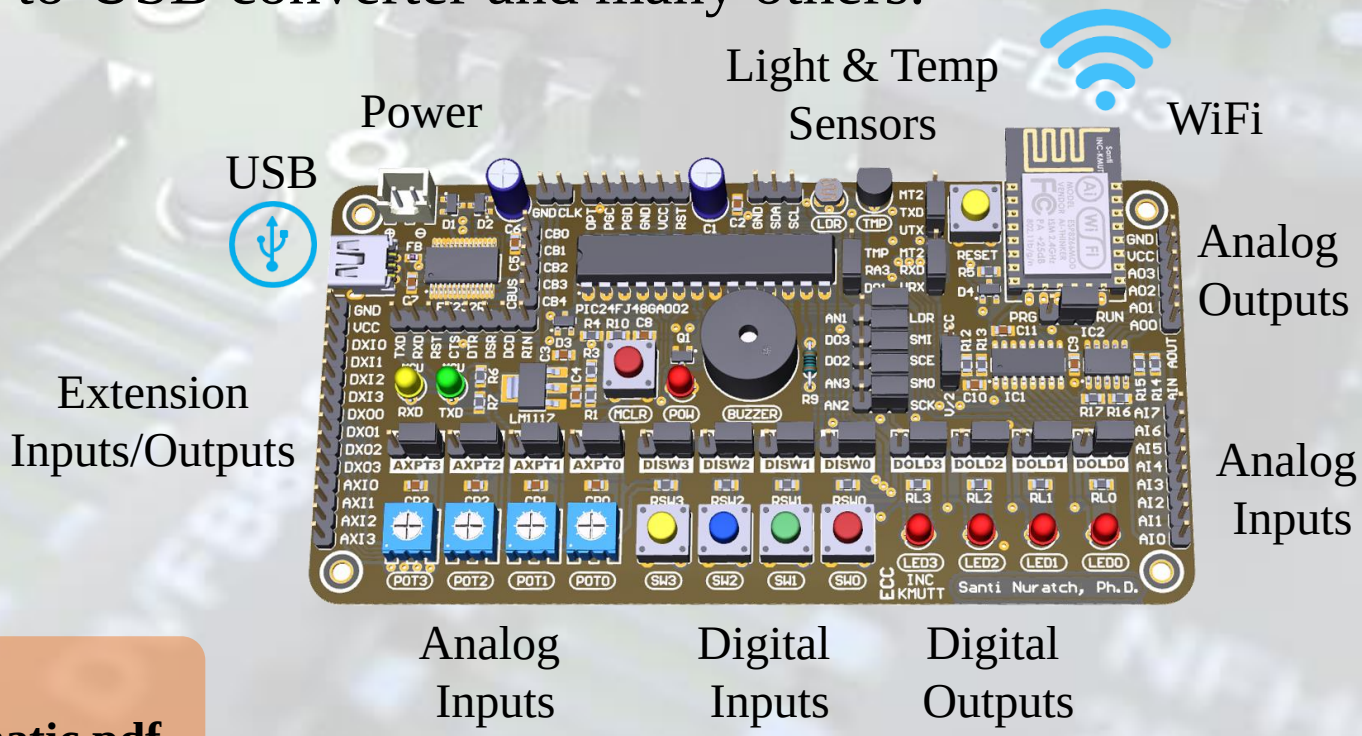
Embedded Computing and Control Lab. @ INC-KMUTT

santi.inc.kmutt@gmail.com

Department of Control System and Instrumentation Engineering,
King Mongkut's University of Technology Thonburi, **KMUTT**

The ECC-PIC24 Experiment Board

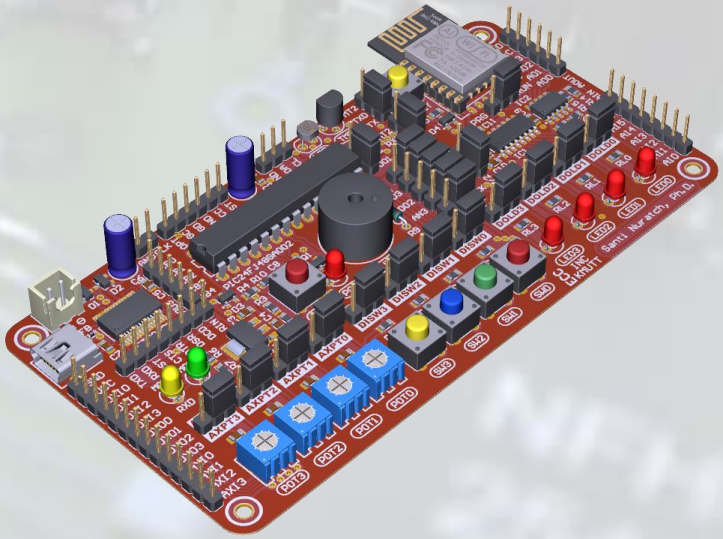
The ECC-PIC24 experiment board is design for embedded learners. The board contains many onboard input/output devices/modules, e.g.; light-emitting diodes (LEDs), push button switches (PBS), light-dependent resistor (LDR), Buzzer, UART-to-USB converter and many others.



More details:
EccPic24_Schematic.pdf

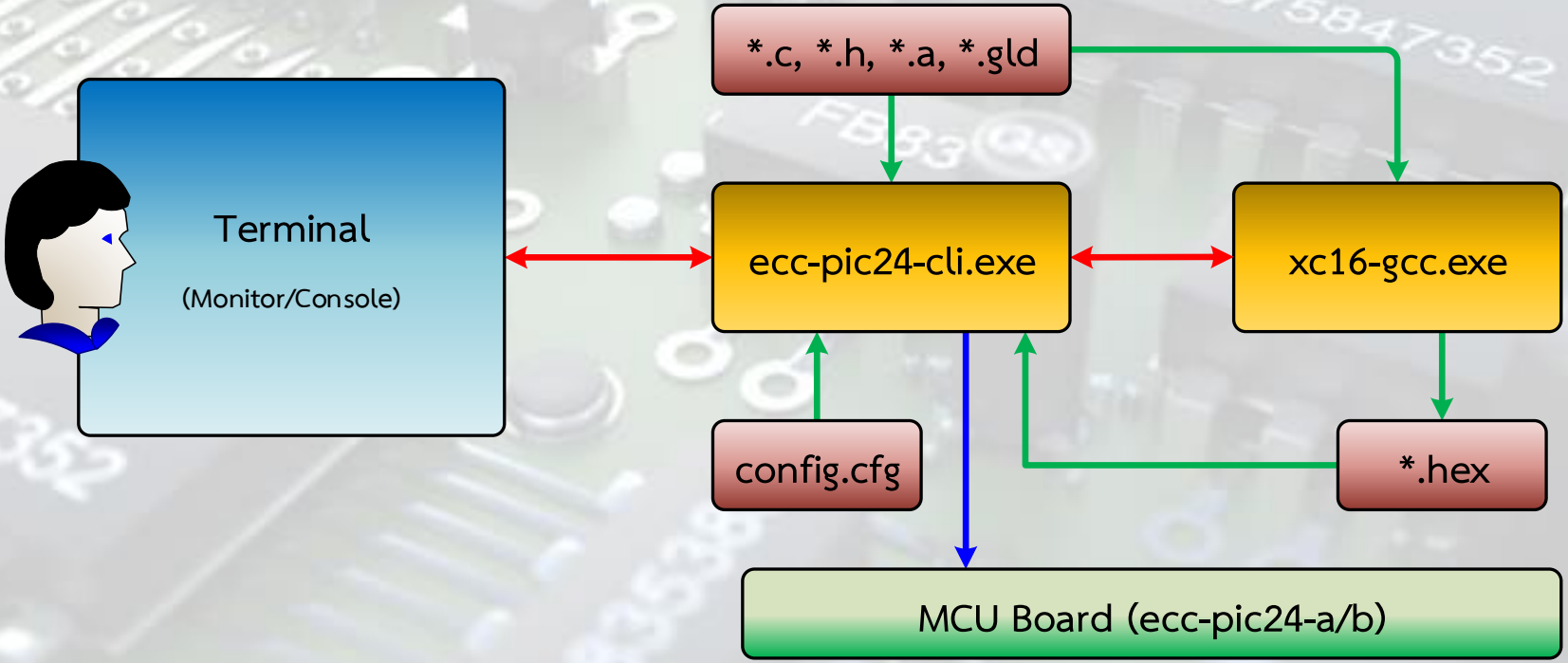
The EccOS (The Real-time Multitasking Operating System)

The EccOS is a small footprint operating system for small microcontrollers. It is deigned and developed using event-driven and callback-function techniques. It fully support many real-time multitasking applications. Internally, the EccOS provides many utility functions, e.g.; serial communication, basic function of input/output ports (Digital and Analog) and time management. Also, it has built-in functions to working with the ECC-PIC24 experiment board



What is the ecc-pic24-cli?

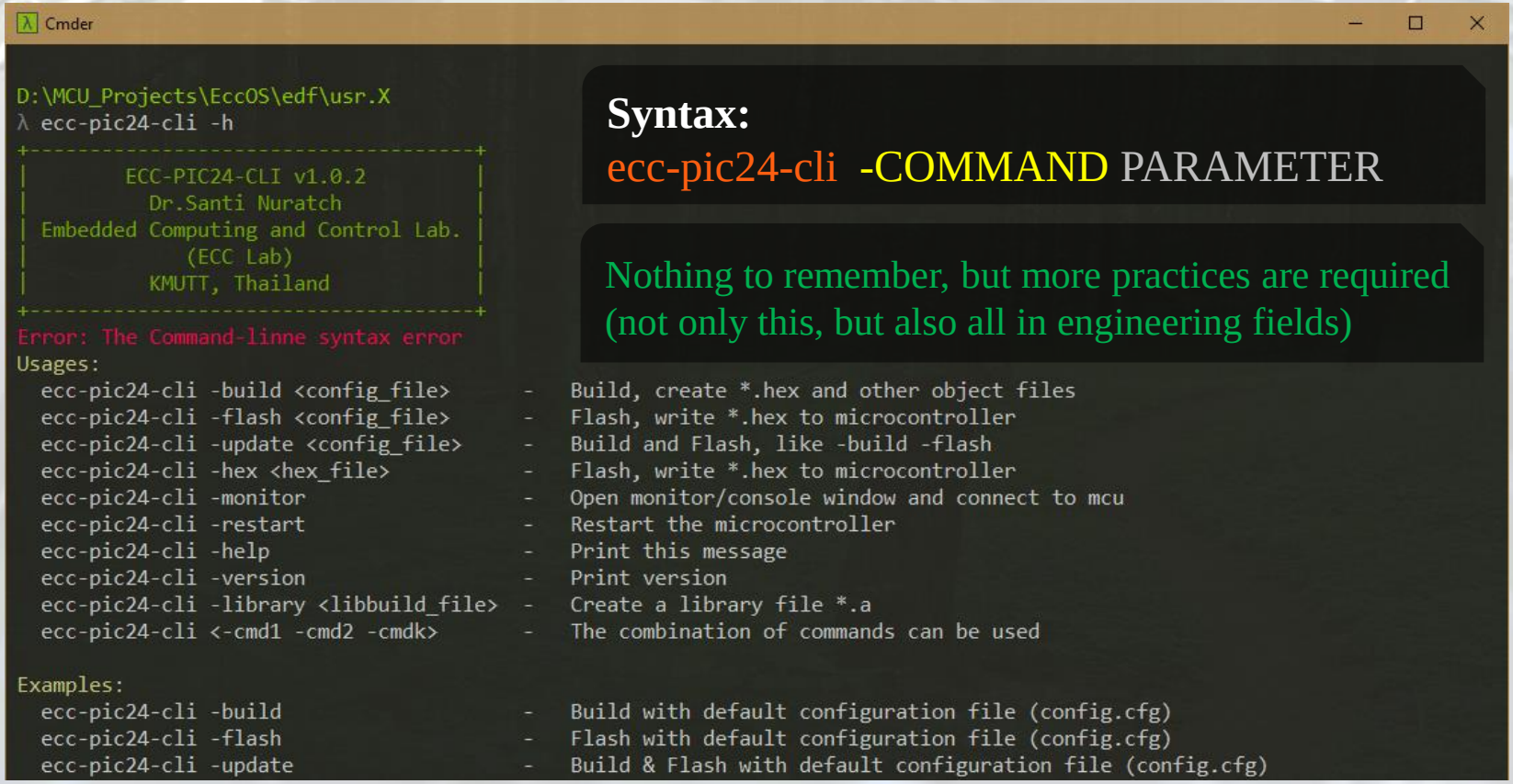
The **ecc-pic24-cli** is a command-line interface application used for linking to **xc16-gcc** compiler and microcontroller board



Mainly, the ecc-pic24-cli receives user's command(s), reads a configuration file (config.cfg) and prepares special commands for the xc16-gcc (compiler). It also reads and processes HEX file (*.hex) before sending to flash memory of target microcontroller.

ecc-pic24-cli commands

To work with the **ecc-pic24-cli**, or others command-line interface applications, a terminal/console is required, e.g.; the Command Prompt (CMD) or other console emulators for Windows. The **cmder** is recommended. Let's try the first command, **ecc-pic24-cli -h**



ecc-pic24-cli configuration file

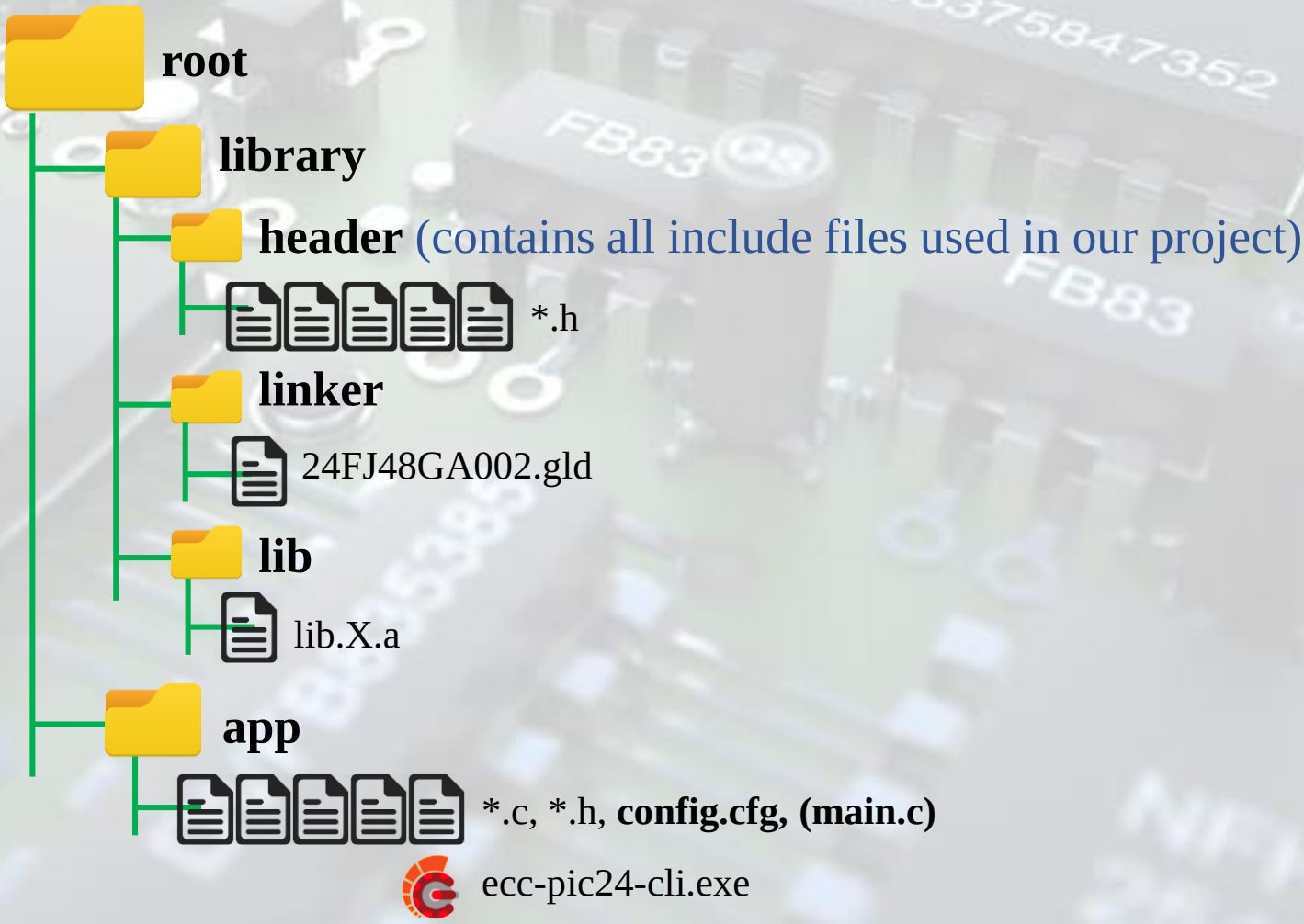
To make the **ecc-pic24-cli** works correctly, a configuration file (config.cfg) is needed to be written carefully. The config.cfg contains all information of a project, e.g.; source files and directories, communication properties, compiler's path and others.

```
MCU_PART = 24FJ48GA002
INC_DIR  = ../src
LNK_FILE = ../src/24FJ48GA002.gld
LIB_FILE = ../lib.X/dist/default/production/lib.X.a
OUT_DIR  = ./output
HEX_FILE = ./ecc.hex

COM_NAME = COM5
COM_BAUD = 57600

SRC_FILE  = ./main.c
#SRC_FILE = D:\MCU_Projects\EccOS\ecc\ecc.X\source\config.c

XC16_DIR  = C:\Program Files(x86)\Microchip\xc16\v1.40\bin
```

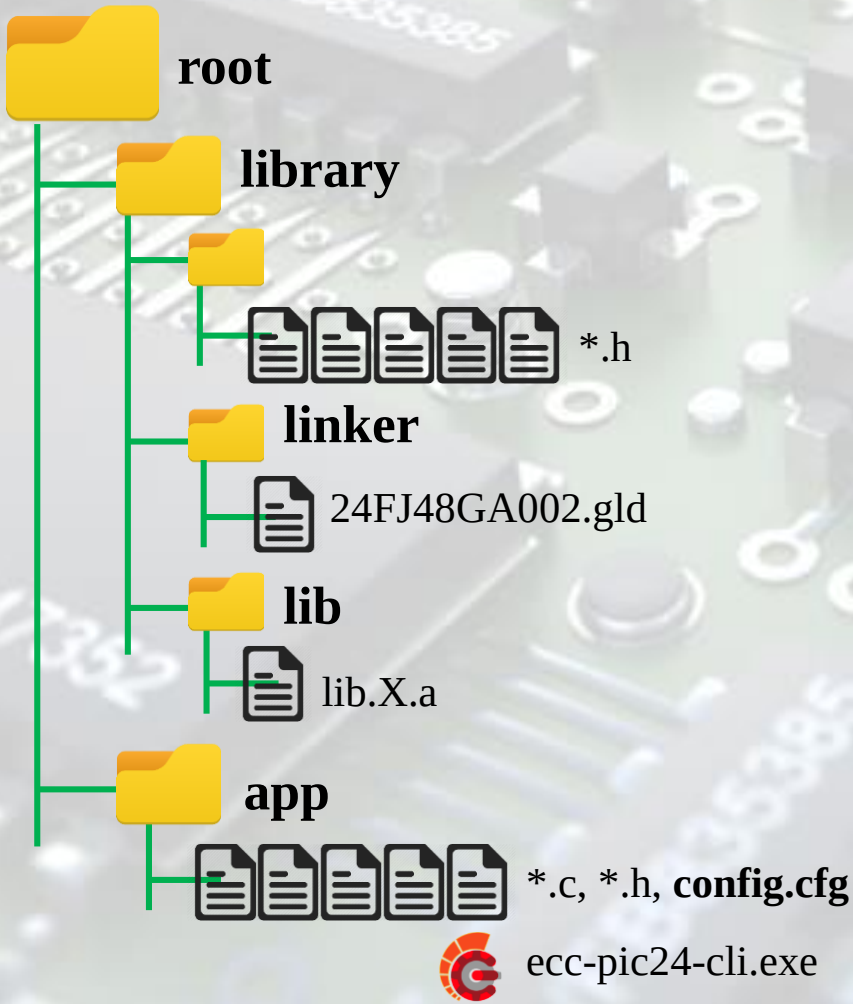
We have to know exactly their meanings

ecc-pic24-cli configuration file

Create (organize) your own project and build the **config.cfg**



ecc-pic24-cli configuration file




```
#MCU Part No.
MCU_PART = 24FJ48GA002

#Include Directory
INC_DIR  = ../library/header

#Linker script file for the MCU
LNK_FILE = ../library/linker/24FJ48GA002.gld

#Library file (OS and other utillities)
LIB_FILE = ../library/lib/lib.X.a

#Output directory for compiler
OUT_DIR  = ./output

#HEX file name and location
HEX_FILE = ./machine_code.hex

#Communication port
#(e.g.; the bord is connected to COM6)
COM_NAME = COM6

#Communication speed (bit/baud rate)
COM_BAUD = 57600

#Applocation file(s)
SRC_FILE = ./main.c
#SRC_FILE = ./another.c

#Compiler bin directory
XC16_DIR = C:\Program Files (x86)\Microchip\xc16\v1.40\bin
```

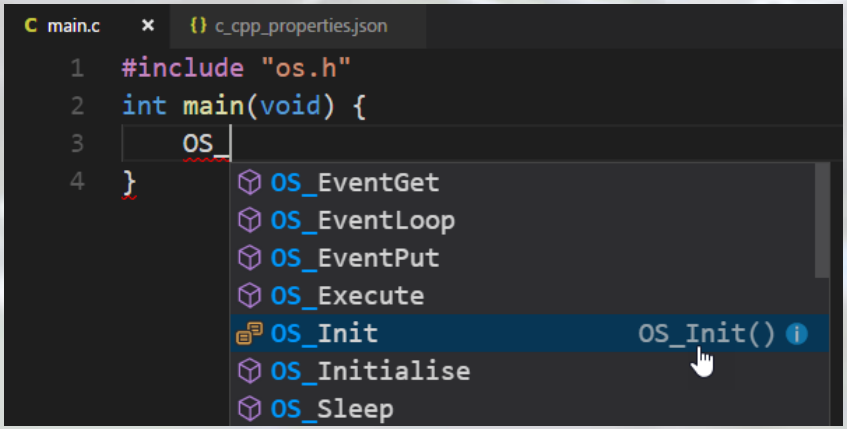
VS-Code Setup

To make the VS-Code shows **intellisense**, the following steps are required:

1) Open the application directory (root or app)

2) Open the **c_cpp_properties.json**, and add the lines shown below into the **includePath** and **intelliSenseMode**

```
"D:/MyDirectory/root/library/header/*",
"C:/Program Files (x86)/Microchip/xc16/v1.40/include",
"C:/Program Files (x86)/Microchip/xc16/ v1.40/support/generic/h",
"C:/Program Files (x86)/Microchip/xc16/ v1.40/support/PIC24F/h",
```



All are depended on your system.

3) Restart the VS-Code and check the intellisense

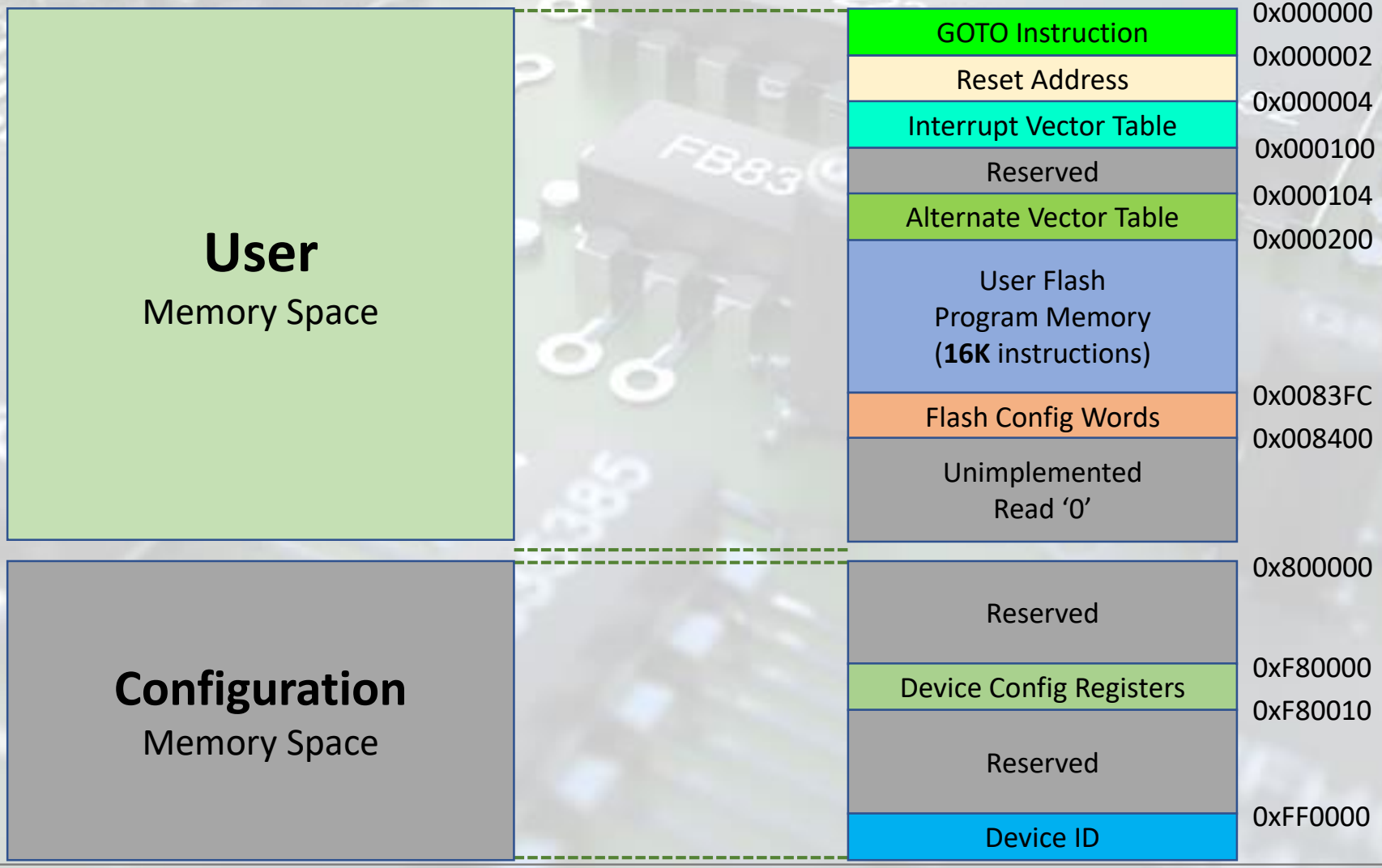
The First C-Program (based-on EccOS)

Programming with EccOS (and others) requires special knowledge of C-Programming, the Embedded C Programming. In this class, we are not going in details on that topic, but you have to do!. The lines of code below are the main function of the C-Program. Inside the main, just two functions of the OS are required. The **`OS_Init()`** must be executed before all functions of the EccOS are called. The **`OS_Start()`** must be executed to start the OS.

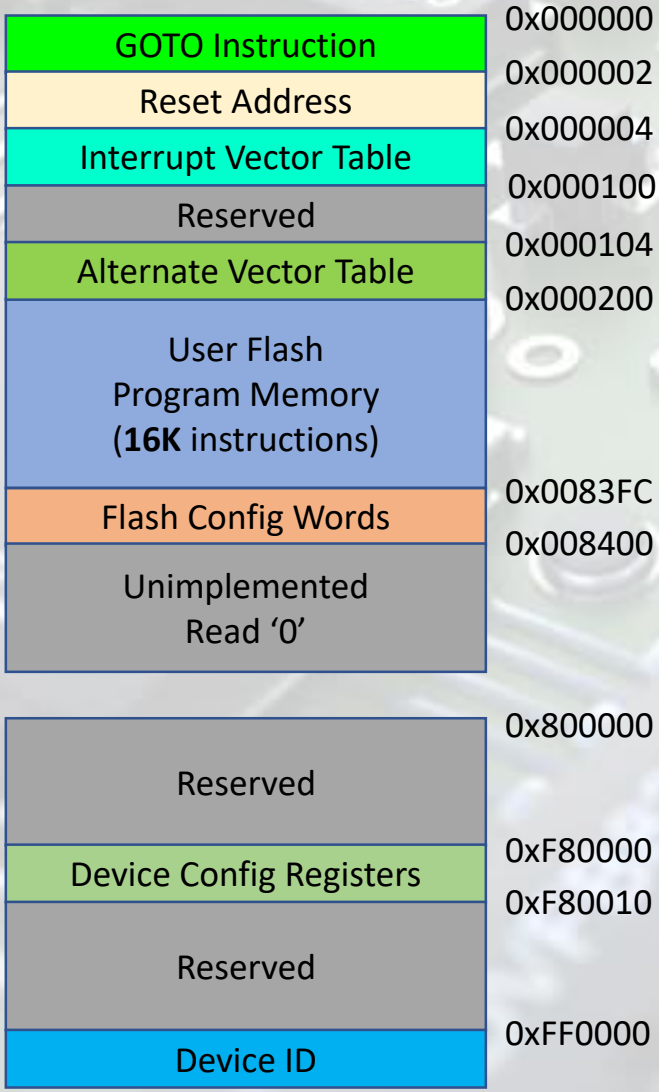
```
int main(void)
{
    // peripherals and variables initialization
    OS_Init();
    // Other initializations
    OS_Start();
}
```

Memory and Variables

Programming is the exchanging (read/write) data between memories located in different locations/addresses



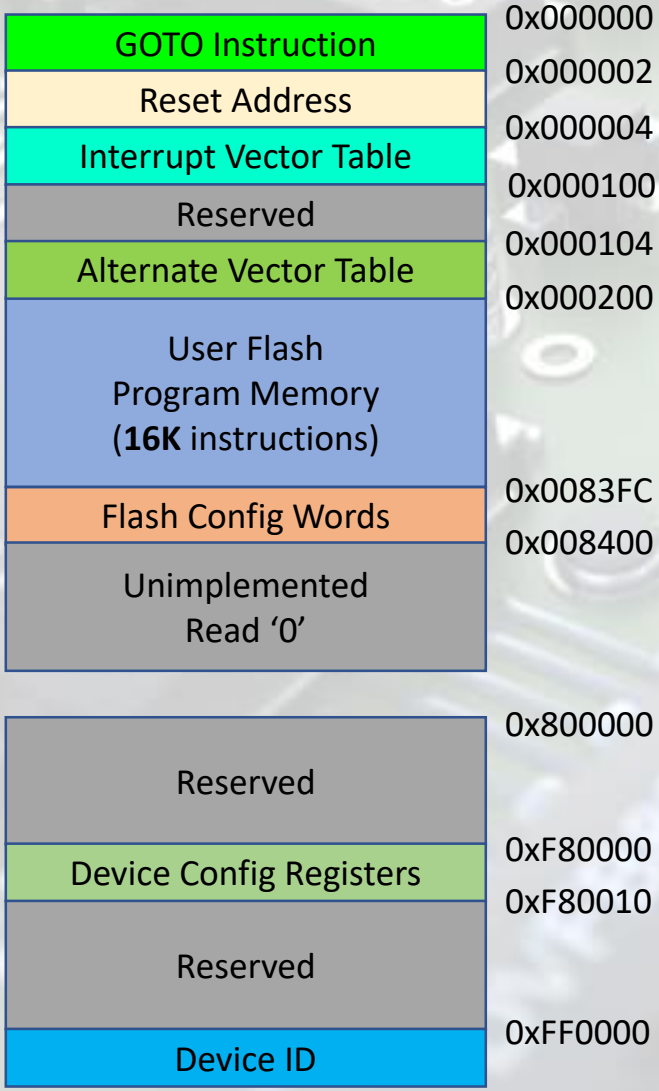
Memory and Variables (PIC2FJxxx)



Hard Memory Vectors

- The addresses between **0x00000** and **0x000200** for hard code program execution vectors
- A hardware **Reset vector** is provided to redirect code execution from the default value of the **PC** on device **Reset** to the **actual start of code**
- A **GOTO instruction** is programmed by the user at **0x000000** with the actual address for the start of code at **0x000002**
- There are two interrupt vector tables, located from **0x000004** to **0x0000FF** and **0x000100** to **0x0001FF**

Memory and Variables (PIC2FJxxx)

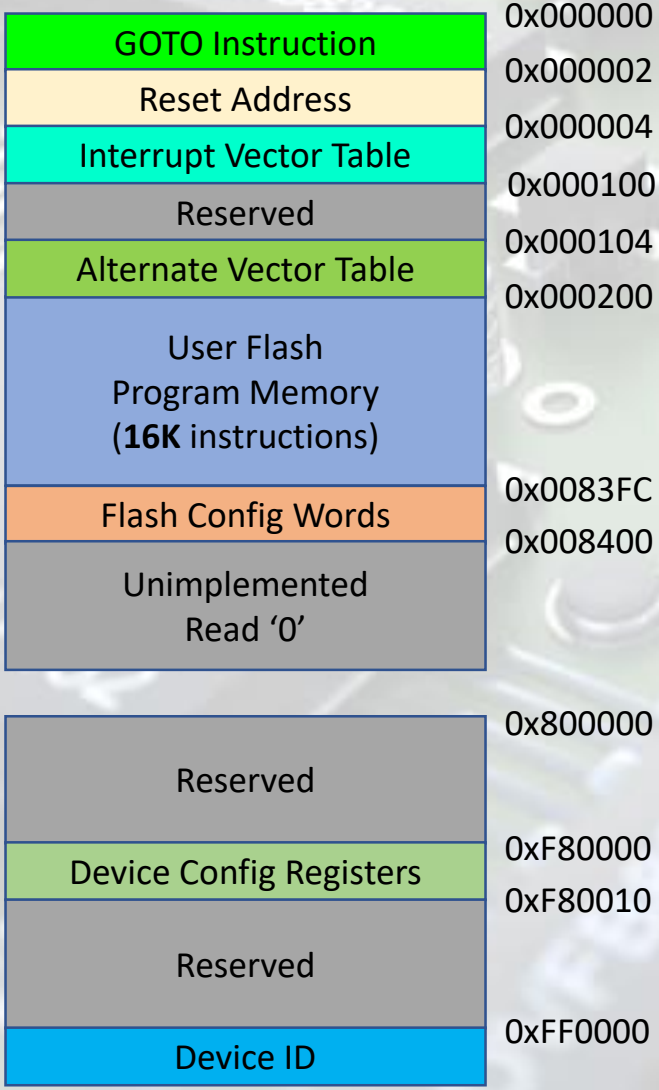


Flash Configuration Words

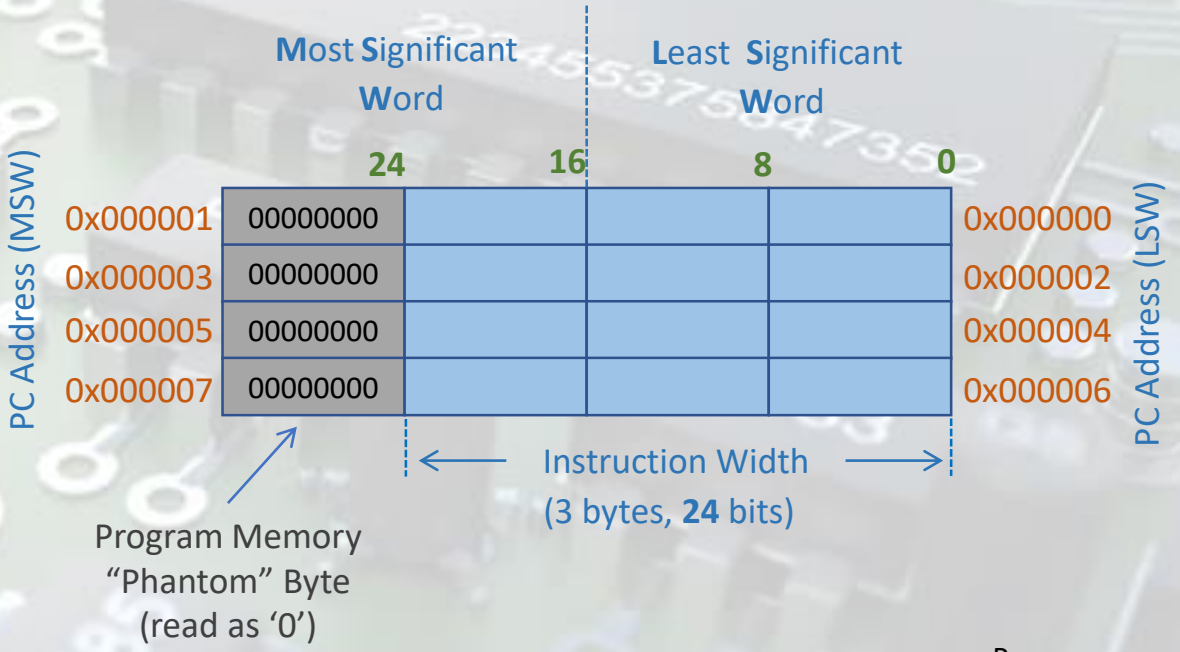
- The **top two words** of on-chip program memory are reserved for configuration information.
- On device **Reset**, the configuration information is copied into the appropriate **Configuration registers**.

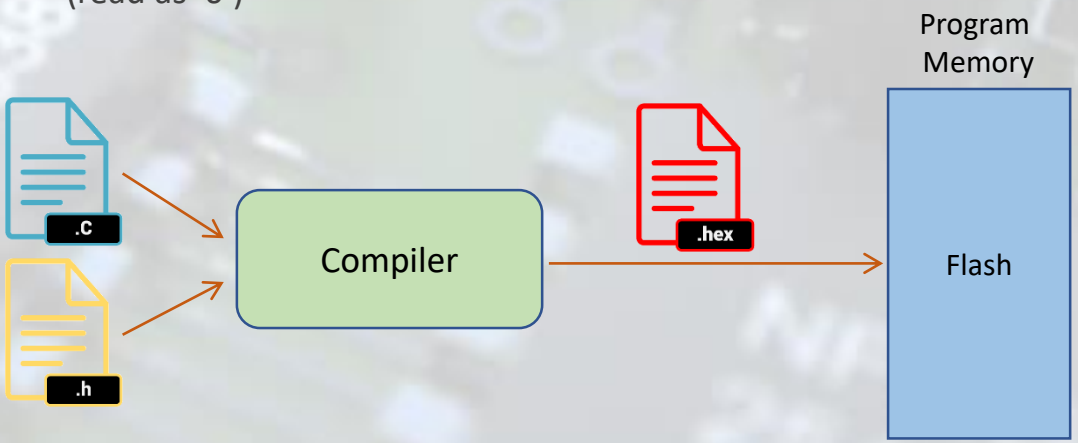
Devices	Program Memory (K words)	Configuration Word Address
PIC24FJ16GA	5.5	0x002BFC – 0x002BFE
PIC24FJ32GA	11	0x0057FC – 0x0057FE
PIC24FJ48GA	**16**	**0x0083FC – 0x0083FE**
PIC24FJ64GA	22	0x00ABFC – 0x00ABFE

Memory and Variables (PIC2FJxxx)



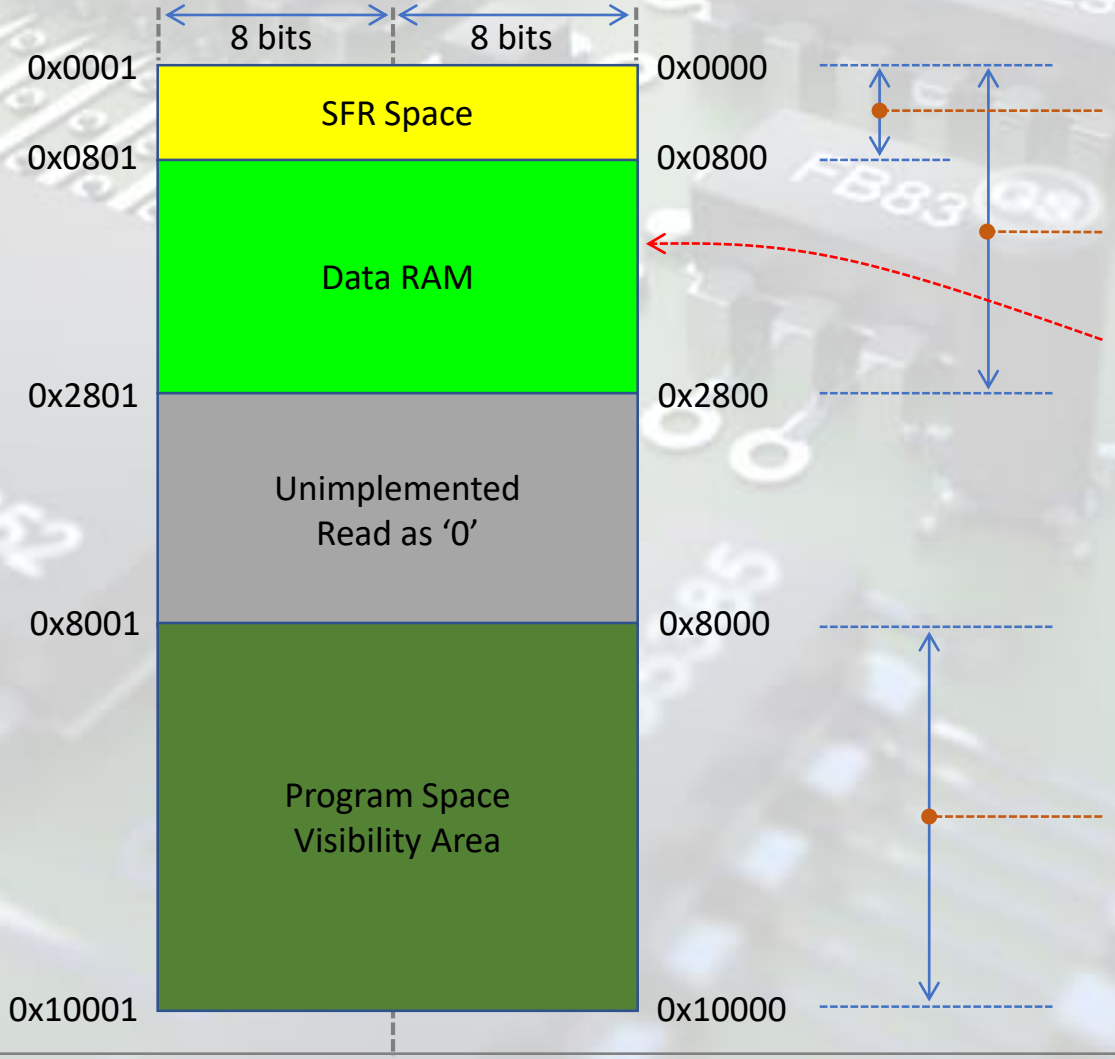
Program Memory Organization





Memory and Variables (PIC2FJxxx)

Data Address Spaces



Special **F**unction **R**egister Space

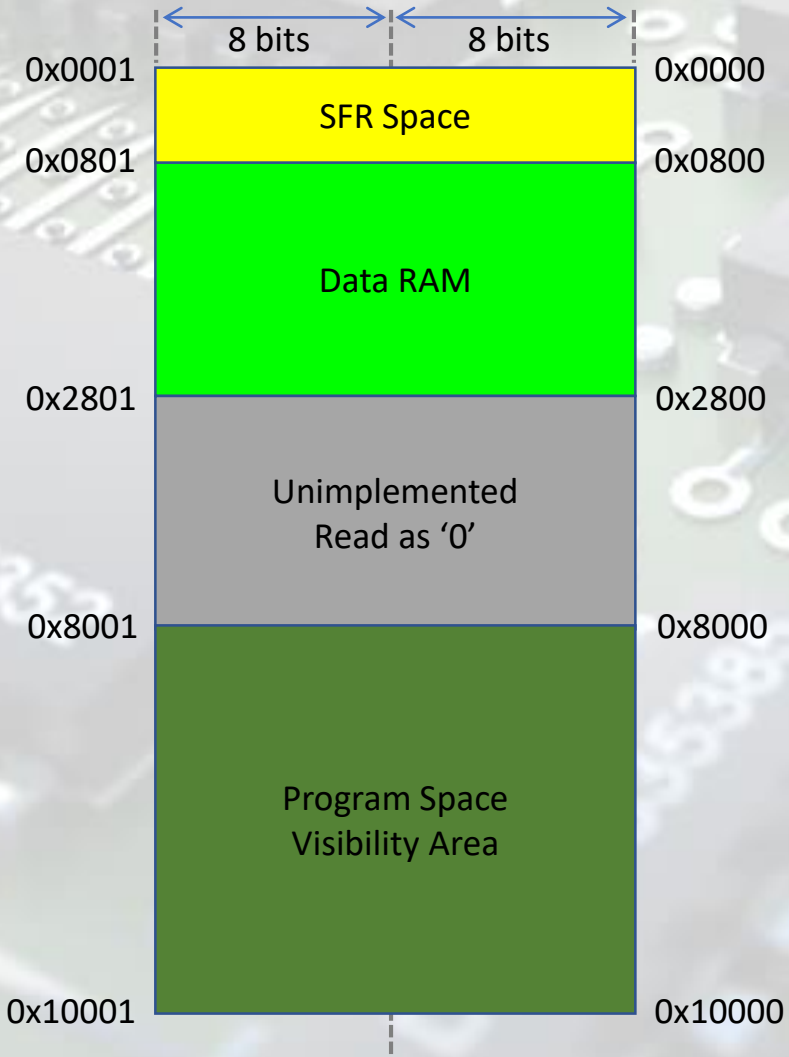
Near Data Space

All variables will be here

The upper **32 Kbytes** of data space may optionally be mapped into any **16K word page** of the program space.

The location of the **program memory space** to be mapped into the **data space** is determined by the Program Space Visibility Page Address register (PSVPAG).

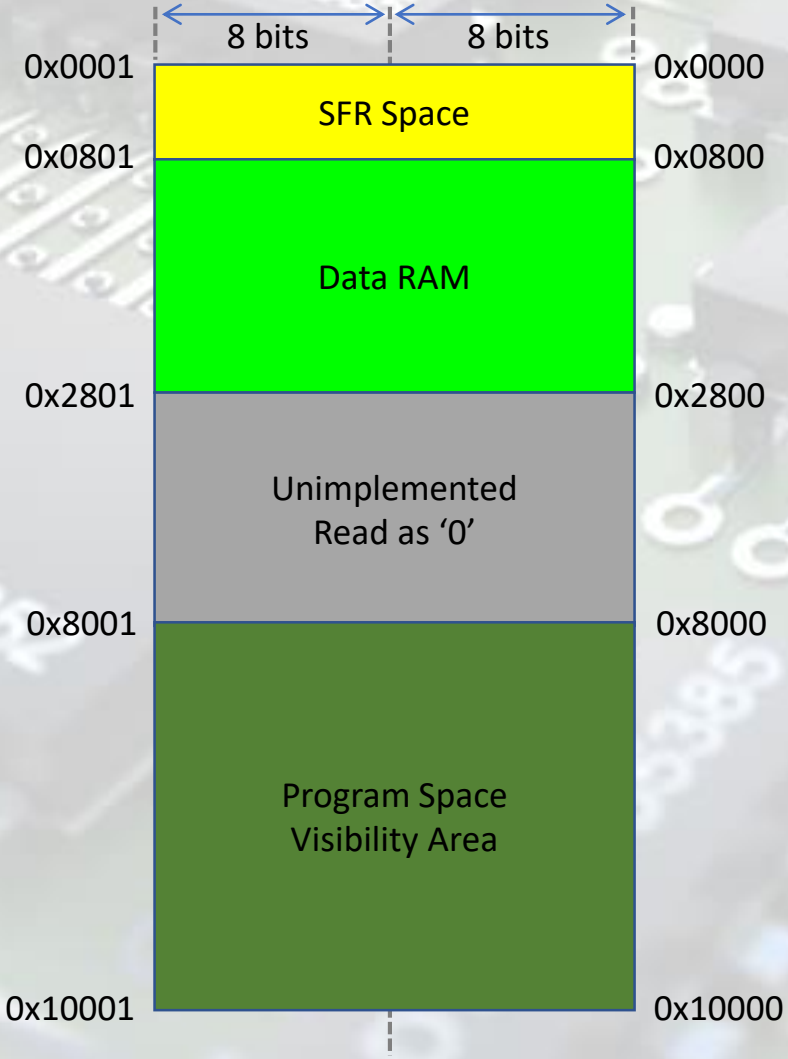
Memory and Variables (PIC2FJxxx)



Data Memory Organization and Alignment

- Data memory and registers are organized as **two parallel**, byte-wide entities with shared (word) address decode but **separate write lines**.
- All word accesses must be aligned to an **even address**. Misaligned word data fetches are not supported, so care must be taken when mixing byte and word operations.
- All byte loads into any **W register** are loaded into the **Least Significant Byte**. The **Most Significant Byte is not modified**.

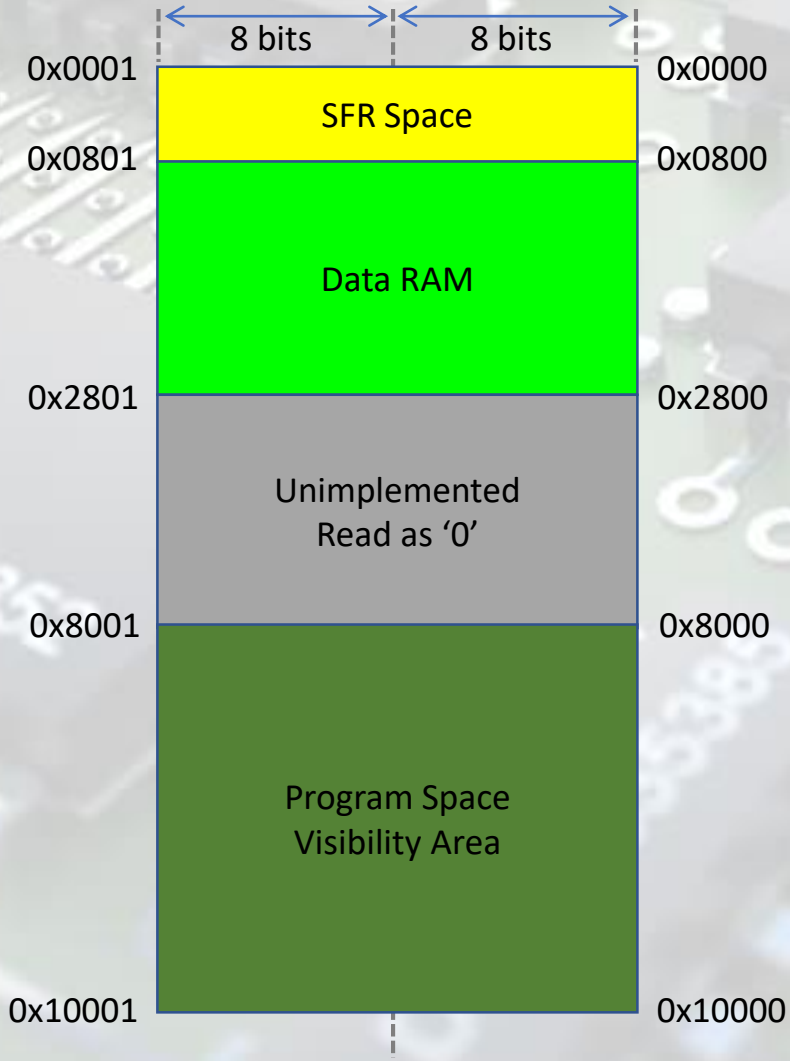
Memory and Variables (PIC2FJxxx)



Near Data Space

- The **8-Kbyte** area between **0x0000** and **0x1FFF** (**0x27FF**) is referred to as the near data space.
- Locations in this space are directly addressable via a **13-bit absolute address** field within all memory **direct** instructions.
- The remainder of the data space is addressable **indirectly**.

Memory and Variables (PIC2FJxxx)

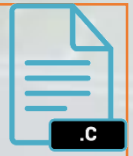


SFR Space

- The first **2 Kbytes** of the near data space, from **0x0000** to **0x07FF**, are primarily occupied with Special Function Registers (**SFRs**).
- These are used by the **PIC24F core** and **peripheral modules** for controlling the operation of the device.

C-Program and Machine-Code (*.HEX)

```c
#include <24FJ48GA002.H>
char x;
int y;
void main(void) {
    x = 5;
    y = 0x7F03;
    while(TRUE) {
        x++;
        y++;
    }
}
```




```
:080000000020400000000000F2
:100400000F782200F07F220020A0B700000000003B
:1004100081E0A8002CA3EF0050C0B30000E8B700B3
:1004200034F027001440880000 68EC000228EC003B
:08043000FDFF37000040FE0053
:020000040001F9
:0807F800F7FF00FF7F3F00FF47
:00000001FF
```

C-Program and Assembly Code

```c
#include <24FJ48GA002.H>
char x;
int y;
void main(void) {
    x = 5;
    y = 0x7F03;
    while(TRUE) {
        x++;
        y++;
    }
}
```

```
1   0000
    0000 040200 000000 GOTO 0x000200
2   ----               #include <24FJ48GA002.H>
3   ----
4   ----               char x;
5   ----               int  y;
6   0200               void main(void)
    0200 22780F        MOV.W #0x2780, W15
    0202 227FF0        MOV.W #0x27FF, W0
    0204 B7A020        MOV.W WREG, SPLIM
    0206 000000        NOP
    0208 A8E081        BSET.W INTCON1, #NSTDIS
    020A EFA32C        SETM.W AD1PCFG, F
7   ----               {
8   020C                   x = 5;
    020C B3C050        MOV.B #0x05, W0
    020E B7E800        MOV.B WREG, 0x0800
9   0210                   y = 0x7F03;
    0210 27F034        MOV.W #0x7F03, W4
    0212 884014        MOV.W W4, 0x0802
10  ----                   while(TRUE)
11  ----                   {
12  0214                       x++;
    0214 EC6800        INC.B 0x0800, F
13  0216                       y++;
    0216 EC2802        INC.W 0x0802, F
    0218 37FFFD        BRA 0x000214
14  ----                   }
15  ----               }
16  021A
```

C-Program and Assembly Code

The First Instruction at address 0x000000, the GOTO

```
1   0000
    0000 040200 000000 GOTO 0x000200
2   ----               #include <24FJ48GA002.H>
```

CPU starts at address 0x0000. At this address the CPU executes the GOTO instruction, "GOTO 0x000**200**" (main function in C)

00 04 02 00

Program Memory

00000000	00	02	04	00	00	00	00	00	FF	FF	FF	00	FF	FF	FF	00
00000010	FF	FF	FF	00	FF	FF	FF	00	FF	FF	FF	00	FF	FF	FF	00
00000020	FF	FF	FF	00	FF	FF	FF	00	FF	FF	FF	00	FF	FF	FF	00
00000030	FF	FF	FF	00	FF	FF	FF	00	FF	FF	FF	00	FF	FF	FF	00

```
6   0200                void main(void)
    0200 22780F         MOV.W #0x2780, W15
    0202 227FF0         MOV.W #0x27FF, W0
    0204 B7A020         MOV.W WREG, SPLIM
    0206 000000         NOP
    0208 A8E081         BSET.W INTCON1, #NSTDIS
    020A EFA32C         SETM.W AD1PCFG, F
```

C-Program and Assembly Code

The variables in Data Memory, **x** (1 byte) and **y** (2 bytes)

```
 7   ----                {
 8   020C                    x = 5;
     020C B3C050         MOV.B #0x05, W0
     020E B7E800         MOV.B WREG, 0x0800
 9   0210                    y = 0x7F03;
     0210 27F034         MOV.W #0x7F03, W4
     0212 884014         MOV.W W4, 0x0802
10   ----                    while(TRUE)
11   ----                    {
12   0214                        x++;
     0214 EC6800         INC.B 0x0800, F
13   0216                        y++;
     0216 EC2802         INC.W 0x0802, F
     0218 37FFFD         BRA 0x000214
14   ----                    }
15   ----                }
16   021A
```

Move 5 to W0
Move WREG to **0x0800** (Data Memory)

Move 0x7F03 to W4
Move W4 to **0x0802** (Data Memory)

Increment value @ **0x0800** (**x**) by 1

Increment value @ **0x0802** (**y**) by 1
Jumps to **0x000214**

x variable at 0x800

y variable at 0x802

Data Memory

The first address of Data Memory (RAM)

0800	05	00	03	7F	00	00	00	00	00	00	00	00	00	00	00	00
0810	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0820	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0830	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

C-Program and Data Memory

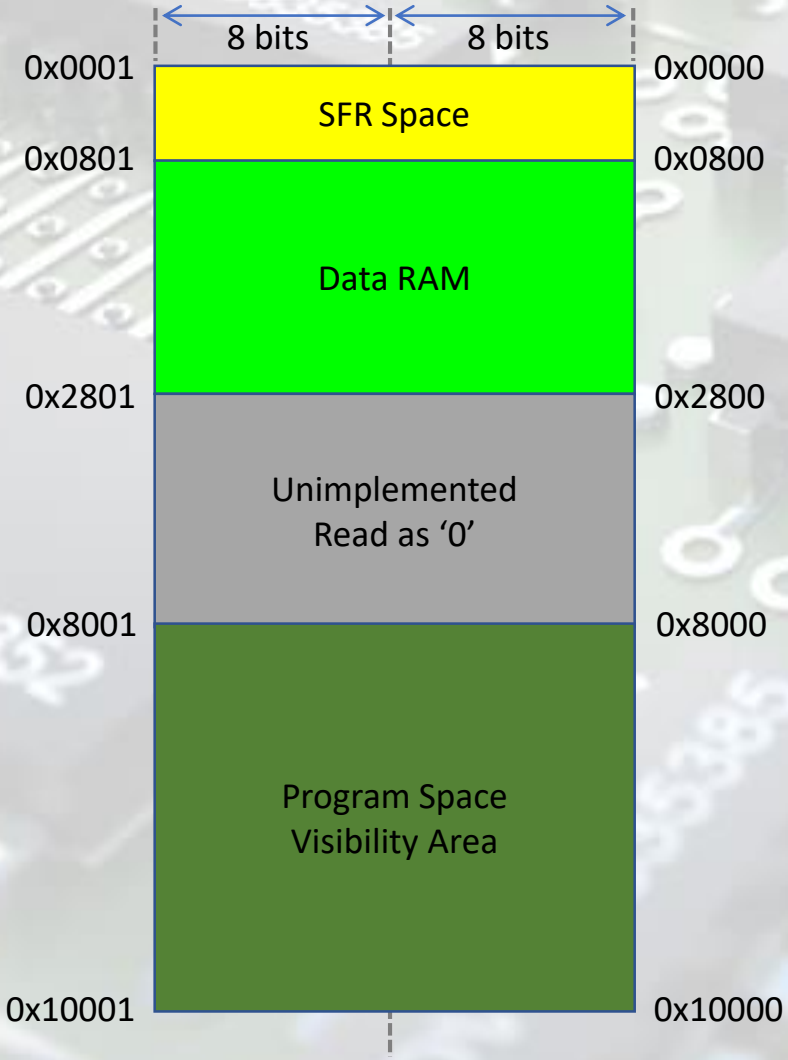
Address, **Size** (in bytes) and **Value** of Variable

```
char   a = 123;
int    b = 5432;
float  c = 456.789;
double d = 2345670.12345;

void main(void) {
    printf("a: Addr=0x%4X, Size=%d, Data=%d\r\n", &a, sizeof(a), a);
    printf("b: Addr=0x%4X, Size=%d, Data=%d\r\n", &b, sizeof(b), b);
    printf("c: Addr=0x%4X, Size=%d, Data=%e\r\n", &c, sizeof(c), c);
    printf("d: Addr=0x%4X, Size=%d, Data=%e\r\n", &d, sizeof(d), d);
    while(TRUE);
}
```

0x0801	0	0x7B	0x0800	a
0x0803	0x15	0x38	0x0802	b
0x0805	0x64	0xFE	0x0804	c
0x0807	0x43	0xE4	0x0806	
0x0809	0x35	0xA8	0x0808	d
0x080B	0x0F	0xCD	0x080A	
0x080D	0xE5	0x63	0x080C	
0x080F	0x41	0x41	0x080E	

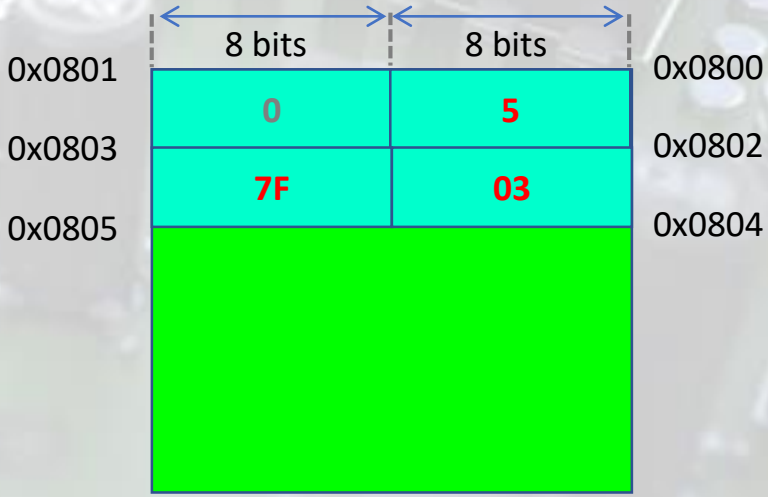
a: Addr=0x0800, Size=1, Data=123
b: Addr=0x0802, Size=2, Data=5432
c: Addr=0x0804, Size=4, Data=4.5678E+02
d: Addr=0x0808, Size=8, Data=2.3456E+06

C-Program and Data Memory




```
char x = 5;
```

```
int y = 0x7F03;
```



- In C Programming, compiler allocates the addresses for the variables.
- Because of the memory is 2 bytes aligned, therefore 0x0801 is not used.

C-Program and Data Memory

printMemHex() prints the address and value of the variables

```
void printMemHex(void) {
    int i;
    int N = sizeof(a) + sizeof(b);
    N = N + sizeof(c) + sizeof(d);
    unsigned char *ptr = 0x0800;
    for(i=0; i<N; i+=2) {
        printf("0x%4X ", ptr+1);
        printf("|0x%2X 0x%2X | ", *(1+ptr), *ptr);
        printf("0x%4X\r\n", ptr);
        ptr+=2;
    }
}
```

```
char   a = 123;
int    b = 5432;
float  c = 456.789;
double d = 2345670.12345;
```

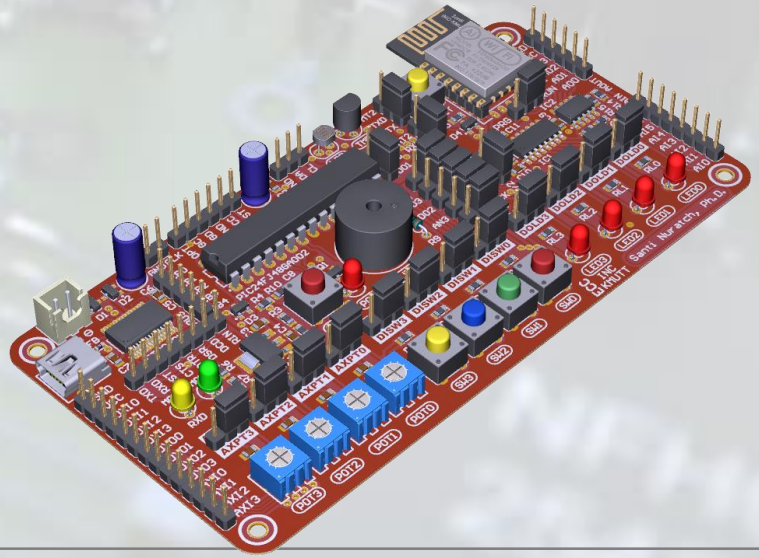
```
a: Addr=0x0800, Size=1, Data=123
b: Addr=0x0802, Size=2, Data=5432
c: Addr=0x0804, Size=4, Data=4.5678E+02
d: Addr=0x0808, Size=8, Data=2.3456E+06
```

0x0801	0	0x7B	0x0800	a
0x0803	0x15	0x38	0x0802	b
0x0805	0x64	0xFE	0x0804	c
0x0807	0x43	0xE4	0x0806	
0x0809	0x35	0xA8	0x0808	d
0x080B	0x0F	0xCD	0x080A	
0x080D	0xE5	0x63	0x080C	
0x080F	0x41	0x41	0x080E	

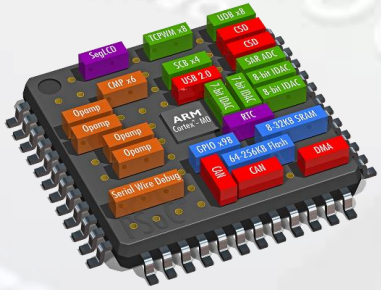
```
0x0801  |0x00  0x7B |  0x0800
0x0803  |0x15  0x38 |  0x0802
0x0805  |0x64  0xFE |  0x0804
0x0807  |0x43  0xE4 |  0x0806
0x0809  |0x35  0xA8 |  0x0808
0x080B  |0x0F  0xCD |  0x080A
0x080D  |0xE5  0x63 |  0x080C
0x080F  |0x41  0x41 |  0x080E
```

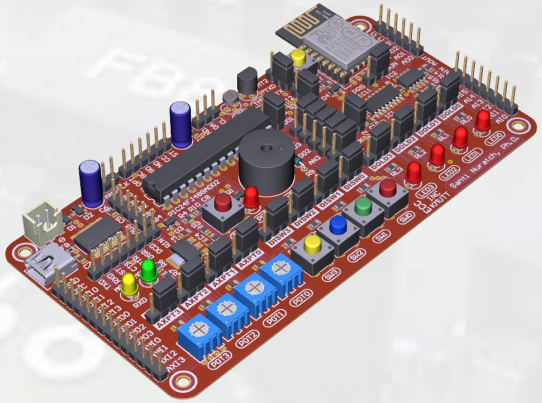
Let's Coding…


```
543  int main(void)
544  {
545      OS_Init();
546      AT_Init();
547      ESP_Init();
548      WiFi_Init();
549      Internet_Init();
550      OS_TimerCreate("AT_Service", 100, 1, AT_Service);
551      OS_WorkerCreate("WiFi_Init", Worker_ESPInitialise);
552      OS_Uart2SetLineReceivedCallback(ESP_LineReceived);
553      UART1_AsyncWriteString("\r\nMQTT Client...\r\n");
554      Beep(100);
555      OS_Start();
556  }
557
```



THANK YOU!





Asst.Prof.Dr.Santi Nuratch

Embedded Computing and Control Lab. @ INC-KMUTT

santi.inc.kmutt@gmail.com

Department of Control System and Instrumentation Engineering,
King Mongkut's University of Technology Thonburi, **KMUTT**