Microprocessor – Microcontroller

201 W03 – Adding Structure To Your Code

Nguyen Tran Huu Nguyen

D: Computer Engineering

E: nthnguyen@hcmut.edu.vn

What we will do in this lecture

- Describe how to use an object-oriented style of programming with C programs
  - allowing the creation of libraries of code that can be easily for use in different embedded projects
- Describe how to create and use a 'Project Header' file.
  - This file encapsulates key aspects of the hardware environment, such as the type of processor to be used, the oscillator frequency, and the number of oscillator cycles required to execute each instructions.
  - This helps to document the system, and make it easier to port the code to a different processor
- Describe how to create and use a 'Port Header' file.
  - This brings together all details of the port access from the whole system. Like project header, this helps during porting and also serves as a means of documenting important system features

Object Oriented Programming with C

Language generation	Example languages
-	Machine Code
First-Generation Language (1GL)	Assembly Language.
Second-Generation Languages (2GLs)	COBOL, FORTRAN
Third-Generation Languages (3GLs)	C, Pascal, Ada 83
Fourth-Generation Languages (4GLs)	C++, Java, Ada 95

Graham notes[1]:

> *"[The phrase] 'object-oriented' has become almost synonymous with modernity, goodness and worth in information technology circles."*

Jalote notes[2]:

> *"One main claimed advantage of using object orientation is that an OO model closely represents the problem domain, which makes it easier to produce and understand designs."*

---

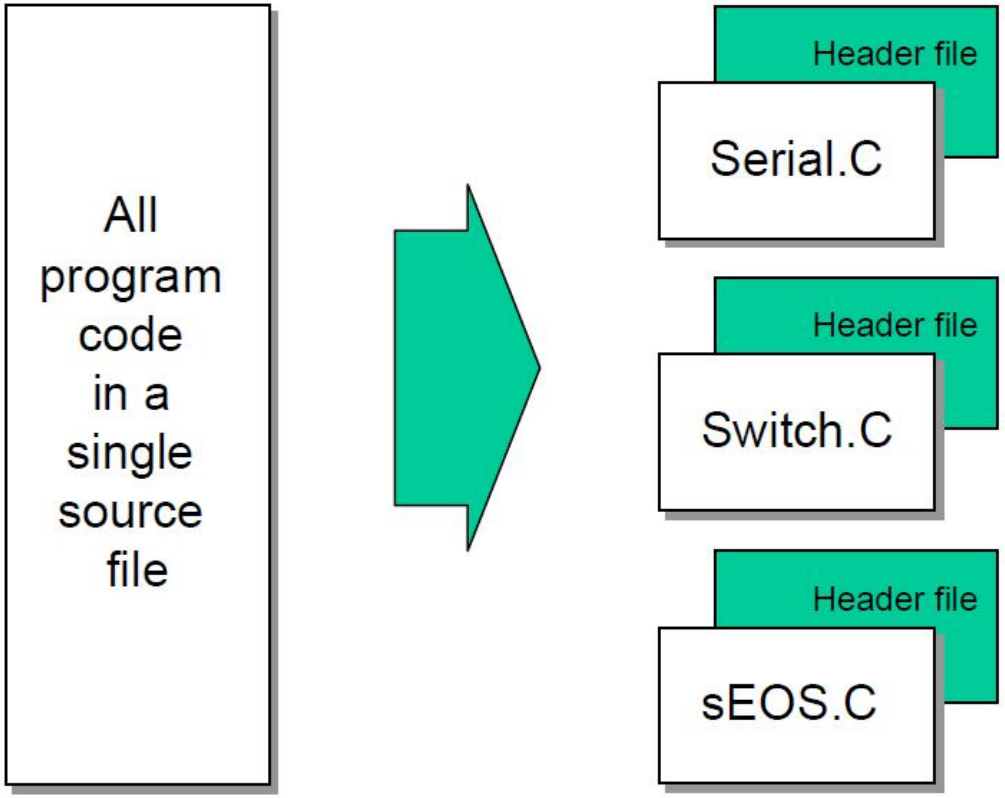
1 **Graham, I.** (1994) *"Object-Oriented Methods,"* (2nd Ed.) Addison-Wesley. Page

2 **Jalote, P.** (1997) *"An Integrated Approach to Software Engineering"*, (2nd Ed.) Springer-Verlag. Page 273.

O-O languages are not readily available for small embedded systems, primarily because of the overheads that can result from the use of some of the features of these languages.

Object Oriented Programming with C

It is possible to create ‘file-based classes’ in C without imposing a significant memory or CPU load.



C Program Layout

.c files

1. Header comment
2. #included files
3. #defines
4. local struct typedefs
5. local prototypes
6. global vars
7. main function (if present)
8. local functions

.h files

1. Header comment
2. #ifndef guard
3. #included files
4. #defines
5. struct typedefs
6. prototypes
7. (extern) global vars

Example of 'O-O C' – PC_IO.h

```
/*-------------------------------------------------------------*-

   PC_IO.H (v1.00)

  ----------------------------------------------------------

   - see PC_IO.C for details.

-*-------------------------------------------------------------*/

#ifndef _PC_IO_H
#define _PC_IO_H

/* ------ Public constants ---------------------------------- */

/* Value returned by PC_LINK_Get_Char_From_Buffer if no char is
   available in buffer */
#define PC_LINK_IO_NO_CHAR 127

/* ------ Public function prototypes ------------------------ */

void PC_LINK_IO_Write_String_To_Buffer(const char* const);
void PC_LINK_IO_Write_Char_To_Buffer(const char);

char PC_LINK_IO_Get_Char_From_Buffer(void);

/* Must regularly call this function... */
void PC_LINK_IO_Update(void);

#endif

/*-------------------------------------------------------------*-
  ---- END OF FILE ------------------------------------------
-*-------------------------------------------------------------*/
```

PC_IO.c

```
/*-------------------------------------------------------------*-

   PC_IO.C (v1.00)

  ------------------------------------------------------------

   [INCOMPLETE - STRUCTURE ONLY - see EC Chap 9 for complete library]

-*-------------------------------------------------------------*/

#include "Main.H"
#include "PC_IO.H"

/* ------ Public variable definitions ------------------------ */

tByte In_read_index_G;     /* Data in buffer that has been read */
tByte In_waiting_index_G;  /* Data in buffer not yet read */

tByte Out_written_index_G; /* Data in buffer that has been written */
tByte Out_waiting_index_G; /* Data in buffer not yet written */

/* ------ Private function prototypes ------------------------ */

static void PC_LINK_IO_Send_Char(const char);

/* ------ Private constants ---------------------------------- */

/* The receive buffer length */
#define RECV_BUFFER_LENGTH 8

/* The transmit buffer length */
#define TRAN_BUFFER_LENGTH 50

#define XON  0x11
#define XOFF 0x13

/* ------ Private variables ---------------------------------- */

static tByte Recv_buffer[RECV_BUFFER_LENGTH];
static tByte Tran_buffer[TRAN_BUFFER_LENGTH];
```

PC_IO.c (cont)

```
/*-------------------------------------------------------------*/
void PC_LINK_IO_Update(...)
    {
    ...
    }


/*-------------------------------------------------------------*/
void PC_LINK_IO_Write_Char_To_Buffer(...)
    {
    ...
    }

/*-------------------------------------------------------------*/
void PC_LINK_IO_Write_String_To_Buffer(...)
    {
    ...
    }

/*-------------------------------------------------------------*/
char PC_LINK_IO_Get_Char_From_Buffer(...)
    {
    ...
    }

/*-------------------------------------------------------------*/
void PC_LINK_IO_Send_Char(...)
    {
    ...
    }
```

Project Heade

- Includes all header files that are used
- Defines operating frequency
- Depends on the project

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f1xx_hal.h"
#include "stm32f1xx_nucleo.h"

#include "stdio.h"
#include "string.h"

typedef enum
{
  ABNORMAL = 0,
  NORMAL = !ABNORMAL
} WorkingStatus;

#define DEBUG_INIT(X)                   //X


/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* Private defines -----------------------------------------------------------*/
#define     VERSION_EBOX                1
#define     INTERRUPT_TIMER_PERIOD      10 //ms
#define     WATCHDOG_ENABLE             1
//#define       SIM5320                     1
#define     SIM7600                     1

#define LED2_PIN                         GPIO_PIN_2
#define LED2_GPIO_PORT                   GPIOB

//LED output control signals
#define LED_SDI                         GPIO_PIN_6
#define LED_SDI_PORT                    GPIOC
#define LED_SCK                         GPIO_PIN_3
#define LED_SCK_PORT                    GPIOC
#define LED_LE                          GPIO_PIN_4
#define LED_LE_PORT                     GPIOC
#define LED_OE                          GPIO_PIN_5
#define LED_OE_PORT                     GPIOC

//BUZZER
#define PB5_BUZZER_PIN                  GPIO_PIN_5
#define PB5_BUZZER_PORT                 GPIOB
#define BUZZER_PIN                      PB5_BUZZER_PIN
#define BUZZER_PORT                     PB5_BUZZER_PORT

//SPI CS pin
#define SPI_CS_PIN                      GPIO_PIN_12
#define SPI_CS_PORT                     GPIOB

```

Project Header file

```
#ifndef _MAIN_H
#define _MAIN_H

/*-------------------------------------------------------
   WILL NEED TO EDIT THIS SECTION FOR EVERY PROJECT
  ------------------------------------------------------- */

/* Must include the appropriate microcontroller header file here */
#include <reg52.h>

/* Oscillator / resonator frequency (in Hz) e.g. (11059200UL) */
#define OSC_FREQ (12000000UL)

/* Number of oscillations per instruction (12, etc)
   12 - Original 8051 / 8052 and numerous modern versions
    6 - Various Infineon and Philips devices, etc.
    4 - Dallas 320, 520 etc.
    1 - Dallas 420, etc. */
#define OSC_PER_INST (12)

/* -------------------------------------------------------
   SHOULD NOT NEED TO EDIT THE SECTIONS BELOW
  ------------------------------------------------------- */

/* Typedefs (see Chap 5)    */
typedef unsigned char tByte;
typedef unsigned int  tWord;
typedef unsigned long tLong;

/* Interrupts (see Chap 7)    */
#define INTERRUPT_Timer_0_Overflow 1
#define INTERRUPT_Timer_1_Overflow 3
#define INTERRUPT_Timer_2_Overflow 5

#endif
```

Common data types

```
typedef unsigned char tByte;
typedef unsigned int tWord;
typedef unsigned long tLong;
```

In C, the typedef keyword allows us to provide aliases for data types: we can then use these aliases in place of the original types.

For example, we use

`tWord Temperature;` to declare a variable rather than use `unsigned int Temperature;`

Common data types

- The main reason for using **typedef** keyword is to simplify and promote the use of unsigned data types.
  - Most MCUs do not support signed arithmetic and extra code is required to manipulate signed data → this reduces your program speed and increases the program size.
  - Use of bitwise operators generally makes sense only with unsigned data types: use of **typedef** variable reduces the likelihood that programmers will inadvertently apply these bitwise operators to signed data.
  - Use of the **typedef** keyword can make it easier to adapt your code for use on a different processor.

Why use the Project Header?

- Use of **Project header** can help to make your code more readable
  - Anyone using your projects knows where to find key information, such as the model of microcontroller and the oscillator frequency required to execute the software.
- The use of a project header can help to make your code more easily portable, by placing some of the key MCU-dependent data in one place
  - If you want to change the processor of the oscillator used, then in many cases- you will need to make changes only to the Project Header

The Port Header File

- Consider, for example, that we have three files in a project (A, B and C), each of which require access to one or more port pins or to a complete port
- All the port access requirements are spread over multiple files
- It is better to intergrade all port access in single port header file

File A may include the following:

```
/* File A */

sbit Pin_A = P3^2;

...
```

File B may include the following:

```
/* File B */

#define Port_B = P0;

...
```

File C may include the following:

```
/* File C */

sbit Pin_C = P2^7;
```

Port Header File

- Includes all Pins that are used
- Depends on the project and the MCU used

```
193
194 /* Definition for SPIx Pins */
195 #define SPI2_NSS_PIN                GPIO_PIN_12
196 #define SPI2_NSS_GPIO_PORT          GPIOB
197
198 #define SPI2_SCK_PIN                GPIO_PIN_13
199 #define SPI2_SCK_GPIO_PORT          GPIOB
200 #define SPI2_MISO_PIN               GPIO_PIN_14
201 #define SPI2_MISO_GPIO_PORT         GPIOB
202 #define SPI2_MOSI_PIN               GPIO_PIN_15
203 #define SPI2_MOSI_GPIO_PORT         GPIOB
204
205
206
```

```
215
    /* Definition for I2Cx Pins */
    #define I2C1_SCL_PIN                GPIO_PIN_6
    #define I2C1_SCL_GPIO_PORT          GPIOB
    #define I2C1_SDA_PIN                GPIO_PIN_7
    #define I2C1_SDA_GPIO_PORT          GPIOB
```

```
265 //Relay control pins and ports
266 #define PA11_OUT0                   GPIO_PIN_11
267 #define PA11_OUT0_PORT              GPIOA
268 #define PA12_OUT1                   GPIO_PIN_12
269 #define PA12_OUT1_PORT              GPIOA
270 #define PB8_OUT2                    GPIO_PIN_8
271 #define PB8_OUT2_PORT               GPIOB
272 #define PB9_OUT3                    GPIO_PIN_9
273 #define PB9_OUT3_PORT               GPIOB
274 #define PA15_OUT4                   GPIO_PIN_15
275 #define PA15_OUT4_PORT              GPIOA
276
277 #define PC10_OUT5                   GPIO_PIN_10
278 #define PC10_OUT5_PORT              GPIOC
279 #define PC11_OUT6                   GPIO_PIN_11
280 #define PC11_OUT6_PORT              GPIOC
281 #define PC12_OUT7                   GPIO_PIN_12
282 #define PC12_OUT7_PORT              GPIOC
283
284 #define PB3_OUT8                    GPIO_PIN_3
285 #define PB3_OUT8_PORT               GPIOB
286
287 #define PB4_OUT9                    GPIO_PIN_4
288 #define PB4_OUT9_PORT               GPIOB
```

Restructuring the Goat-counting Example