Appendix F: Advanced C Programming

Sepehr Naimi

www.NicerLand.com

In this Appendix, first we introduce some preprocessor directives, and then we explain how to manipulate registers using the defined bits.

Section F.1: Preprocessor Directives

#define

#define can be used to define constant values like PI:

```
#define PI 3.1415
```

When the above line of code is written, the compiler replaces PI with 3.1415 at compile time. For example, after defining the PI constant if we write “float a = PI*2.0;”, at compile time, the PI will be considered as 3.1415 and then it will be compiled.

Example F-1: Find the values for variables a and b in the following program:

```
#define NUM_OF_LINES       4
int main ( )
{
   int a = NUM_OF_LINES;
   int b = NUM_OF_LINES * 20;
   while(1);
}
```

Solution:

NUM_OF_LINES is replaced with 4 and the above program will be converted to:

```
int main ( )
{
   int a = 4;
   int b = 4 * 20;
   while(1);
}
```

So, a and b will contain 4 and 80, respectively.

Using #define we can define nickname for any expression. For example, in the following program, we name { and } as BEGIN and END respectively:

```
#define BEGIN  {
#define END    }

int main ()
BEGIN
   while(1)
   BEGIN
      while(1);
   END
END
```

Naming pins

It is a good practice to name pins of microcontrollers according to their usages. This makes the code more readable and also more reusable. For example, see the following snippet of code:

```
#define LED_PORT    GPIOC
#define LED_BIT     13
int main(){
   ...
   LED_PORT->ODR |= (1<<LED_BIT);
}
```

#define and Macros

#define can be used to define macros in C language. For example, in the following piece of code, a macro is defined that sets a bit of a variable:

```
#define SET_BIT(varName,bitNum) varName |= (1<<bitNum)
int main()
{
   ...
   SET_BIT(GPIOC->ODR,13); /* set bit 13 of GPIOC->ODR */
   while(1);
}
```

At compile time, SET_BIT(GPIOC->ODR,13) is replaced with "varName = varName|(1<<bitNum)" and the arguments varName and bitNum are substituted with GPIOC->ODR and 13, respectively. So, the line of code is replaced with "GPIOC->ODR |= (1<<13);".

See also the following snippet of code:

```
#define SQUARE(x)          x*x
int main ( )
{
  GPIOB->ODR = SQUARE(GPIOA->IDR);
  ...
}
```

In the above program, the compiler substitutes "GPIOB->ODR = SQUARE(GPIOA->IDR);" with "GPIOB->ODR = GPIOA->IDR*GPIOA->IDR;" and then converts it to machine code.

Note: macros vs. functions

Macros do not have the function call overhead and do not use the stack space, neither. But they can increase the used program memory. It is good to use macros (or inline functions) when the function is too small.

#undef

#undef is opposite of #define. It undefines a symbol which was defined by #define:

```
#undef symbol
```

For example, in the following program we undefine MAX_VALUE which is defined in a few lines before:

```
#define MAX_VALUE 100
...
#undef MAX_VALUE
```

Conditional preprocessors

The conditional preprocessor has the following format:

```
#if condition
...  //The program that should be compiled when the condition is met
```

```
#endif
```

In the following program, the #if preprocessor checks the value of DISPLAY_CONFIG and the printf lines will be compiled only when DISPLAY_CONFIG is 1. Since DISPLAY_CONFIG is defined 0, the printf lines are not be compiled, now.

```
#define DISPLAY_CONFIG 0
int main ( )
{
   #if DISPLAY_CONFIG == 1
      printf("Hello World!\n\r");
      printf("Now Display config is set to 1.");
   #endif
}
```

Together with conditional preprocessor, we can use #else as well:

```
#if condition
 ...  /* The code that should be compiled when the condition is met */
#else
 ...  /* The code will be compiled when the condition is not met */
#endif
```

In the following program i is defined as uint8_t when ARRAY_LEN is defined less than 256:

```
#define ARRAY_LEN 64
int main ( )
{
   char ourArray[ARRAY_LEN];
#if ARRAY_LEN < 256
      uint8_t i = 0;
#else
      uint16_t i = 0;
#endif

   for (i = 0; i < ARRAY_LEN; i++)
      ourArray[i] = ' ';
}
```

Instead of #if we can use #ifdef or #ifndef directives, as well. We can check to see if a macro is defined or not using #ifdef and #ifndef respectively. For example, in the following code the printf line will be compiled only when DISPLAY_RESULTS is defined:

```
int main ()
{
   #ifdef DISPLAY_RESULTS
   printf("The result is as follows");
   #endif
}
```

Section F.2: Manipulating Registers Using Defined Bit Names

In Section 8-1, you learned to enable the clocks for different peripherals. For example, the following instruction enables the clock for GPIO port B:

```
RCC->APB2ENR |= (1<<3); /* set the IOPBEN bit to 1 (IOPBEN is bit 3 of APB2ENR) */
```

To write the above line of code, you have to check the book (or the reference manual of the chip) to see that bit 3 of the APB2ENR is for IOPBEN. Then, you can write the code.

In the “stm32f10x.h” header file, there are definitions for all bits of the I/O registers. For example, the bits of APB2ENR register are defined as shown below:

```
/******************  Bit definition for RCC_APB2ENR register  *****************/
#define RCC_APB2ENR_AFIOEN  ((uint32_t)0x00000001)  /*!< Alternate Function I/O clock
enable */
#define RCC_APB2ENR_IOPAEN  ((uint32_t)0x00000004)  /*!< I/O port A clock enable */
#define RCC_APB2ENR_IOPBEN  ((uint32_t)0x00000008)  /*!< I/O port B clock enable */
#define RCC_APB2ENR_IOPCEN  ((uint32_t)0x00000010)  /*!< I/O port C clock enable */
#define RCC_APB2ENR_IOPDEN  ((uint32_t)0x00000020)  /*!< I/O port D clock enable */
#define RCC_APB2ENR_ADC1EN  ((uint32_t)0x00000200)  /*!< ADC 1 interface clock enable
*/
...
```

So, to enable the clock for GPIO port B you can also write the following instruction:

```
RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;
```

Similarly, the following line of code, enables the clock for ADC1:

```
RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;
```

The bit definition naming follows the following convention:

peripheralName_registerName_bitName

For example, the BR5 bit for GPIOA->BSRR is named as GPIO_BSRR_BR5. So, the following line of code, sets PA4 and clears PA5:

```
GPIOA->BSRR = GPIO_BSRR_BS4|GPIO_BSRR_BR5;
```

As another example, the following snippet of code toggles PC13:

```
   while(1)
   {
      GPIOC->BRR = GPIO_BRR_BR13; /* make the pin low */
      delay_ms(500);              /* wait 0.5 sec. */
      GPIOC->BSRR = GPIO_BSRR_BS13; /* make the pin high */
      delay_ms(500);              /* wait 0.5 sec. */
   }
```

In cases that a field of a register is bigger than 1 bit, the field is named as peripheralName_registerName_FieldName. To name the bits of the field, the bit number is added to the end of field name. For example, in the CRL and CRH registers, the MODEx fields are 2-bit. So, the fields are named as GPIO_CRL_MODEx and the bits are named as GPIO_CRL_MODEx_0 and GPIO_CRL_MODEx_1. For example, in STM32F10x.h the followings are defined for MODE6:

```
#define  GPIO_CRL_MODE6  ((uint32_t)0x03000000)   /*!< MODE6[1:0] bits (Port x mode
bits, pin 6) */
#define  GPIO_CRL_MODE6_0  ((uint32_t)0x01000000)        /*!< Bit 0 */
#define  GPIO_CRL_MODE6_1  ((uint32_t)0x02000000)        /*!< Bit 1 */
```

The following piece of code sets MODE6 to 2 using the bit definitions. It clears the MODE6 bits using AND and then, sets the MODE6_1 bit:

```
GPIOA->CRL &= ~GPIO_CRL_MODE6;
GPIOA->CRL |= GPIO_CRL_MODE6_1;
```

See the following programs.

Program F-1: Rewrite Program 8-5 using bit definitions (Toggle PC13 using BSRR and BRR)

```
#include <stm32f10x.h>

void delay_ms(uint16_t t);

int main()
{
  RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;          /* Enable clocks for GPIOs */
  /* PC13 as output */
  GPIOC->CRH &= (GPIO_CRH_MODE13|GPIO_CRH_CNF13); /* clear MODE13 and CNF13 fields */
  GPIOC->CRH |=GPIO_CRH_MODE13_1|GPIO_CRH_MODE13_0; /* set MODE13 to 3 (Output) */

  while(1)
  {
    GPIOC->BRR = GPIO_BRR_BR13; /* make the pin low */
    delay_ms(500);              /* wait 0.5 sec. */
    GPIOC->BSRR = GPIO_BSRR_BS13;      /* make the pin high */
    delay_ms(500);              /* wait 0.5 sec. */
  }
}
// copy delay_ms from Program 8-1
```

Program F-2: Rewrite Program 8-9 using bit definitions (It toggles PC13 if PA2 is low)

```
#include <stm32f10x.h>

void delay_ms(uint16_t t);

int main()
{
  RCC->APB2ENR |= RCC_APB2ENR_IOPAEN|RCC_APB2ENR_IOPCEN; /* Enable clocks for PB and 
PC*/
  /* PC13 as output */
  GPIOC->CRH = (GPIOC->CRH&~GPIO_CRH_CNF13)|GPIO_CRH_MODE;

  /* PA2 as input with pull-up */
  GPIOA->CRL &= ~(GPIOA->CRL&GPIO_CRL_CNF2);
  GPIOA->CRL |= GPIO_CRL_CNF2_1;
  GPIOA->ODR |= (1<<2);   /* pull-up PA2 */

  while(1)
  {
    if((GPIOA->IDR&GPIO_IDR_IDR2) == 0) /* is PA2 low? */
      GPIOC->ODR ^= GPIO_ODR_ODR13; /* toggle PC13 */
    else
      GPIOC->ODR &= ~GPIO_ODR_ODR13;
    delay_ms(500);
```

```
   }
}
//copy delay_ms from Program 8-1 to here
```

Program F-3: Rewrite Program 8-6 using bit definitions (Reading from port B and writing to port A)

```
#include <stm32f10x.h>

int main()
{
   RCC->APB2ENR |= RCC_APB2ENR_IOPAEN|RCC_APB2ENR_IOPBEN; /* Enable clocks for PA and
PB */

   GPIOA->CRL &= ~(GPIO_CRL_CNF|GPIO_CRL_MODE); /* PA0-PA7 as outputs */
   GPIOA->CRL |= GPIO_CRL_MODE;
   /* PA8-PA15 as outputs */
   GPIOA->CRH &= ~(GPIO_CRH_CNF|GPIO_CRH_MODE);
   GPIOA->CRH |= GPIO_CRH_MODE;
   /* PB0-PB7 as inputs */
   GPIOB->CRL &= ~(GPIO_CRL_CNF|GPIO_CRL_MODE);
   GPIOB->CRL |=
GPIO_CRL_CNF0_0|GPIO_CRL_CNF1_0|GPIO_CRL_CNF2_0|GPIO_CRL_CNF3_0|GPIO_CRL_CNF4_0|GPIO_C
RL_CNF5_0|GPIO_CRL_CNF6_0|GPIO_CRL_CNF7_0;
   /* PB8-PB15 as inputs */
   GPIOB->CRH &= ~(GPIO_CRH_CNF|GPIO_CRH_MODE);
   GPIOB->CRH |=
GPIO_CRH_CNF8_0|GPIO_CRH_CNF9_0|GPIO_CRH_CNF10_0|GPIO_CRH_CNF11_0|GPIO_CRH_CNF12_0|GPI
O_CRH_CNF13_0|GPIO_CRH_CNF14_0|GPIO_CRH_CNF15_0;

   while(1)
   {
      GPIOA->ODR = GPIOB->IDR; /* read from port B and write to port A */
   }
}
```