به نام خدا

آشنایی با زبان AVR C

انواع داده

Dr. Aref Karimiafshar
A.karimiafshar@iut.ac.ir

Why program the AVR in C?

- Compilers produce hex files
  - We download it into the Flash of **μC**
  - The size of hex file is one of main concerns
    - Because of the limited on-chip Flash
- The choice of programming language affects the compiled program size
  - Assembly language produce a hex file that is much smaller than C
  - Assembly language is often tedious and time consuming
  - C programming is less time consuming and much easier to write
    - But the size is much larger

Major reasons for programming in C instead of Assembly

- ✓ It is easier and less time consuming
- ✓ Easier to modify and update
- ✓ You can use code available in function libraries
- ✓ C code is portable to other μCs with little or no modification

C Data Types for AVR C

- To create smaller hex file
  - You need, A good understanding of C data types
- Most common and widely used data types

Some Data Types Widely Used by C Compilers

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
char	8-bit	–128 to +127
unsigned int	16-bit	0 to 65,535
int	16-bit	–32,768 to +32,767
unsigned long	32-bit	0 to 4,294,967,295
long	32-bit	–2,147,483,648 to +2,147,483,648
float	32-bit	±1.175e-38 to ±3.402e38
double	32-bit	±1.175e-38 to ±3.402e38

Unsigned char

- Because the AVR is an 8-bit μC
  - The character data type is the most natural choice
- Range: 0-255 (00-FF H)
- In situations, such as setting a counter value
- We must pay careful attention to the size of data
  - Try to use unsigned char instead of int, if possible
- C compliers use the signed char as the default
  - Unless we put the keyword unsigned in front of the char
- In situations where + and – are needed to represent a given quantity such as temperature
  - The use of signed char is necessary

Unsigned int

- Unsigned int is a 16-bit data type
- Used to define 16-bit variables such as memory addresses
  - Also set counter values of more than 256
- Because AVR is an 8-bit μC, int data type takes 2 bytes of RAM
  - We must not use int data type unless we have to!
  - Misuse of int result in
    - Larger hex files
    - Slower execution
    - More memory usage
  - Such misuse is not problem in PCs (with 512 MB, bus speed 133 Mhz, …)

Other Data Types

- Unsigned int is limited to 0-65,535 (0000-FFFF H)
- For values greater than16-bit
  - AVR C compiler supports long data types
- Also to deal with fractional numbers
    - Float
    - Double

Data Types

Write an AVR C program to toggle all bits of Port B 50,000 times.

```
#include <avr/io.h>                         //standard AVR header
int main(void)
{
  unsigned int z;
  DDRB = 0xFF;                              //PORTB is output

  for(z=0; z<50000; z++)
  {
    PORTB = 0x55;
    PORTB = 0xAA;
  }

  while(1);                                 //stay here forever
  return 0;
}
```

Data Types

Write an AVR C program to toggle all bits of Port B 100,000 times.

```
//toggle PB 100,00 times
#include <avr/io.h>                //standard AVR header
int main(void)
{
  unsigned long z;                 //long is used because it should
                                   //store more than 65535.
  DDRB = 0xFF;                     //PORTB is output

  for(z=0; z<100000; z++){
    PORTB = 0x55;
    PORTB = 0xAA;
  }

  while(1);                        //stay here forever
  return 0;
}
```

Time Delay

- There are three ways to create a time delay in AVR C:
  - Using a simple for loop
  - Using predefined C functions
  - Using AVR timers
- In creating a time delay using for loop, there are two factors that can affect the accuracy of the delay:
  - The Crystal frequency
  - The Compiler used to compile the C program
    - Because it is the C compiler that convert C statements and functions to assembly language instructions
    - Different compilers produce different code
  - For the above reasons, we must use oscilloscope to measure the exact duration

Time Delay

Write an AVR C program to toggle all the bits of Port B continuously with a 100 ms delay. Assume that the system is ATmega 32 with XTAL = 8 MHz.

```
#include <avr/io.h>                    //standard AVR header
void delay100ms(void)
{
  unsigned int i;
  for(i=0; i<42150; i++);              //try different numbers on your
}                                      //compiler and examine the result.

int main(void)
{
  DDRB = 0xFF;                         //PORTB is output
  while (1)
  {
    PORTB = 0xAA;
    delay100ms();
    PORTB = 0x55;
    delay100ms();
  }
  return 0;
}
```

Predefined Functions Time Delay

- Generating time delay using predefined functions:
  - delay_ms()
  - delay_us()
- Drawback of using these functions
  - Portability problem
    - Because different compilers do not use the same name for delay functions
    - You have to change every place in which the delay functions are used
      - To compile the program on another compiler
  - To overcome the problem
    - Use macro or wrapper function
      - Wrapper functions do nothing more than call the predefined delay function
    - Instead of changing all instances of predefined delay functions, you simply change the wrapper function

Predefined Functions Time Delay

Write an AVR C program to toggle all the pins of Port C continuously with a 10 ms delay. Use a predefined delay function in Win AVR.

```
#include <util/delay.h>          //delay loop functions
#include <avr/io.h>              //standard AVR header

int main(void)
{
        void delay_ms(int d)     //delay in d microseconds
        {
                _delay_ms(d);
        }
        DDRB = 0xFF;             //PORTA is output
        while (1){
                PORTB = 0xFF;
                delay_ms(10);
                PORTB = 0x55;
                delay_ms(10);
        }
        return 0;
}
```

I/O programming in C

- To access a PORT register as a byte
  - We use the PORTx label
    - x indicates the name of the port
- To access the data direction register
  - We use DDRx label
    - x indicates the name of the port
- To access a PIN register
  - We use PINx label
    - x indicates the name of the port

I/O programming in C

LEDs are connected to pins of Port B. Write an AVR C program that shows the count from 0 to FFH (0000 0000 to 1111 1111 in binary) on the LEDs.

```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
   DDRB = 0xFF;                        //Port B is output
   while (1)
   {
      PORTB = PORTB + 1;
   }
   return 0;
}
```

I/O programming in C

Write an AVR C program to get a byte of data from Port B, and then send it to Port C.

```
#include <avr/io.h>                    //standard AVR header
int main(void)
{
   unsigned char temp;

   DDRB = 0x00;                        //Port B is input
   DDRC = 0xFF;                        //Port C is output

   while(1)
   {
      temp = PINB;
      PORTC = temp;
   }
   return 0;
}
```

I/O programming in C

Write an AVR C program to get a byte of data from Port C. If it is less than 100, send it to Port B; otherwise, send it to Port D.

```
#include <avr/io.h>                          //standard AVR header
int main(void)
{
   DDRC = 0;                                 //Port C is input
   DDRB = 0xFF;                              //Port B is output
   DDRD = 0xFF;                              //Port D is output
   unsigned char temp;
   while(1)
   {
      temp = PINC;                           //read from PINB
      if ( temp < 100 )
         PORTB = temp;
      else
         PORTD = temp;
   }
   return 0;
}
```

Bit size I/O

- I/O ports of ATmega32 are bit-accessible
  - But some AVR C compilers do not support this features
  - Also, there is no standard way of using it
- To set the first pin of Port B
  - In Code Vision, we can use

```
PORTB.0=1
```

  - But it can not be used in other compilers such as WinAVR
- To write portable code
  - We must use AND or OR bit-wise operations

Logic Operations in C

- Bit-Wise operators in C
  - Widely used in software engineering for embedded system and control

Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	**B**	**A&B**	**A|B**	**A^B**	**Y=~B**
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

The following shows some examples using the C bit-wise operators:

1. 0x35 & 0x0F = 0x05 /* ANDing */
2. 0x04 | 0x68 = 0x6C /* ORing */
3. 0x54 ^ 0x78 = 0x2C /* XORing */
4. ~0x55 = 0xAA /* Inverting 55H */

Logic Operations in C

```
#include <avr/io.h>                  //standard AVR header
int main(void)
{
   DDRB = 0xFF;                      //make Port B output
   DDRC = 0xFF;                      //make Port C output
   DDRD = 0xFF;                      //make Port D output
   PORTB = 0x35 & 0x0F;              //ANDing
   PORTC = 0x04 | 0x68;              //ORing
   PORTD = 0x54 ^ 0x78;              //XORing
   PORTB = ~0x55;                    //inverting
   while (1);
   return 0;
}
```

Logic Operations in C

Write an AVR C program to toggle only bit 4 of Port B continuously without disturbing the rest of the pins of Port B.

```
#include <avr/io.h>                    //standard AVR header

int main(void)
{
   DDRB = 0xFF;                        //PORTB is output

   while(1)
   {
      PORTB = PORTB | 0b00010000;      //set bit 4 (5th bit) of PORTB
      PORTB = PORTB & 0b11101111;      //clear bit 4 (5th bit) of PORTB
   }

   return 0;
}
```

Logic Operations in C

Write an AVR C program to monitor bit 5 of port C. If it is HIGH, send 55H to Port B; otherwise, send AAH to Port B.

```
#include <avr/io.h>              //standard AVR header

int main(void)
{
  DDRB = 0xFF;                   //PORTB is output
  DDRC = 0x00;                   //PORTC is input
  DDRD = 0xFF;                   //PORTB is output

  while(1)
  {
    if (PINC & 0b00100000)       //check bit 5 (6th bit) of PINC
      PORTB = 0x55;
    else
      PORTB = 0xAA;
  }

  return 0;
}
```

Logic Operations in C

A door sensor is connected to bit 1 of Port B, and an LED is connected to bit 7 of Por C. Write an AVR C program to monitor the door sensor and, when it opens, turn on th LED.

```
#include <avr/io.h>                   //standard AVR header

int main(void)
{
   DDRB = DDRB & 0b11111101;          //pin 1 of Port B is input
   DDRC = DDRC | 0b10000000;          //pin 7 of Port C is output

   while(1)
   {
      if (PINB & 0b00000010)          //check pin 1 (2nd pin) of PINB
         PORTC = PORTC | 0b10000000;  //set pin 7 (8th pin) of PORTC
      else
         PORTC = PORTC & 0b01111111;  //clear pin 7 (8th pin) of PORTC
   }
   return 0;
}
```

Compound Assignment Operators in C

- To reduce coding (typing)
  - We can use compound statements

Compound Assignment Operator in C

Operation	Abbreviated Expression	Equal C Expression
And assignment	a &= b	a = a & b
OR assignment	a |= b	a = a | b

```
#include <avr/io.h>            //standard AVR header
int main(void)
{
   DDRB &= 0b11011111;         //bit 5 of Port B is input
   DDRC |= 0b10000000;         //bit 7 of Port C is output

   while (1)
   {
      if(PINB & 0b00100000)
         PORTC |=  0b10000000;  //set bit 7 of Port C to 1
      else
         PORTC &= 0b01111111;   //clear bit 7 of Port C to 0
   }
   return 0;
}
```

Bit-wise Shift Operation in C

Bit-wise Shift Operators for C

Operation	Symbol	Format of Shift Operation
Shift right	>>	data >> number of bits to be shifted right
Shift left	<<	data << number of bits to be shifted left

The following shows some examples of shift operators in C:

1. 0b00010000 >> 3 = 0b00000010 /* shifting right 3 times */
2. 0b00010000 << 3 = 0b10000000 /* shifting left 3 times */
3. 1 << 3 = 0b00001000 /* shifting left 3 times */

Bit-wise Shift Operation in C

Write code to generate the following numbers:

(a) A number that has only a one in position D7
(b) A number that has only a one in position D2
(c) A number that has only a one in position D4
(d) A number that has only a zero in position D5
(e) A number that has only a zero in position D3
(f) A number that has only a zero in position D1

(a) `(1<<7)`
(b) `(1<<2)`
(c) `(1<<4)`
(d) `~(1<<5)`
(e) `~(1<<3)`
(f) `~(1<<1)`

Bit-wise Shift Operation in C

Write an AVR C program to monitor bit 7 of Port B. If it is 1, make bit 4 of Port B input; else, change pin 4 of Port B to output.

```
#include <avr/io.h>                         //standard AVR header

int main(void)
{
   DDRB = DDRB & ~(1<<7);                   //bit 7 of Port B is input

   while (1)
   {
      if(PINB & (1<<7))
         DDRB = DDRB & ~(1<<4);             //bit 4 of Port B is input
      else
         DDRB = DDRB | (1<<4);              //bit 4 of Port B is output
   }

   return 0;
}
```

پایان

موفق و پیروز باشید