AVR C Programming

Discussion IV (Version 2.0)

UMBC - CE

September 15, 2015

Version 1.0 - Initial Document
Version 2.0 - Minor updates in references

Objectives

- Review AVR I/O in C

Objectives

- Review AVR I/O in C
- Implement a demo AVR C program on the AVR Butterfly

PORTx and DDRx Review

- Summary of control signals for port pins

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Micro-controller Specific Constants/Defines

- All programs will have a line of code to include various utility functions, as well as various value definitions and processor specific definitions

```
# include <avr/io.h >
        or
# include <avr/iom169p.h >
```

Micro-controller Specific Constants/Defines

- All programs will have a line of code to include various utility functions, as well as various value definitions and processor specific definitions

```
# include <avr/io.h >
        or
# include <avr/iom169p.h >
```

- The files are located at "C:\Program Files (x86)\Atmel\Atmel Toolchain\AVR8 GCC\Native\3.4.1056\avr8-gnu-toolchain\avr\include\avr"

Setting up the Direction bits

- To set the direction of all 8 pins of port D, assign a 8-bit value to DDRD

```
DDRD=0xFF; //set all port D pins as outputs
DDRD=0x00; //set all port D pins as inputs
DDRD=0b10101010; // alternating pin directions
```

Setting up the Direction bits

- To set the direction of all 8 pins of port D, assign a 8-bit value to DDRD

```
DDRD=0xFF; //set all port D pins as outputs
DDRD=0x00; //set all port D pins as inputs
DDRD=0b10101010; // alternating pin directions
```

- To just set pin 2 of port D to output, not touching the others

```
DDRD=DDRD | 0b00000100;
Or just
DDRD |= 0b00000100; //recommended!!!!
```

Outputting values to PORTx

- Do not set all 8 bits in register PORTD without regard for the directions of each individual pin, i.e. all the bits stored in DDRD

Outputting values to PORTx

- Do not set all 8 bits in register PORTD without regard for the directions of each individual pin, i.e. all the bits stored in DDRD
- Use bit operations when possible to suggest use of I/O bit assembly commands and to avoid unintentionally setting extra bit values

```
Set one pin:
PORTD |=(1<<3);
      same as
PORTD |=(1<<PD3);
Clear one pin:
PORTD &= ~((1<<3));
      same as
PORTD &=~((1<<PD3));
```

Outputting values to PORTx

- Do not set all 8 bits in register PORTD without regard for the directions of each individual pin, i.e. all the bits stored in DDRD
- Use bit operations when possible to suggest use of I/O bit assembly commands and to avoid unintentionally setting extra bit values

```
Set one pin:
PORTD |=(1<<3);
        same as
PORTD |=(1<<PD3);
Clear one pin:
PORTD &= ~((1<<3));
        same as
PORTD &=~((1<<PD3));
```

Outputting values to PORTx

- Do not set all 8 bits in register PORTD without regard for the directions of each individual pin, i.e. all the bits stored in DDRD
- Use bit operations when possible to suggest use of I/O bit assembly commands and to avoid unintentionally setting extra bit values

Set one pin:
```
PORTD |=(1<<3);
```
same as
```
PORTD |=(1<<PD3);
```
Clear one pin:
```
PORTD &= ~((1<<3));
```
same as
```
PORTD &=~((1<<PD3));
```

- **Don't forget to set direction of pins first!**

Outputting values to PORTx

- Do not set all 8 bits in register PORTD without regard for the directions of each individual pin, i.e. all the bits stored in DDRD
- Use bit operations when possible to suggest use of I/O bit assembly commands and to avoid unintentionally setting extra bit values

```
Set one pin:
PORTD |=(1<<3);
        same as
PORTD |=(1<<PD3);
Clear one pin:
PORTD &= ~((1<<3));
        same as
PORTD &=~((1<<PD3));
```

- **Don't forget to set direction of pins first!**
- Remember if pins are configured as inputs (DDRDn bit is 0) then the corresponding bit in PORTD (PORTDn) sets the pull-up status

Setting multiple bits

- Letś say we need 0,2,4,6 pins to be as input and 1,3,5,7 as output

```
DDRD =(1<<1)|(1<<3)|(1<<5)|(1<<7); //set all port D pins as outputs
Same as
DDRD = (1<<7)|(0<<6)|(1<<5)|(0<<4)|(1<<3)|(0<<2)|(1<<1)|(0<<0);
                                          // alternating pin directions
```

Setting multiple bits

- Letś say we need 0,2,4,6 pins to be as input and 1,3,5,7 as output

```
DDRD =(1<<1)|(1<<3)|(1<<5)|(1<<7); //set all port D pins as outputs
Same as
DDRD = (1<<7)|(0<<6)|(1<<5)|(0<<4)|(1<<3)|(0<<2)|(1<<1)|(0<<0);
                                        // alternating pin directions
```

- PD7 is defined as 7 in the device include file. USING PD7 instead of 7 is arguably more self-documenting:

```
DDRD = (1<<PD7)|(0<<PD6)|(1<<PD5)|(0<<PD4)|(1<<PD3)|
                                        (0<<PD2)|(1<<PD1)|(0<<PD0);
So we can output values to 1,3,5 and 7 pins
PORTD |=(1<<1)|(1<<3)|(1<<5)|(1<<7);
Or clear them
PORTD &= ~((1<<1)|(1<<3)|(1<<5)|(1<<7));
```

Checking multiple bits

```
flag = PIND & (0b00000001 | 0b01000000);
    if (flag){
        // do something when flag is non-zero
    }
```

- The following modification changes nothing but expresses intent more explicitly

```
    if (flag!=0){
        // do something when flag is non-zero
    }
```

Special Functions

- You may also use the **_BV(x)** macro defined in **avr/sfr _defs.h** which is included through **avr/io.h** as **# define _BV(x) (1$<<$x)**

```
# include "avr\io.h"
int main(void) {
    DDRD &=~_BV(0); //set PORTD pin0 to zero as input
    PORTD |=_BV(0);//Enable pull up;
    DDRD |=_BV(1);//set PORTD pin1 to one as output
    PORTD |=_BV(1);//led ON
    while(1) {
            if (bit_is_clear(PIND, 0)){
                    //if button is pressed
                    while(1) {
                        PORTD &=~_BV(1);//led OFF
                        //LED OFF while Button is pressed
                        loop_until_bit_is_set(PIND, 0);
                        PORTD|=_BV(1);//led ON
 }      }       }       }
```

Using predefined bits

-

```
UCSR0B = _BV(TXEN0)|_BV(RXEN0); //enable RX and and TX
```

Using predefined bits

-

```
UCSR0B = _BV(TXEN0)|_BV(RXEN0); //enable RX and and TX
```

- Both RXEN0 and TXEN0 is predefined in **iom169p.h**

Software Delay Functions

- _delay_ms(double _ms)

Software Delay Functions

- _delay_ms(double _ms)
  - Requires # include <util/delay.h >

Software Delay Functions

- _delay_ms(double _ms)
  - Requires # include <util/delay.h >
  - F_CPU preprocessor symbol should be defined as MCPU frequency in Hz using # define or passed through the -D compiler option

Software Delay Functions

- _delay_ms(double _ms)
  - Requires # include <util/delay.h >
  - F_CPU preprocessor symbol should be defined as MCPU frequency in Hz using # define or passed through the -D compiler option
    - In code: # define F_CPU 8000000UL // 8 MHz

Software Delay Functions

- _delay_ms(double _ms)
  - Requires # include <util/delay.h >
  - F_CPU preprocessor symbol should be defined as MCPU frequency in Hz using # define or passed through the -D compiler option
    - In code: # define F_CPU 8000000UL // 8 MHz
    - Command line option: -D F_CPU=8000000UL

Software Delay Functions

- _delay_ms(double _ms)
  - Requires # include <util/delay.h >
  - F_CPU preprocessor symbol should be defined as MCPU frequency in Hz using # define or passed through the -D compiler option
    - In code: # define F_CPU 8000000UL // 8 MHz
    - Command line option: -D F_CPU=8000000UL
  - Max delay is $4294967.295 \star 10^6$ / F_CPU ms (ex: 536871 ms for a 8MHz clock)

Software Delay Functions

- _delay_ms(double _ms)
  - Requires # include <util/delay.h >
  - F_CPU preprocessor symbol should be defined as MCPU frequency in Hz using # define or passed through the -D compiler option
    - In code: # define F_CPU 8000000UL // 8 MHz
    - Command line option: -D F_CPU=8000000UL
  - Max delay is 4294967.295 $\star$ 10^6 / F_CPU ms (ex: 536871 ms for a 8MHz clock)
    - assumes the avr-gcc toolchain being used has __builtin_avr_delay_cycles (unsigned long) support

Software Delay Functions

- _delay_ms(double _ms)
  - Requires # include <util/delay.h >
  - F_CPU preprocessor symbol should be defined as MCPU frequency in Hz using # define or passed through the -D compiler option
    - In code: # define F_CPU 8000000UL // 8 MHz
    - Command line option: -D F_CPU=8000000UL
  - Max delay is $4294967.295 \star 10^6$ / F_CPU ms (ex: 536871 ms for a 8MHz clock)
    - assumes the avr-gcc toolchain being used has __builtin_avr_delay_cycles (unsigned long) support
    - Otherwise max delay is less and reduced precision is used for long delays (see documentation)

Software Delay Functions

- _delay_ms(double _ms)
  - Requires # include <util/delay.h >
  - F_CPU preprocessor symbol should be defined as MCPU frequency in Hz using # define or passed through the -D compiler option
    - In code: # define F_CPU 8000000UL // 8 MHz
    - Command line option: -D F_CPU=8000000UL
  - Max delay is $4294967.295 \star 10^6$ / F_CPU ms (ex: 536871 ms for a 8MHz clock)
    - assumes the avr-gcc toolchain being used has __builtin_avr_delay_cycles (unsigned long) support
    - Otherwise max delay is less and reduced precision is used for long delays (see documentation)
    - Use multiple delay commands if needed

Software Delay Functions

- _delay_ms(double _ms)
  - Requires # include <util/delay.h >
  - F_CPU preprocessor symbol should be defined as MCPU frequency in Hz using # define or passed through the -D compiler option
    - In code: # define F_CPU 8000000UL // 8 MHz
    - Command line option: -D F_CPU=8000000UL
  - Max delay is $4294967.295 \star 10^6$ / F_CPU ms (ex: 536871 ms for a 8MHz clock)
    - assumes the avr-gcc toolchain being used has __builtin_avr_delay_cycles (unsigned long) support
    - Otherwise max delay is less and reduced precision is used for long delays (see documentation)
    - Use multiple delay commands if needed
  - Conversion of delay to clock cycles will be rounded up to the next integer to ensure at least the specified delay

Software Delay Functions

- _delay_ms(double _ms)
  - Requires # include <util/delay.h >
  - F_CPU preprocessor symbol should be defined as MCPU frequency in Hz using # define or passed through the -D compiler option
    - In code: # define F_CPU 8000000UL // 8 MHz
    - Command line option: -D F_CPU=8000000UL
  - Max delay is $4294967.295 \star 10^6$ / F_CPU ms (ex: 536871 ms for a 8MHz clock)
    - assumes the avr-gcc toolchain being used has __builtin_avr_delay_cycles (unsigned long) support
    - Otherwise max delay is less and reduced precision is used for long delays (see documentation)
    - Use multiple delay commands if needed
  - Conversion of delay to clock cycles will be rounded up to the next integer to ensure at least the specified delay
    - Alternatively, user can define __DELAY_ROUND_DOWN__and __DELAY_ROUND_CLOSEST__to round down and round to closest integer

Software Delay Functions

- _delay_us(double _us)

Software Delay Functions

- _delay_us(double _us)
  - Same as before but max delay is 1000 times less: 4294967.295 $\star$ 10^6 / F_CPU us (ex: 536871 us for a 8MHz clock)

AVR C Code

Download code from instructor website (c_example.c)