

Exploring the Discrete Input Output Functions of CCS Compiler

Input

Syntax:

```
char input(constant pin);
```

The input function returns the current state of pin. pin is the bit address of an I/O pin, and the definitions can be found in the .h file of the particular device in use.

The method of I/O used is dependent on the last #use*_IO directive.

Returns: 0 if the pin is low or 1 if the pin is high

Examples:

```
while ( !input(PIN_B1) )
{
    if( input(PIN_A0) )
        printf("A0 is now high\r\n");

    int16 i=PIN_B1;
}
```

Input port

Syntax:

```
char input_A();
char input_B();
```

The input_port function reads the entire port, and returns the 8-bit value. The direction register is changed in accordance with the last specified #use*_io directive. By default with standard I/O before the input is done, the data direction is set to input.

Returns: 8-bit value representing port

Examples:

```
data = input_b();
```

Output bit

Syntax:

```
void output_bit(const pin, char value);
```

The output_bit function outputs value (0 or 1) to the specified pin. pin is the bit address of an I/O pin, and the definitions can be found in the .h file of the particular device in use.

The method of I/O used is dependent on the last #use*_IO directive.

Returns: None

Examples:

```
output_bit( PIN_B0, 0);  
// Same as output_low(pin_B0);  
  
output_bit( PIN_B0,input( PIN_B1 ) );  
// Make pin B0 the same as B1  
  
output_bit( PIN_B0,shift_left(&data,1,input(PIN_B1)));  
  
// Output the MSB of data to  
// B0 and at the same time  
// shift B1 into the LSB of data  
  
int16 i=PIN_B0;  
output_bit(i,shift_left(&data,1,input(PIN_B1)));  
  
//same as above example, but  
//uses a variable instead of a constant
```

Output_float

Syntax:

```
void output_float(const pin);
```

The `output_float` function sets the data direction of `pin` to input. `pin` is the bit address of an I/O pin, and the definitions can be found in the `.h` file of the particular device in use.

The method of I/O used is dependent on the last `#use*_IO` directive.

Returns: None

Examples:

```
#include<16F877.h>
#fuses HS,NOWDT
#use delay(clock=10000000)
#use fast_IO(D)
#use fixed_IO(c_outputs = PIN_C2)

void main()
{
    char d;
    while(1){
        output_low(PIN_C2);
        delay_ms(500);
        output_float(PIN_D0); //Port D pin 0 is an input
        d =0;
        while(d==0)
            d = input(PIN_D0);

        output_high(PIN_C2);
        for(d=0;d<10;d++){
            output_low(PIN_D0);
            delay_ms(500);
            output_high(PIN_D0);
            delay_ms(500);
        }
    }
}
```

```
if( (data & 0x80)==0 )
    output_low(pin_A0);
else
    output_float(pin_A0);
```

Output high

Syntax:

```
void output_high(const pin);
```

The output_high function sets pin to the high state. pin is the bit address of an I/O pin, and the definitions can be found in the .h file of the particular device in use.

The method of I/O used is dependent on the last #use*_IO directive.

Returns: None

Examples:

```
output_high(PIN_A0);
Int16 i=PIN_A1;
output_low(PIN_A1);
```

Output low

Syntax:

```
void output_low(const pin);
```

The `output_low` function sets `pin` to the ground state. `pin` is the bit address of an I/O pin, and the definitions can be found in the `.h` file of the particular device in use.

The method of I/O used is dependent on the last `#use*_IO` directive.

Returns: None

Examples:

```
#include<16F877.h>
#fuses HS,NOWDT
#use fixed_io(d_outputs=PIN_D3,PIN_D2,PIN_D1,PIN_D0)

void main()
{
    char d;

    port_b_pullups(true);
    while(1){
        b = input(PIN_B0);
        if(b==0)
            output_low(PIN_D0);
        else
            output_high(PIN_D0);

        b = input(PIN_B1);
        output_bit(PIN_D1,b);

        b = input(PIN_B2);
        output_bit(PIN_D2,b);

        b = input(PIN_B3);
        output_bit(PIN_D3,b);
    }
}

output_low(PIN_A0);
Int16i=PIN_A1;
output_low(PIN_A1);
```

Output port

Syntax:

```
void output_A(char value);
```

```
void output_B(char value);
```

The output_port function outputs value to the port specified in the function call. The direction register is changed in accordance with the last specified #use*_io directive. The availability of the specific functions is dependent on the device in use.

Returns: None

Examples:

```
output_B (0xf0);
```

```
//B7,B6,B5,B4 are input pins
```

```
//B3,B2,B1,B0 are output pins.
```

Port b pullups

Syntax:

```
void port_b_pullups(char value);
```

The port_b_pull_ups function enables or disables the internal pull-up resistors on Port B according to value. If value is true (0), the pull_ups are enabled. If value is false(1), the pull-ups are disabled.

Returns: None

Examples:

```
port_a_pullups(0);
```

Set tris port

Syntax:

```
void set_tris_A(char value);
```

```
void set_tris_B(char value);
```

The `set_tris_port` function sets the I/O port direction register associated with port according to value. Each bit in value corresponds to a pin on the port. If the bit is a 1, then the pin is set up as an input. If the bit is a 0, then the pin is set up as an output.

These functions must be used with `#use fast_io`. When the default standard I/O is used, the built-in functions will set the I/O direction automatically and then overrides any settings made by the `set_tris_port` functions.

Examples:

```
#include<16F877.h>
#fuses HS,NOWDT
#use delay(clock=10000000)
#use fast_IO(D)

void main()
{
    char d;
    while(1){

        //sequence of the program is, set C2 low,
        //wait 2 seconds, read port D. Set C2 high
        //output the inverted portD value,
        //wait for 2 second and repeat

        output_low(PIN_C2);
        delay_ms(5000);
        set_tris_D(0xFF); //Port D is all inputs
        d = input_D();
        output_high(PIN_C2);
        set_tris_d(0x00);
        output_D(d^0xFF);
        delay_ms(5000);
    }
}
```

```
set_tris_b( 0x0F ); // Here we can use set_tris_b() or set_tris_B().
```

```
// B7,B6,B5,B4 are outputs
```

```
//B3,B2,B1,B0 are inputs
```