Digital Input and Output

Steve Gunn
Hardware Interfacing

• Digital Output
  – Used for switching things on/off
  – e.g. LED, CS pin on another device, ...
  – Generally robust

• Digital Input
  – Detection of presence or absence of a signal
  – e.g. Switches, Rotary Encoders, ...
  – Signal from mechanical devices is not always robust
  – Filtering/de-bouncing in hardware or software
Resources Used

- AVR Datasheet
  - Section 13: I/O-Ports
- Il Matto Quick Reference
  - Pin Functions
  - Input/Output
- AVR Libc
  - #include <avr/io.h>
## Pin Functions

<table>
<thead>
<tr>
<th>Port</th>
<th>Pin</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>PA7</td>
<td>ADC7</td>
<td>ADC input channel 7</td>
</tr>
<tr>
<td></td>
<td>PA nuts</td>
<td>ADCn</td>
<td>ADC input channel n</td>
</tr>
<tr>
<td></td>
<td>PA0</td>
<td>ADC0</td>
<td>ADC input channel 0</td>
</tr>
<tr>
<td></td>
<td>PB7</td>
<td>SCK</td>
<td>SPI Bus Master clock input</td>
</tr>
<tr>
<td></td>
<td>PB6</td>
<td>MISO</td>
<td>SPI Bus Master Input/Slave Output</td>
</tr>
<tr>
<td></td>
<td>PB5</td>
<td>MOSI</td>
<td>SPI Bus Master Output/Slave Input</td>
</tr>
<tr>
<td></td>
<td>PB4</td>
<td>SS</td>
<td>SPI Slave Select input</td>
</tr>
<tr>
<td></td>
<td>PB3</td>
<td>OC0B</td>
<td>Timer/Counter 0 Output Compare Match B Output</td>
</tr>
<tr>
<td></td>
<td>PB2</td>
<td>AIN1</td>
<td>Analog Comparator Negative Input</td>
</tr>
<tr>
<td></td>
<td>PB1</td>
<td>AIN0</td>
<td>Analog Comparator Positive Input</td>
</tr>
<tr>
<td></td>
<td>PB0</td>
<td>INT2</td>
<td>External Interrupt 2 Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T1</td>
<td>Timer/Counter 1 External Counter Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CLK0</td>
<td>Divided System Clock Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T0</td>
<td>Timer/Counter 0 External Counter Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XCK0</td>
<td>USART0 External Clock Input/Output</td>
</tr>
<tr>
<td></td>
<td>PC7</td>
<td>TOSC2</td>
<td>Timer Oscillator pin 2</td>
</tr>
<tr>
<td></td>
<td>PC6</td>
<td>TOSC1</td>
<td>Timer Oscillator pin 1</td>
</tr>
<tr>
<td></td>
<td>PC5</td>
<td>TDI</td>
<td>JTAG Test Data Input</td>
</tr>
<tr>
<td></td>
<td>PC4</td>
<td>TD0</td>
<td>JTAG Test Data Output</td>
</tr>
<tr>
<td></td>
<td>PC3</td>
<td>TMS</td>
<td>JTAG Test Mode Select</td>
</tr>
<tr>
<td></td>
<td>PC2</td>
<td>TCK</td>
<td>JTAG Test Clock</td>
</tr>
<tr>
<td></td>
<td>PC1</td>
<td>SDA</td>
<td>2-wire Serial Bus Data Input/Output Line</td>
</tr>
<tr>
<td></td>
<td>PC0</td>
<td>SCL</td>
<td>2-wire Serial Bus Clock Line</td>
</tr>
<tr>
<td></td>
<td>PD7</td>
<td>OC2A</td>
<td>Timer/Counter2 Output Compare Match A Output</td>
</tr>
<tr>
<td></td>
<td>PD6</td>
<td>OC2B</td>
<td>Timer/Counter2 Output Compare Match B Output</td>
</tr>
<tr>
<td></td>
<td>PD5</td>
<td>OC1A</td>
<td>Timer/Counter1 Output Compare Match A Output</td>
</tr>
<tr>
<td></td>
<td>PD4</td>
<td>OC1B</td>
<td>Timer/Counter1 Output Compare Match B Output</td>
</tr>
<tr>
<td></td>
<td>PD3</td>
<td>ICP1</td>
<td>Timer/Counter1 Input Capture Trigger</td>
</tr>
<tr>
<td></td>
<td>PD2</td>
<td>XCK1</td>
<td>USART1 External Clock Input/Output</td>
</tr>
<tr>
<td></td>
<td>PD1</td>
<td>INT1</td>
<td>External Interrupt 1 Input</td>
</tr>
<tr>
<td></td>
<td>PD0</td>
<td>TXD1</td>
<td>USART1 Transmit Pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RXD1</td>
<td>USART1 Receive Pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TXD0</td>
<td>USART0 Transmit Pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RXD0</td>
<td>USART0 Receive Pin</td>
</tr>
</tbody>
</table>

Notes: Each pin also has a pin change interrupt. Each pin can also be configured as GPIO.

- 32 I/O pins
- Grouped as 4 ports
- Each pin has a special function
- Or can be used as GPIO
- Sec. 13 of AVR Datasheet
Figure 13-2. General Digital I/O \(^{(1)}\)
I/O Registers

- DDRx, PORTx, and PINx are registers inside the AVR
- These are mapped into memory
- Writing and reading to/from these locations enables hardware control
- C programming language can read/write to/from memory
Working with Port Registers

• Data Memory locations consist of 1 byte (= 8bits)

• Each port register controls all 8 bits of the port

• Learn to control only the relevant I/O pins
  
  – E.g. a green LED connected to PB7 and a red LED connected to PB6.
  
  – How do you turn the green LED on/off without changing the red LED?

• Learn how to manipulate individual bits in C
Bit Manipulation
Bit Manipulation Operators in C

• Bit Shift Operators
  
  <<  Left Shift
  
  >>  Right Shift

• Logical Bit Operators
  
  &   AND
  
  |   OR
  
  ^   XOR
  
  ~   NOT

• Very important to know the difference between & and && in C
Shifting bits Left

Suppose we wish to set the fifth bit (zeroth bit is on the right).

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

uint8_t a;      /* Unsigned 8-bit integer */
a = 0b00100000; /* 32 */
a = (1 << 5);    /* 32 */
a = (1 << PA5);  /* 32 */
a = _BV(5);     /* 32 */
```

where

```
#define _BV(bit) (1 << (bit))
```
Shifting bits Right

#include <avr/io.h>

uint8_t a;      /* Unsigned 8-bit integer */
a = 0b00100000; /* 32 */
a = (a >> 3);    /*  4 */
a = (a >> 1);    /*  2 */

Shifting right is equivalent to dividing by 2.

Shifting right by n is equivalent to dividing by $2^n$.

Shifting left by n is equivalent to multiplying by $2^n$. 
Logical Bit {AND, OR} Operator

```c
#include <avr/io.h>

uint8_t a, b, c, z;

a = 0b00101010; /* 42 */
b = 0b00011111; /* 31 */
c = 0b00000000; /* 0 */

z = a & b;  /* 10 */
z = b & a;  /* 10 */
z = a & c;  /* 0 */
z = a | b;  /* 63 */
z = b | a;  /* 63 */
z = a | c;  /* 42 */
```
Logical Bit \{XOR, NOT\} Operator

```c
#include <avr/io.h>

uint8_t a, b, c, z;

a = 0b00101010; /* 42 */
b = 0b00011111; /* 31 */
c = 0b00000000; /* 0 */

z = a ^ b;    /* 53 : 0b00110101 */
z = b ^ a;    /* 53 */
z = a ^ c;    /* 42 */

z = ~a;       /* 213 : 0b11010101 */
z = ~b;       /* 224 : 0b11100000 */
z = ~c;       /* 255 : 0b11111111 */
```
AVR Bit Manipulation

**Bit Manipulation** \( (n \in \{0,1,\ldots,7\}, \ r \in \{I/O \ Registers\}) \)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uint8_t value;</code></td>
<td>Declare value as an 8-bit byte</td>
</tr>
<tr>
<td>#define _BV(n) ((1 \ll (n)))</td>
<td>Bit Value</td>
</tr>
<tr>
<td>value = 0xFF;</td>
<td>Set all 8-bits of byte value</td>
</tr>
<tr>
<td>value = 0x00;</td>
<td>Clear all 8-bits of byte value</td>
</tr>
<tr>
<td>value = ~value;</td>
<td>Invert all bits of byte value</td>
</tr>
<tr>
<td>value</td>
<td>= _BV(n);</td>
</tr>
<tr>
<td>value &amp;= ~_BV(n);</td>
<td>Clear bit (n) of byte value</td>
</tr>
<tr>
<td>if bit_is_set(r, n) { ... }</td>
<td>Test if bit (n) of (r) is set</td>
</tr>
<tr>
<td>if bit_is_clear(r, n) { ... }</td>
<td>Test if bit (n) of (r) is clear</td>
</tr>
<tr>
<td>loop_until_bit_is_set(r, n);</td>
<td>Wait until bit (n) of (r) is set</td>
</tr>
<tr>
<td>loop_until_bit_is_clear(r, n);</td>
<td>Wait until bit (n) of (r) is clear</td>
</tr>
</tbody>
</table>

```c
value = PIN_x;
```

```c
DDR_x &= ~_BV(n);
```

```c
PORT_x &= ~_BV(n);
```

```c
PORT_x |= _BV(n);
```

```c
PORT_x |=_ BV(n);
```

```c
PORT_x &= 0xFF;
```

```c
PORT_x &= 0x00;
```

```c
PIN_x &= _BV(n);
```

```c
DDR_x |= _BV(n);
```

```c
DDR_x &= _BV(n);
```

```c
value = 0xFF;
```

```c
value = 0x00;
```

```c
value = ~value;
```

```c
if (PIN_x & _BV(n)) \{ ... \}
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```

```c
```
Digital Output
What can be connected to an Output?

**Output**

Symmetrical pin drive (40mA max.)

\[ V_{OH} \geq 2.53V, \ V_{OL} \leq 0.66V \text{ (@10mA)} \]

- Current drive means many devices can be directly driven:
  - LEDs (10-20mA)
  - Chip Select Signals (< 1mA)
  - Piezo Speaker (30mA)

- Transistor or MOSFET required to switch higher currents

- Level Shifter required for different voltage (Logic) levels
Two ways to drive an LED

**Sourcing Current**

```
+3V3
  +  +R1
   +  D1
    +  PB7
      +  GND
```

**Sinking Current**

```
+3V3
  +  +R2
   +  D2
    +  PB7
```
## Output Pin Configuration

### Table 13-1. Port Pin Configurations

<table>
<thead>
<tr>
<th>DDxn</th>
<th>PORTxn</th>
<th>PUD (in MCUCR)</th>
<th>I/O</th>
<th>Pull-up</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Input</td>
<td>Yes</td>
<td>Pxn will source current if ext. pulled low.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output Low (Sink)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output High (Source)</td>
</tr>
</tbody>
</table>
I/O Pin as an Output

Output

<table>
<thead>
<tr>
<th>DDR( x ) = 0xFF;</th>
<th>Set 8-bits of port ( x ) as outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORT( x ) = 0xFF;</td>
<td>Set all output bits on port ( x ) high</td>
</tr>
<tr>
<td>PORT( x ) = 0x00;</td>
<td>Set all output bits on port ( x ) low</td>
</tr>
<tr>
<td>PIN( x ) = 0xFF;</td>
<td>Toggle all output bits on port ( x ) high</td>
</tr>
<tr>
<td>DDR( x ) | = _BV(( n ));</td>
<td>Set bit ( n ) of port ( x ) as output</td>
</tr>
<tr>
<td>PORT( x ) | = _BV(( n ));</td>
<td>Set bit ( n ) of port ( x ) high</td>
</tr>
<tr>
<td>PORT( x ) &amp; = ~_BV(( n ));</td>
<td>Set bit ( n ) of port ( x ) low</td>
</tr>
<tr>
<td>PIN( x ) &amp; = _BV(( n ));</td>
<td>Toggle bit ( n ) of port ( x )</td>
</tr>
</tbody>
</table>
Output Example

#include <avr/io.h>

int main()
{
    uint8_t i;
    /* Set all Port A bits as outputs */
    DDRA = 0b11111111;
    while(1) {
        for(i=0x80; i>0x00; i>>=1) PORTA = i;
        for(i=0x01; i>0x00; i<<=1) PORTA = i;
    }
}
LEDTest.c example from X2

```c
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    /* set LED pin as an output */
    DDRB |= _BV(PB7);
    /* forever loop */
    for (; ;)
    {
        /* Set pin B7 high */
        PORTB |= _BV(PINB7);
        _delay_ms(100);
        /* Set pin B7 low */
        PORTB &= ~_BV(PINB7);
        _delay_ms(900);
    }
}
```

Change only one bit (others unchanged)

Set one bit (Logic High)

```c
PORTB |= _BV(PB7);
```

Clear one bit (Logic Low)

```c
PORTB &= ~_BV(PB7);
```
Digital Input
What can be connected to an input?

**Input**

- Tri-state or internal pull-up (~35kΩ)
- $V_{IH} \geq 1.98\text{V}, \ V_{IL} \leq 0.99\text{V}$

- High input impedance
  - Push button switches (pull-up enabled)
  - Rotary encoder (2 or 3 bits required)
  - Digital output from another device

- Level Shifter required for different voltage (Logic) levels
Equivalent I/O Pin Schematic

Figure 13-1. I/O Pin Equivalent Schematic
## Input Pin Configuration

### Table 13-1. Port Pin Configurations

<table>
<thead>
<tr>
<th>DDxn</th>
<th>PORTxn</th>
<th>PUD (in MCUCR)</th>
<th>I/O</th>
<th>Pull-up</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Input</td>
<td>Yes</td>
<td>Px n will source current if ext. pulled low.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output Low (Sink)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output High (Source)</td>
</tr>
</tbody>
</table>
I/O Pin as an Input

**Input**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDR$x = 0x00;</td>
<td>Set 8-bits of port $x$ as inputs</td>
</tr>
<tr>
<td>PORT$x = 0xFF;</td>
<td>Enable pull-ups on input port $x$</td>
</tr>
<tr>
<td>PORT$x = 0x00;</td>
<td>Configure inputs as tri-state on port $x$</td>
</tr>
<tr>
<td>value = PIN$x;$</td>
<td>Read value of port $x$</td>
</tr>
<tr>
<td>DDR$x &amp; = ~_BV(n);</td>
<td>Set bit $n$ of port $x$ as input</td>
</tr>
<tr>
<td>PORT$x</td>
<td>= _BV(n);</td>
</tr>
<tr>
<td>PORT$x &amp; = ~_BV(n);</td>
<td>Configure tri-state on bit $n$ of port $x$</td>
</tr>
<tr>
<td>if (PIN$x &amp; _BV(n)) { ... }</td>
<td>Test value of pin $n$ on port $x$</td>
</tr>
</tbody>
</table>
Input Example

```
#include <avr/io.h>
int main() {
    uint8_t count = 0;
    /* Set all Port A bits as inputs */
    DDRA = 0b00000000;
    while(1) {
        /* Wait for switch to be pressed */
        while (((PINA & _BV(PA0)) != 0);
        count++; /* Increment count */
        /* Wait for switch to be released */
        while (((PINA & _BV(PA0)) == 0);
    }
}
```