

I/O Ports programming

Kizito NKURIKIYEZU,
Ph.D.

AVR I/O ports

- The input/output (I/O) pins of an AVR MCU are controlled by PORT peripheral registers.
- The ports are named PORTA, PORTB, PORTC, etc. All pin functions are configurable individually per pin
- Each pin on a port can be modified without modifying any other pin
- Three I/O memory address locations allocated for each port²
 - Data Register – PORTx (Read/Write)
 - Data Direction Register – DDRx (Read/Write)
 - Port Input Pins – PINx (Read)
- DDRs and PORTs have a zero initial values for all bits being 0.
- On reset, all ports are configured as input
- To read the value of the PORT, the PIN register is used

²<https://microchipdeveloper.com/avr:ioports>

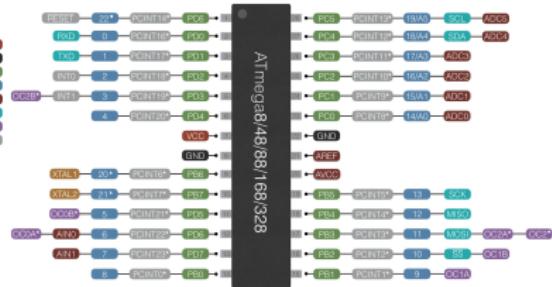


FIG 1. Pinout of the ATmega328P ¹

ATmega328 has 28 pins in total grouped into 3 ports, PORTB, PORTC and PORTD. Port C is an analogue Port and it has six pins in total

¹<https://github.com/MCUduude/MiniCore#pinout>

AVR I/O pin

- All port pins have individually selectable pull-up resistors with a supply-voltage invariant resistance.
- All I/O pins have protection diodes to both VCC and Ground
- Each pin has a maximum operating voltage of 6V
- Output buffer can source or sink an absolute maximum current of 40mA per I/O pin and the whole device can

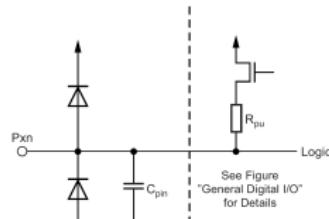


FIG 2. I/O Pin Equivalent Schematic

- DDRx—selects pin direction, which can be input or output:

- Writing 1 to DDRx make the corresponding PORTx pins as output.
- Writing 0's to DDRx make the corresponding PORTx pins as Input.

- PORTx—Used to write the data to the Port pins

- PINx —holds the value of the pin

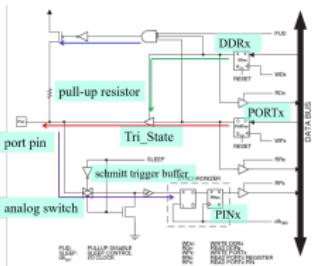


FIG 3. AVR MCU Port Block Diagram

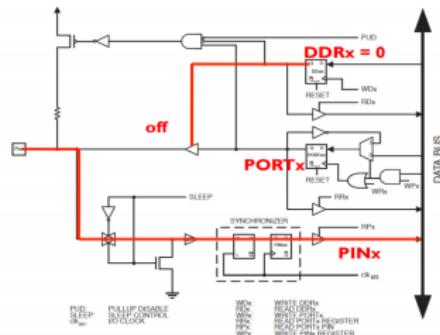


FIG 4. Configuring a pin as an input

If DDRx.n has logic '0' value, then buffer enters the high impedance tri-state (Hi-Z). In this case, PORTx.n is disconnected from the pin, but there appears an ability to read pin value directly.

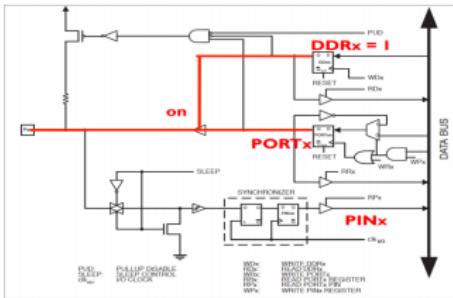


FIG 5. Configuring a pin as an output

If DDRx.n has a logic 1 value, then the buffer let bit through from PORTx register

TAB 1. 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)

- If pin is HIGH when the pin is configured as an input pin, the pull-up resistor is activated.
- To switch the pull-up resistor off, the pin has to be LOW or the pin has to be configured as an output pin.

AVR I/O ports

- DDRs and PORTs have a zero initial values for all bits being 0.
- Writing a 0 to a bit in DDRD sets the corresponding pin to input (and a 1 will set the pin to output). This implies that all pins are initially configured for input.
- When set as an input pin, a pull-up resistor can be activated by writing a 1 to the corresponding PORTD bit.

Operating Temperature -55°C to +125°C
Storage Temperature -65°C to +150°C

Voltage on any Pin except RESET with respect to Ground -0.5V to V_{CC}+0.5V
Voltage on RESET with respect to Ground -0.5V to +13.0V
Maximum Operating Voltage 6.0V
DC Current per I/O Pin 40.0 mA
DC Current V_{CC} and GND Pins 200.0 - 400.0mA

Bare metal AVR I/O programming

- How do you change the state of a specific pin in an AVR MCU?
- For instance, let us say we want to blink an LED connected to pin 5 of PORTB of the ATMEGA328.
- In arduino, this is done with the following code

```
1 #define LED_BUILTIN 13
2 void setup() {
3     // initialize digital pin LED_BUILTIN as an
4     // output.
5     pinMode(LED_BUILTIN, OUTPUT);
6 }
7 void loop() {
8     digitalWrite(LED_BUILTIN, HIGH);      // turn the
9         // LED on
10    delay(1000);                      // wait for
11        // a second
```

Other usage considerations

- Regardless of the setting of the DDRx register, the port pin can be read from PINx. Thus, an driven output value in PORTx can always be read in PINx.
- When the “pull-up disable bit” in the Special Function I/O Register (SFIOR) is set, all pull-ups are disabled regardless of the setting of DDRx and PORTx. Pullups are also disabled during reset.
- Input pins have a 1.5 clock cycle delay before a new value can be read. Thus 1 NOP instruction (short delay) necessary to read updated pin
- Use pull-ups on unused I/O pins to lower power consumption.
- Using alternative functions of some port pins does not effect other pins.
- When configuring pins as output pins with HIGH logic, make sure that the pin is not directly connected to the ground.
- When configuring pins as output pins with LOW logic, make

Bare metal AVR I/O programming

- The above code, however, hides lots of details
- In reality, the code is changing the state of some memory address.
- If you know the memory address, you can manually change it
- These details are typically found in a datasheet of each MCU
- In the case of the ATmega328, this information is found in Figure 7-2 of the [datasheet](#)

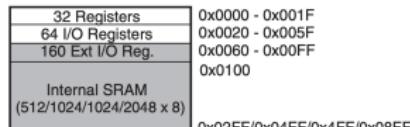


FIG 6. Data Memory Map

Bare metal AVR I/O programming

In a similar manner, page 100 of the datasheet shows the address of PORTB

PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB

ReadWrite	R/W								
Initial Value	0	0	0	0	0	0	0	0	

DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB

ReadWrite	R/W								
Initial Value	0	0	0	0	0	0	0	0	

PINB – The Port B Input Pins Address^[1]

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB

| ReadWrite | R/W | |
|---------------|-----|-----|-----|-----|-----|-----|-----|-----|--|
| Initial Value | N/A | |

FIG 7. PORTB Data Register

Bare metal AVR I/O programming

As we known the address of PORT, the previous code could be written as

```
1 int main (void)
2 {
3     while(1)
4     {
5         // Turn on the LED
6         *((volatile byte *) 0x25) |= (1 << 5);
7         // Delay 1 second (Not implemented)
8         // Turn off the LED
9         *((volatile byte *) 0x25) &= ~(1 << 5);
10    }
11 }
```

LISTING 2: Blink LED with AVR registers

Bare metal AVR I/O programming

- #include <avr/io.h> header includes the appropriate IO definitions for the device that has been specified by the -mmcu= compiler command-line switch.
- For example, for the ATMEGA328, this header will indirectly include another header "/avr/include/avr/iom328.h" which define statements are used to make shorthand notation for ports and bits.

```
1 #define PINB _SFR_IO8(0x03)
2 #define DDRB _SFR_IO8(0x04)
3 #define PORTB _SFR_IO8(0x05)
```

Bare metal AVR I/O programming

- We will use the AVR GCC Compilers for AVR³ and the AVR Libc⁴.
- A simple introduction can be found at [this website⁵](#).
- With this approach, the blink LED can be simplified

```
1 #include <avr/io.h>
2 #include <util/delay.h>
3 int main(void){
4     DDRB=(1<<PB5);
5     while(1){
6         PORTB=(1<<PB5);
7         _delay_ms(1000);
8         PORTB=(0<<PB5);
9         _delay_ms(1000);
10    }
11 }
```

LISTING 3: Blink LED with AVR registers

```
[preamble & includes]
[possibly some function definitions]
int main(void)
{
    [chip initializations]
    [event loop]
    while(1)
    {
        [do this stuff forever]
    }
    return 0;
}
```

LISTING 4: Structure of a bare-metal AVR C code

The end