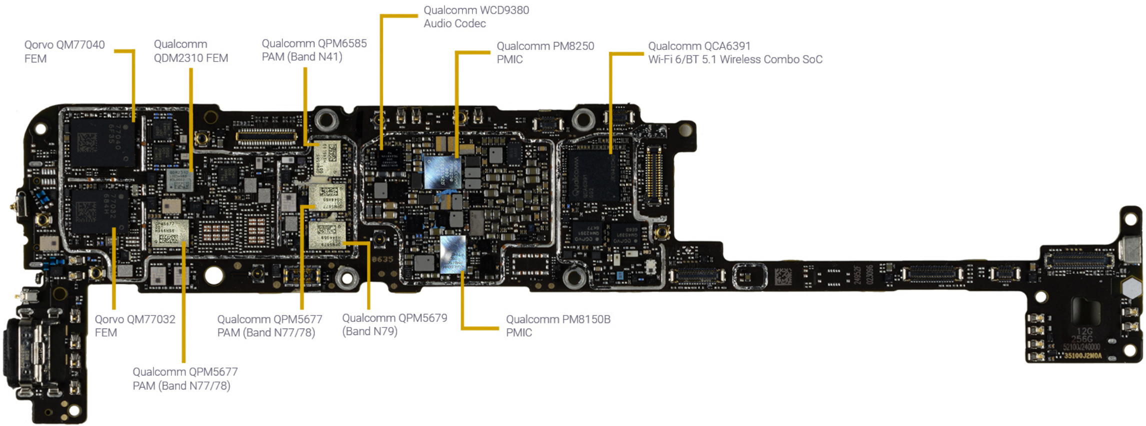


I/O Ports programming

Kizito NKURIKIYEYEZU, Ph.D.

AVR I/O ports

- All AVR Ports have true Read-Modify-Write functionality. Each pin on a port can be modified without unintentionally modifying any other pin
- Three I/O memory address locations allocated for each port
  - Data Register – PORTx (Read/Write)
  - Data Direction Register – DDRx (Read/Write)
  - Port Input Pins – PINx (Read)

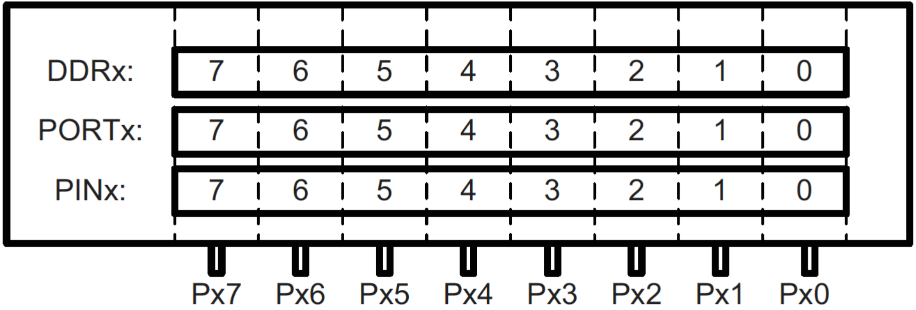
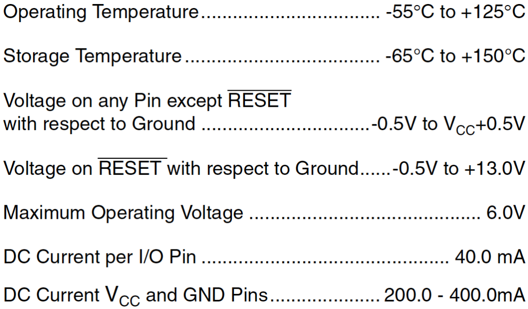


FIG 1. Relations Between the Registers and the Pins of AVR

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.
- Output buffer can source or sink an absolute maximum current of 40mA per I/O pin and the whole device can cope with a total of 200mA. (see datasheet)



Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ 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

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 sure that the pin is not directly connected to Vcc. When configuring pins as input pins, the internal pull-up structure must be kept in mind and connections should be made accordingly.

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

```
#define LED_BUILTIN 13
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off
  delay(1000);                       // wait for a second
}
```

LISTING 1: Blink LED with Arduino

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

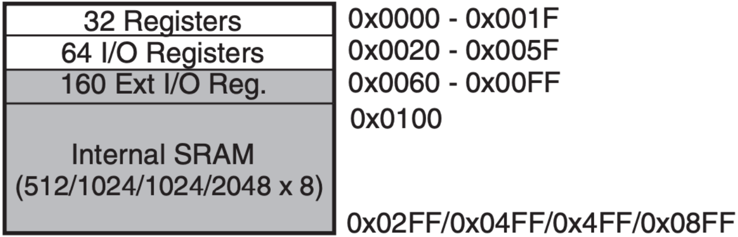
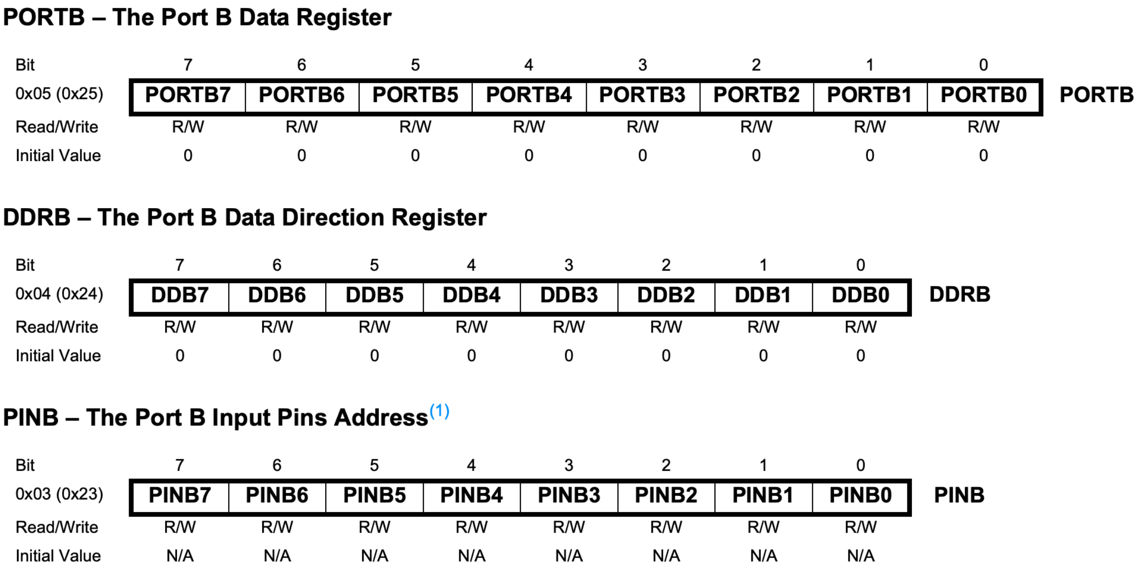


FIG 2. Data Memory Map

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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	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
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Bare metal AVR I/O programming

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

```c
int main (void)
{
  while(1)
  {
    // Turn on the LED
    *((volatile byte *) 0x25)|= (1 << 5);
    // Delay 1 second (Not implemented)
    // Turn off the LED
    *((volatile byte *) 0x25)&= ~(1 << 5);
  }
}
```

Listing 2: Blink LED with AVR registers

Bare metal AVR I/O programming

- #include <avr/io.h> header includes the apropriate 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 includeanother header "/avr/include/avr/iom328.h" which define statements are used to make shorthand notation for ports and bits.

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

Bare metal AVR I/O programming

- We will use the AVR GCC Compilers for AVR[1] and the AVR Libc[2].
- A simple introduction can be found at this website[3].
- With this approach, the blink LED can be simplified

```c
#include <avr/io.h>
#include <util/delay.h>
int main(void){
  DDRB=(1<<PB5);
  while(1){
    PORTB=(1<<PB5);
    _delay_ms(1000);
    PORTB=(0<<PB5);
    _delay_ms(1000);
  }
}
```

LISTING 3: Blink LED with AVR registers

[1] https://gcc.gnu.org/wiki/avr-gcc

[2] https://www.nongnu.org/avr-libc/

The end