AVR Freaks

DESIGN NOTE #034

AUTHOR: YVON HACHÉ, P. ENG

KEYWORDS: IO

# How to Declare I/O Pins to Ease Software Design and Maintenance

## Introduction

When developing software and hardware, often during the design phase, a signal on one I/O pin may change pin and/or port on the microcontroller. Also, if the same software is re-used on another design, the signal used is not necessarily on the same I/O pins as the previous design. When this happens, everywhere in the software where each signal is used they have to be re-mapped to a different I/O pin and/port. This can make the task difficult, long and also add risk of introducing errors.

The following shows a way to declare the I/O pins to limit the number of changes in the software to a one constant per signal when this situation happens. It makes the software more re-usable and eliminates the need of re-testing the parts that have been tested already.

Note: Macros defined in this document are written for AVRASM. This must be considered when used with other assemblers.

## Details

The examples in this design note are using a macro to make the code more readable. The macro configures the pin as an output an then set the pin state to high.

The following shows an example of how a signal is often associated with an I/O pin in assembly language.

```
.equ     Signal1   = 0                ;Pin 0
.equ     Signal2   = 1                ;Pin 1
...
sbi      DDRA, Signal1                ;Configure Signal 1 as an output
sbi      PORTA, Signal1               ;Set Signal 1 to logic 1.
...
```

In this example, if Signal1 is moved to pin 4 on Port C during the design phase of the project; Or if the same software is used on a different circuit, the following modifications have to be done:

```
.equ      Signal1   = 4                  ;Pin 4
.equ      Signal2   = 1                  ;Pin 1
...
sbi       DDRA, Signal1                  ;Configure Signal 1 as an output
sbi       PORTA, Signal1                 ;Set Signal 1 to logic 1.
...
```

These changes have to be done everywhere in the code where Signal1 is used. This could be a difficult and time-consuming task, which introduces a risk of errors.

By using a simple technique, Signal1 can be declared at one place and only that place in the code will need to be modified if Signal1 changes position. Further, to make the code more readable a macro is used instead of the bit manipulating instruction itself. The following example does the same thing as the previous example, but Signal1 is declared and used differently:

```
.equ      GEN_PORT  = 0                  ;PORT output) register address offset
.equ      GEN_DIR   = -1                 ;DDR(direction) register address offset
.equ      GEN_INP   = -2                 ;PIN(input) register address offset
.equ      Signal1   = (PORTA <<8) + 0    ;Equivalent to Signal1=0x1B00+0=0x1B00
.equ      Signal2   = (PORTA <<8) + 1    ;Equivalent to Signal2=0x1B00+1=0x1B01
...
.macro    setIoBit                       ;Usage:setIoBit GEN_PORT/DIR/INP,Signal
sbi       high(@1)+@0, low(@1)           ;AVRASM specific calls
.endmacro
...
setIoBit  GEN_DIR, Signal1               ;Configure Signal 1 as an output.
setIoBit  GEN_PORT, Signal1              ;Set Signal1 to logic 1.
...
```

The three first constants are the address offsets of each register of a particular port: Output Port Register, Direction Register and Input Register. The Signal1 constant now contains the pin number and the port address (two bytes long).

Using Signal1 as an example to show how easily changes can now be made: It is required to move the test signal (Signal1) from PIN0 on PORTA to PIN4 on PORTC. To accomplish this change only one line in the code needs to be modified:

```
.equ      GEN_PORT  = 0                 ;PORT(output) register address offset
.equ      GEN_DIR   = -1                ;DDR(direction) register address offset
.equ      GEN_INP   = -2                ;PIN(input) register address offset
.equ      Signal1   = (PORTC <<8) + 4   ;Equivalent to Signal1=0x1500+4=0x1504
.equ      Signal2   = (PORTA <<8) + 1   ;Equivalent to Signal2=0x1B00+1=0x1B01
...
.macro    setIoBit                      ;Usage:setIoBit GEN_PORT/DIR/INP,Signal
sbi       high(@1)+@0, low(@1)          ;AVRASM specific calls
.endmacro
...
setIoBit  GEN_DIR, Signal1              ;Configure Signal 1 as an output.
setIoBit  GEN_PORT, Signal1             ;Set Signal1 to logic 1.
...
```

The three first constants are always the same, no matter which port is used. The two instructions to configure Signal1 stay exactly the same. When using this technique, the constants could be declared in a configuration file separate from the software source code file. If the software is re-used on a different hardware, only the configuration file has to be modified to match the new hardware. The software is never modified. If the software is designed in a modular fashion, a library of software modules can be created and maintained for re-use where each module has its own configuration file.

Please note that this method only applies to ports/pins in the IO memory space and not to PORTs in extended IO memory space (which the ATmega128 have). This is because the SBI instruction is limited to only address up to 0x20 in the IO address space and because the three registers associated with the PORT in extended IO are not organized as the rest of the ports.