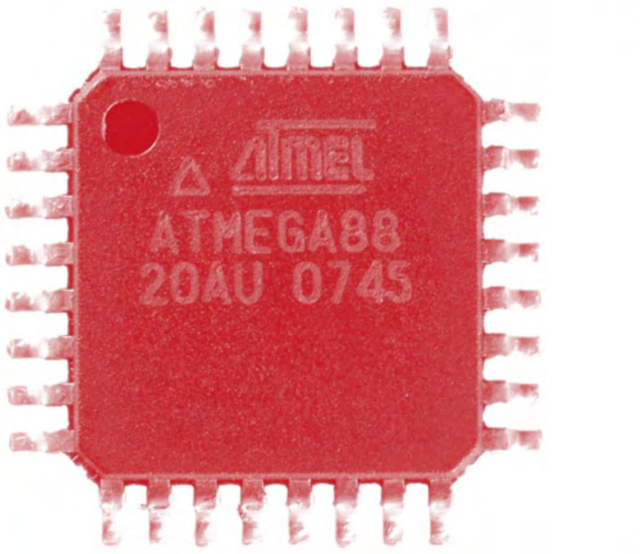
BASCOM AVR Course (2)

Using the ATmega ports

Burkhard Kainka



The port pins are the gateway between real world events and the microcontroller. The user can send out control signals and read back information. Here we give a few simple programming examples to quickly get you started inputting and outputting data.

A look at the data sheet gives some insight into the complexity of the port architecture of these microcontrollers (**Figure 1**). The ports can be configured as output or input (with or without pull-up resistors). Despite their complexity they are quite easy to use and only three important registers are needed to define the port configuration: The Data Direction Register (DDRx), the Port Output Register (PORTx) and the Port Input Register (PINx). There is also a single PUD bit (pull-up disable) which disconnects all pull-ups. The following example programs begin by using Port B.

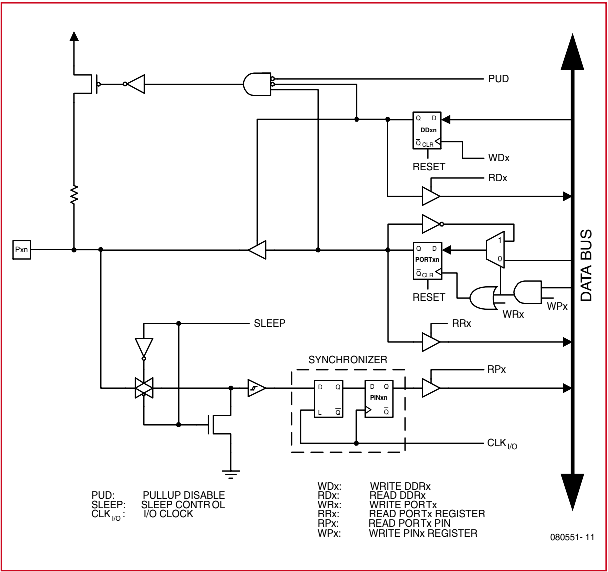


Figure 1. The ATmega port architecture.

Reading input values

After a reset the internal Data Direction Register is reset to zero which configures all the ports as inputs. The Port Register is also reset to zero. In this condition the port pins look to the outside world like typical high impedance digital CMOS inputs (**Figure 2**). With all the inputs open-circuit the value stored in PINB is random and changes if you touch the pins with your finger (first discharge any static charge you may be carrying).

Listing 1 uses Port B as an input port. The following is an example of values you will see on the screen.

```
63
0
61
0
```

The values of PINB are changing but PORTB remains at zero, which is not surprising because we have not yet changed the port output register. PORTB is displayed in this example just to underline the difference between the PINB and PORTB registers. Experience has shown that this causes a great deal of frustration for newcomers who confuse the two register names: "how come I get a reading of zero when there is 5 V on the input pin?" The answer of course is that you should not read PORTB but PINB (read it as Port In B) to get the value of the input pin.

Writing to an output port

The second example outputs data from Port B. It is necessary to write to the Data Direction Register to configure B as an output port. In BASCOM-AVR there are two ways this can be achieved; you can use the Register notation method (`Ddrb=255`) or the BASIC version (`Config Portb = Output`) either method has the same effect.

To run this example it's necessary to change the Goto instruction at the beginning of the program to read `Goto Test2` and recompile.

To turn on alternate LEDs at the output port the decimal value 85 is written into Portb. **Listing 2** includes the hexadecimal (&H55) and binary equivalent (&B01010101) of this value, they are only included to demonstrate alternate formats. All of the LEDs on portB are switched (**Figure 3**) to produce the lighting effect (the LED boogie-woogie!).

The Mega32 has all eight port lines available for use but the Mega8 or Mega88 uses port pins PB6 and PB7 for connection of a crystal. When the fuses are configured to use an external crystal these two port pins are no longer available as I/O. The same is true for other dual purpose pins i.e if the hardware UART is used PD0 and PD1are not available as I/O pins.

Using the pull-up resistors

When the inputs are connected to devices like switches or optocouplers (with open-collector outputs requiring a load resistor connected to VCC) it is ideal to use the built-in pull-up resistors instead of fitting additional external resistors (**Figure 4**). Writing a '0' to any of the DDRx bits configures the port pin as an input and writing a '1' to the corresponding PORTx bit connects a pull-up resistor to that pin (**Listing 3**).

With nothing connected to the inputs the program displays:

```
63
255
63
255
```

When the pull-ups are used the quiescent state of the input pin is a logic '1' so external signals must pull the input low. Connecting PB0 to ground produces an PINB value of 62. With an input shorted a current of around 100 μA flows to ground which indicates that the pull-up resistor has a value of 50 kΩ. This corresponds well with the 20 kΩ to 100 kΩ range quoted in the datasheet.

measuring Capacitance

The ATmega port architecture is very versatile and allows a very simple capacitance meter to be built. The capacitor under test (in the range 1 nF to 10 μF) is simply connected directly to port PB0 and ground (**Figure 5**). The program Test 4 (**Listing 4**) first discharges the capacitor by outputting an active low level. The internal pull-up resistor is then enabled which charges the capacitor. The program measures the time taken for the capacitor voltage to reach a logic '1' (2.5 V approximately). The value of capacitance is proportional to the charge time.

It is necessary to calibrate the unit because of the manufacturing tolerances in the values of both the pull-up resistance and the input voltage threshold. Calibrate using a close-tolerance capacitor and change the multiplication factor (0.0730) to obtain a result corresponding to the

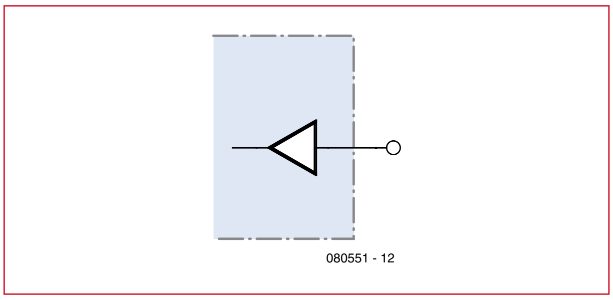


Figure 2.
A floating CMOS input.

Listing 1
Port input

```
'Bascom ATmega Ports
$regfile = "m88def.dat"
$crystal = 16000000
Baud = 9600
Goto Test1

Test1:
Dim D As Byte
Do
  D = Pinb
  Print D
  D = Portb
  Print D
  Waitms 300
Loop
```

Listing 2
Port output

```
Test2:
Config Portb = Output
'Ddrb = 255

Do
  Portb = 85
  Portb = &H55
  Portb = &B01010101
  Waitms 200
  Portb = 170
  Portb = &HAA
  Portb = &B10101010
  Waitms 200
Loop
```

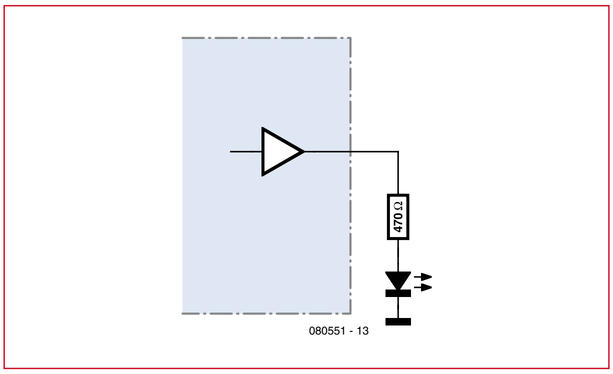


Figure 3.
Connecting an LED.

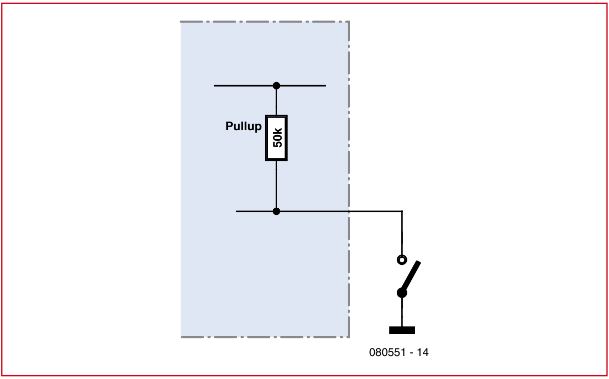


Figure 4. The internal pull-up resistors.

Listing 3

Using the pull-ups

```
Test3:
Ddrb = 0
Portb = 255
'Pullups
Do
  D = Pinb
  Print D
  D = Portb
  Print D
  Waitms 300
Loop
```

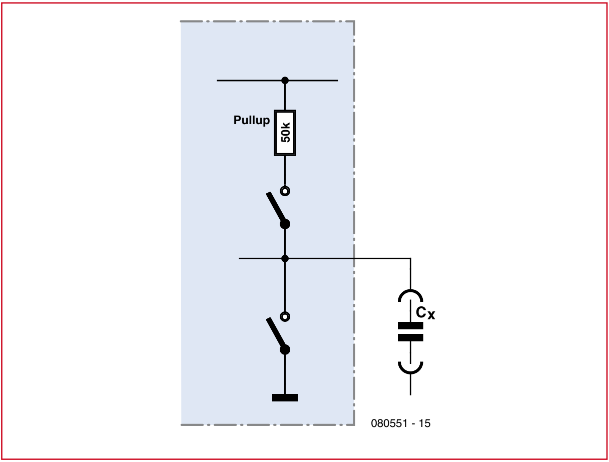


Figure 5. The principle used for capacitance measuring.

stated capacitor value. The measurements show some variation but should be accurate enough for most applications. Repeated measurements of the same capacitor gave the following spread:
1009 nF
1001 nF
1005 nF
1002 nF

Driving a stepper motor

Those of you who have a unipolar stepper motor (maybe salvaged from an old printer or 5.25-inch disk drive) may wish to experiment using this next example. Here the microcontroller uses the ULN2003 open-collector driver chip on the Elektor ATM18 test board (**Figure 6**). Only four outputs are required so we use pins PC0 to PC3. When this chip is required to drive inductive loads (e.g. motors or relays) it is necessary to connect the common cathode of the chip's protection diodes (pin 9 on IC2 or pin 8 on K6) to the load supply voltage pin 2 on K2 (VIN). The supply voltage on K2 depends on the type of motor used and can be in the range 6 V to 12 V.
Two pushbuttons are connected to PB0 and PB1 to provide direction control of the motor.
The BASCOM program is really simple, it just sequences through all four phases with four variables Phase(1) to Phase(4).
In the case where the motor just vibrates instead of rotating it is a simple job to swap phases in the program and saves changing the motor connections.
The programming examples Test5 to Test7 in Ports.bas (free download from www.elektor.com) contain several exercises to drive a stepper motor one of which shows how to build an analogue voltmeter where the motor controls the needle position.

(080551-I)

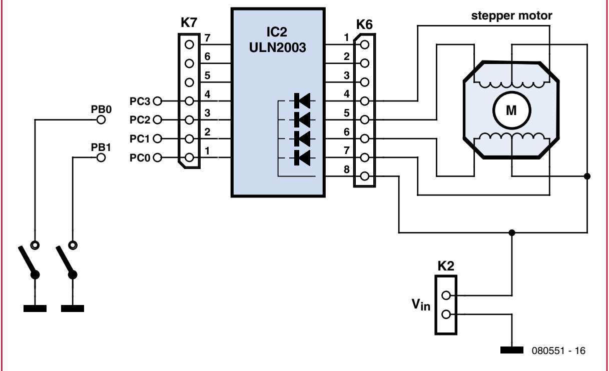


Figure 6. Connecting a unipolar stepper motor to the ATM18 test board.

Listing 4

Capacitance measurement

```
Test4:
'C-meter 1 nF .. 10µF
Dim T As Word
Dim C As Single
Dim S As String * 10
Do
  T = 0
  Ddrb.0 = 1
  Portb.0 = 0
  'low Z, 0 V
  Waitms 1000
  Ddrb.0 = 0
  Portb.0 = 1
  'Pullup
  Do
    T = T + 1
  Loop Until Pinb.0 = 1
  C = T * 0.0730
  C = Round(c)
  Print C ; " nF   "
Loop
```