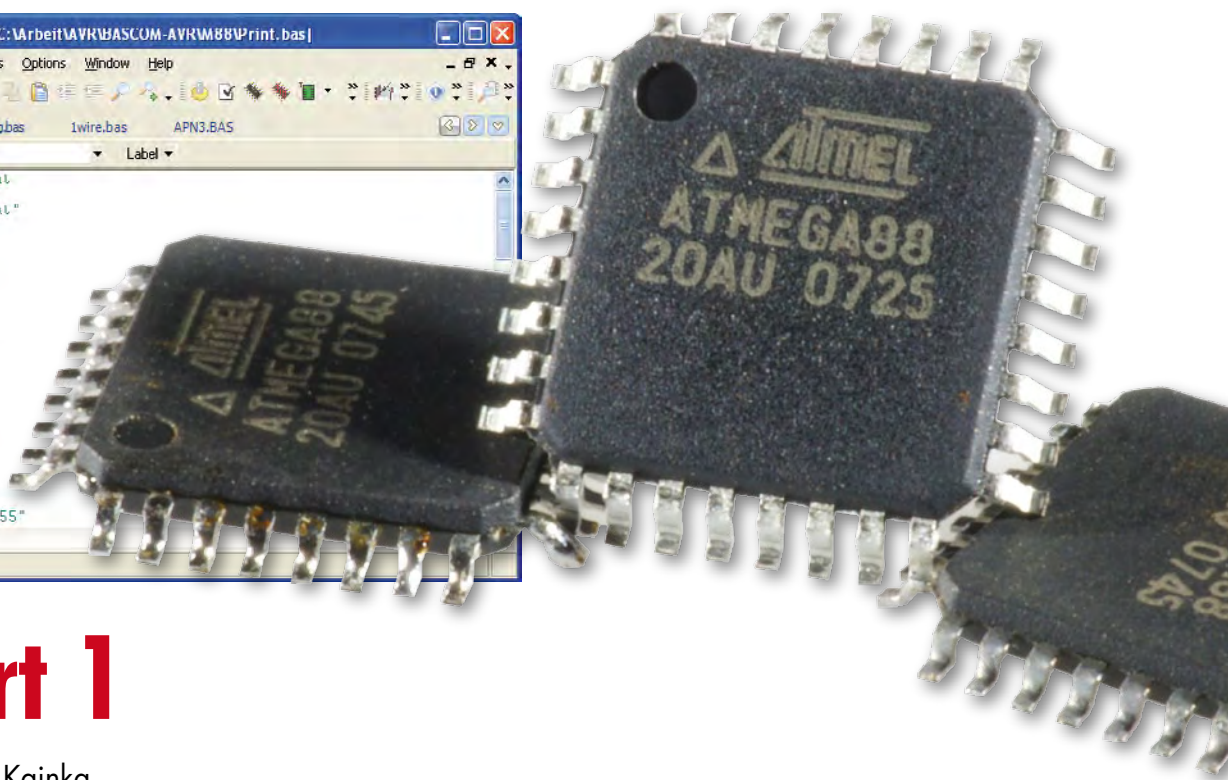
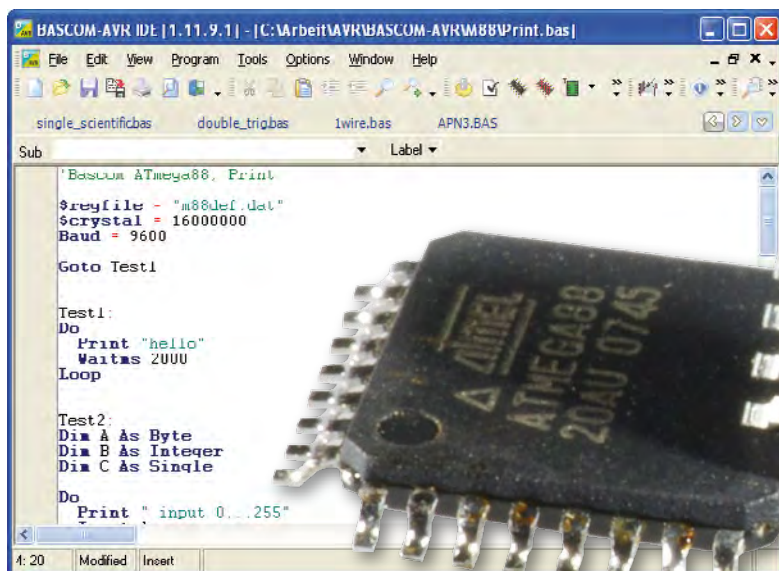


BASCOM AVR Course

Programming the ATmega controller



Part 1

Burkhard Kainka

The AVR series of microcontrollers from Atmel are very popular. Many projects already featured in Elektor have an ATmega beating away at their heart. In this mini course we turn the spotlight onto software development for these controllers. BASCOM is an ideal programming language for the newcomer; it has a steep learning curve ensuring that your path to success (and a working prototype) is reassuringly short.

The ATmega controller and BASCOM together make a strong team! Whatever application you have in mind, controllers from the ATmega range are sure to have the majority of the necessary peripheral hardware already integrated on-board: ports, timer, A/D converter, PWM output, and UART are all standard together with a choice of RAM, ROM and EEPROM size. BASCOM is one of the easier languages to use and makes interfacing to peripherals using LCDs, RC5 and I²C a simple task requiring very few instructions. There is a good range of development hardware for this microcontroller family. The STK500 from Atmel or the Elektor CC² ATM18 AVR board [1] are both suitable platforms for this course. Alternatively there is no reason why you should not experiment with your own homebrew design built on a

piece of perfboard. It also makes little difference whether you choose a Mega8, Mega88 or even the larger Mega16 or Mega32. They all basically have the same core; the main differences are in the number of port pins and the amount of internal memory space. In this first instalment of the course we look at the controller's UART and A/D converter.

The serial interface

All of the ATmega series have a built-in serial interface (UART) using pins RXD (PD0) and TXD (PD1). The signals are TTL compatible so it is necessary to add an RS232 interface buffer such as the MAX232 to implement a true

RS232 interface. These buffers are already implemented on the Atmel STK500 development system. A suitable serial to USB Adapter can be connected directly to the Elektor ATM18 AVR board for serial communication [2]. The PC will need to be running a terminal emulator program before serial communication from the ATmega can be viewed on the screen. The serial interface is covered here first because the software routine is very simple and can be easily modified if required.

Listing 1 shows all the elements necessary for all of the BASCOM programs. The first line \$regfile = "m88def.dat" indicates which controller the code will be running on; in this case it is the ATmega88. The line can be omitted and the controller type specified using the Options/Compiler/Chip menu but this method will generate an error if a different AVR system is used. It is far better to declare the controller type clearly in the program header with any program that you write. It also has priority and overwrites any setting defined in the Options menu.

It is also important to specify the crystal frequency (\$crystal = 16000000 for 16 MHz). It influences both the division ratio necessary to provide the requested communications baud rate clock (Baud = 9600) and also the timebase used to count milliseconds in the 'wait' statements (Waitms 2000 for 2 s).

One feature of the test program given here is the use of the unconditional jump instruction

Goto Test1. It is normally good programming practice to avoid using

Goto statements because they

interrupt the program structure. In this instance we have

included several programming

examples in the same source

code so it is only necessary to alter

the first Goto Test1 to Goto Test2 or

3, etc. (depending on which example

you want to run) and then recompile. This

avoids the need to compile individual source

files for each example and reduces the number

of files generated in the compilation process. Additional test

programs can simply be added to the code and run using

an appropriate Goto statement.

The small program example used for Test1 forms an endless

loop between the Do and Loop statements. The program

outputs a 'hello' message every two seconds. A terminal

emulator program like HyperTerminal is needed to receive

the message.

Calculating

The program in **Listing 2** is used to calculate the area (C) of a circle where 'A' is the given radius:

$$C = A^2 * 3.1415$$

'A' must be an integer variable, which is dimensioned as a byte and input via the serial interface. The value of 'A' (in the range 0 to 255) is first multiplied by itself. The resultant intermediate value 'B' can be up to a word long (0 to 65535). The result 'C' is a real value and is dimensioned as single which requires four bytes to store its value.

Anyone familiar with other dialects of BASIC may be wondering why the calculation has not been written as $C = 3.1415 * A * A$ or $\text{Print } 3.1415 * A * A$.

The reason is that BASCOM only allows one calculation per expression so it is necessary to break down complex

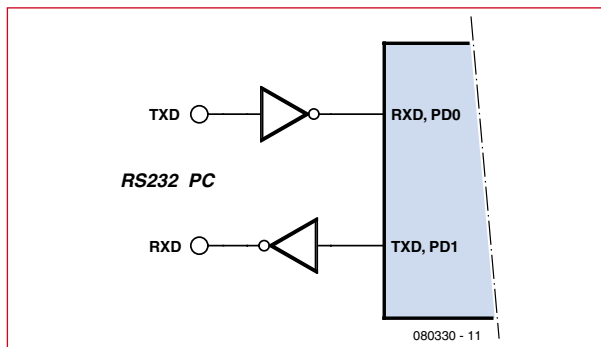


Figure 1.
The serial interface.

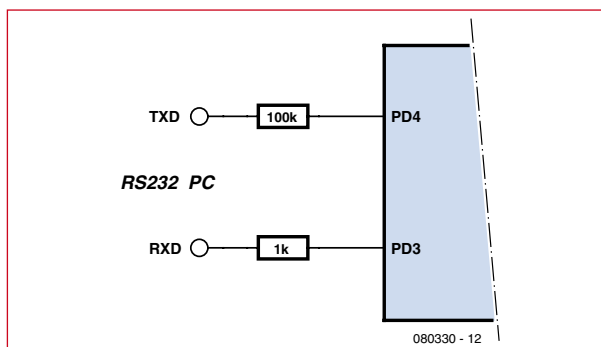


Figure 2.
The simplest serial interface without an inverter.

calculations into series of single steps to avoid violating this rule.

Procedures

The A/D converter in the ATmega has a 10-bit resolution. **Listing 3** shows an example of how the converter is initialised:

Config Adc = Single , Prescaler = 64 , Reference = Off.

The converter clock frequency is defined here as 1/64 of the processor clock, i.e. 250 kHz with a processor clock of

Listing 1

Print 'hello'

```
,Bascom ATmega88, Print
$regfile = „m88def.dat“
$crystal = 16000000
Baud = 9600
```

```
Goto Test1
```

```
Test1:
Do
  Print „hello“
  Waitms 2000
Loop
```

```
Test2:
...
Test3:
...
End
```

Listing 2

Calculating in BASCOM

```
Dim A As Byte
Dim B As Word
Dim C As Single

Do
  Print " input 0...255"
  Input A
  B = A * A
  C = B * 3.1415
  'not allowed: C = 3.1415 * A * A
  Print C
Loop
```

16 MHz. The internal reference voltage is not used but is derived externally and applied to pin AREF. In most cases the 5 V stabilised supply voltage can be used.

A procedure is used to input the ADC measurement and convert it to a voltage reading. It is only worthwhile using a procedure if it can be reused by different parts of the main program body. In this example two voltage measurements are displayed. The new procedure is called Sub Voltage. Before the procedure can be called it must first be declared: Declare Sub Voltage.

The program fragment shown here does not conform to good software writing practice. In Visual Basic it is usual to pass the channel parameter 'N' when the procedure is called: Voltage(N). Alternatively a function could be written and then called using: U = Voltage(N). In the example here we are using only global variables so that D, N and U are accessible from all parts of the program including other procedures. All of the accepted software guidelines indicate that this is not good programming practice but in this instance it simplifies the job for the compiler and controller. Experience has shown that even large programs occupying 100% of the flash memory of a Mega32 and using a large number of global variables are completely stable in operation and run without problem. Passing variables to procedures can sometimes generate errors which are quite difficult to trace.

Listing 3

Using a procedure

```
Declare Sub Voltage
Dim N As Byte
Config Adc = Single , Prescaler = 64 , Reference = Off
Start Adc

Do
  N = 0 : Voltage
  Print "ADC(0) = " ; U ; " V"
  N = 1 : Voltage
  Print "ADC(1) = " ; U ; " V"
  Print
  Waitms 500
Loop

Sub Voltage
  D = Getadc(n)
  D = D * 5
  U = D / 1023
End Sub
```

The software serial interface

One of the many highlights of the BASCOM compiler is its software serial interface. Say you are already using the hardware serial interface with the RXD and TXD pins and you require another COM interface? Alternatively it could be that you do not have any RS232 interface inverter chip (e.g. a MAX232) on your board but still need to provide a connection for an RS232 cable. In both cases BASCOM has a solution to the problem. Any available port pins can be defined in software as serial interface pins for the software serial interface.

The example **Listing 4** uses PD3 as a serial output. Communication speed is set to 9600 Baud and the interface number 2. To output a simple message you can use Print #2, 'hello' for example.

Using the INVERTED parameter allows the interface to be built without the need for a MAX232 interface inverter. BASCOM inverts the signal so that inactive level is a logic Low. The PC interface RXD signal can then be connected directly. The signal levels are not truly RS232 but it provides a useable interface solution and can, for example, be used to interface a USB to serial adapter to the Elektor ATM18 AVR board without the need for a MAX232.

(080330-1)

Listing 4

Using the Software UART

```
Baud = 9600
,Open "comd.3:9600,8,n,1" For Output As #2
Open "comd.3:9600,8,n,1,INVERTED" For Output As #2
Config Adc = Single , Prescaler = 64 , Reference = Off

Do
  N = 0 : Voltage
  Print #2 , "ADC(0) = " ; U ; " V"
  N = 1 : Voltage
  Print #2 , "ADC(1) = " ; U ; " V"
  Print #2 ,
  Waitms 500
Loop
```

References

[1] ATM18 AVR Board, *Elektor* April 2008.

[2] USB <-> RS-232 Cable, *Elektor* July/August 2008.

Downloads and further information

The programming examples and more information for this course can be downloaded from the project page at www.elektor.com. We also welcome your feedback in the Elektor Forum.

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.

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.

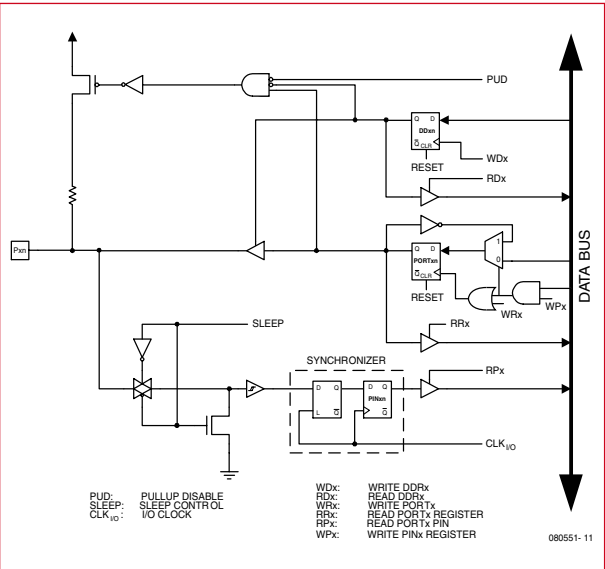


Figure 1.
The ATmega port
architecture.

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 PD1 are 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 PBO to ground produces an PINB value of 62. With an input shorted a current of around 100 μ A flows to ground which indicates that the pull-up resistor has a value of 50 k Ω . This corresponds well with the 20 k Ω to 100 k Ω 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 μ F) is simply connected directly to port PBO 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

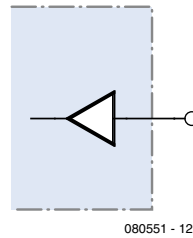


Figure 2.
A floating CMOS input.

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
```

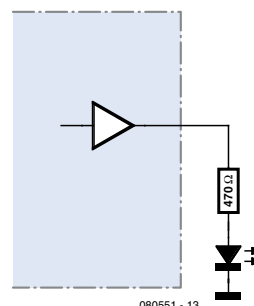


Figure 3.
Connecting an LED.

Figure 4.
The internal pull-up
resistors.

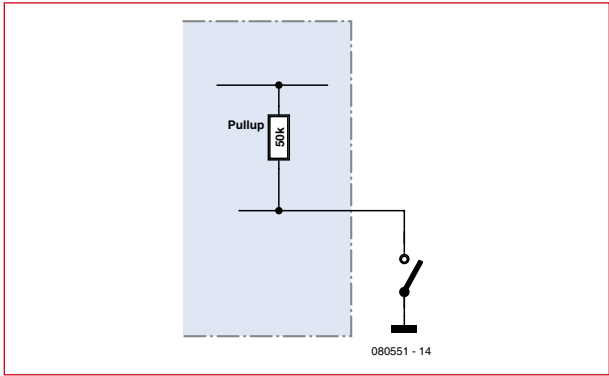
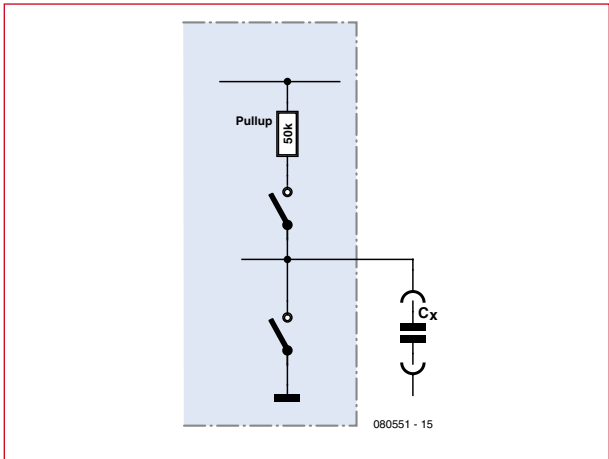


Figure 5.
The principle used for
capacitance measuring.

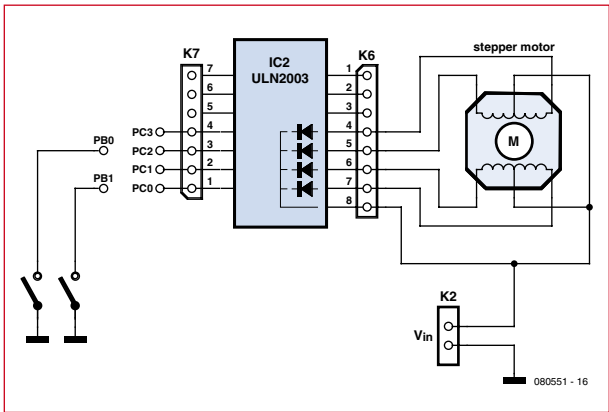


stated capacitor value. The measurements show some variation but should be accurate enough for most applications. Repeated measurements of the same capacitor gave the following spread:
1009 nF
1001 nF
1005 nF
1002 nF

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

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



Listing 3

Using the pull-ups

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

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.

(080551-1)

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
```

BASCOM AVR

Course (3)

Timers and Interrupts

Burkhard Kainka (Germany)

Many practical tasks can only be solved by using accurate timing. The ATmega controllers are well equipped in this respect; the Mega8 to Mega32 controllers all have three timers, Timer 0 and 2 are 8-bit while Timer 1 is a full 16 bit wide.

The ATmega controller's timer/counter section looks a little daunting at first sight (**Figure 1**). They are highly configurable and require a certain amount of care to ensure they are set up correctly for your application. For those programming in Assembler this configuration procedure is quite involved but as you will see BASCOM simplifies things a lot.

The first thing to decide is the source of the timer/counter clock signal. It can come from the internal clock (directly or via a prescaler) or from an external source (e.g. connect to pin P1 for Timer 1). The counters can count on either the rising or falling clock edge and the counter value can be read or changed at any time via the TCNT1 register. When an overflow occurs it can generate an interrupt. The counters are commonly used for generating Pulse Width Modulated (PWM) signals. This is just a brief outline of some of the more basic properties of the timer/counters, as you become more familiar with the controller you will begin to get a better appreciation of their versatility.

Reading the timer

For the first exercise we are using the 16-bit timer driven by the system clock crystal and divided by 256 in the prescaler. In BASCOM all this information can be written on one line: Config Timer1 = Timer , Prescale = 256. The timer also begins counting so it is not necessary to use Start Timer1.

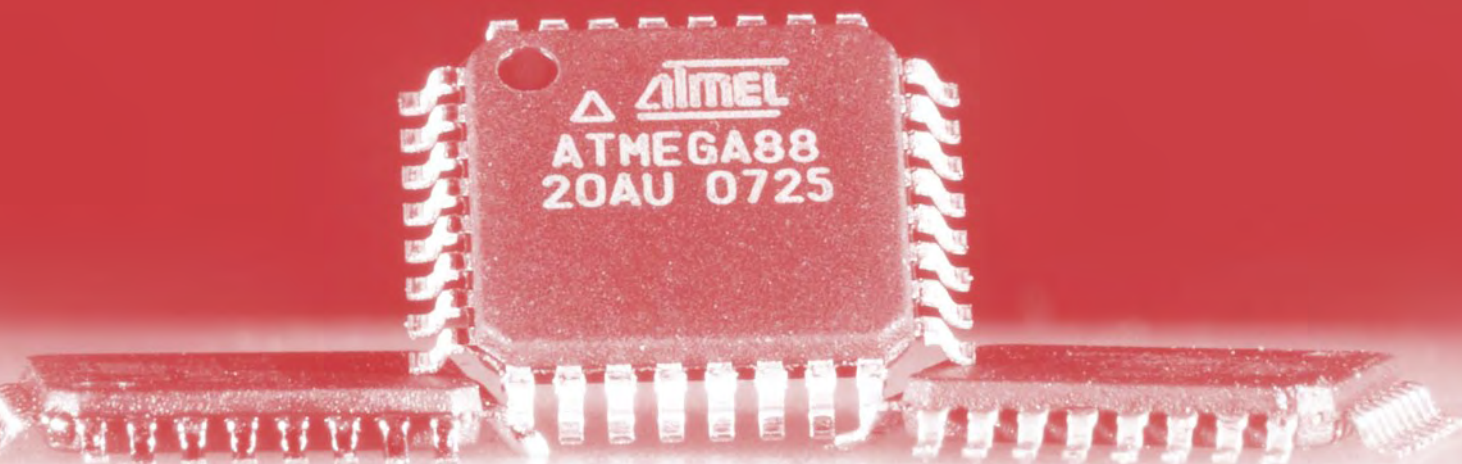
Listing 1 is the first test, as before we are using a Goto to reduce 'compilation clutter'. The listing as printed will only ever go to the first example, you will need to change fifth line to Goto Test2 and recompile for the next exercise.

In Test1 timer/counter1 just runs continuously and the counter value is displayed five times per second. The values are

in the range from 0 to 65535, and we can see that after roughly one second an overflow occurs:

```
088
17864
30706
43547
56389
69230
82071
94912
107753
120594
133435
146276
159117
171958
184799
197640
210481
223322
236163
249004
261845
274686
287527
299368
312209
325050
337891
350732
363573
376414
389255
402096
414937
427778
440619
453460
466301
479142
491983
504824
517665
530506
543347
556188
569029
581870
594711
607552
620393
633234
646075
658916
671757
684598
697439
710280
723121
735962
748803
761644
774485
787326
800167
813008
825849
838690
851531
864372
877213
890054
902895
915736
928577
941418
954259
967100
979941
992782
1005623
1018464
1031305
1044146
1056987
1069828
1082669
1095510
1108351
1121192
1134033
1146874
1159715
1172556
1185397
1198238
1211079
1223920
1236761
1249602
1262443
1275284
1288125
1300966
1313807
1326648
1339489
1352330
1365171
1378012
1390853
1403694
1416535
1429376
1442217
1455058
1467899
1480740
1493581
1506422
1519263
1532104
1544945
1557786
1570627
1583468
1596309
1609150
1621991
1634832
1647673
1660514
1673355
1686196
1699037
1711878
1724719
1737560
1750401
1763242
1776083
1788924
1801765
1814606
1827447
1840288
1853129
1865970
1878811
1891652
1904493
1917334
1930175
1943016
1955857
1968698
1981539
1994380
2007221
2020062
2032903
2045744
2058585
2071426
2084267
2097108
2109949
2122790
2135631
2148472
2161313
2174154
2186995
2199836
2212677
2225518
2238359
2251200
2264041
2276882
2289723
2302564
2315405
2328246
2341087
2353928
2366769
2379610
2392451
2405292
2418133
2430974
2443815
2456656
2469497
2482338
2495179
2508020
2520861
2533702
2546543
2559384
2572225
2585066
2597907
2610748
2623589
2636430
2649271
2662112
2674953
2687794
2700635
2713476
2726317
2739158
2751999
2764840
2777681
2790522
2803363
2816204
2829045
2841886
2854727
2867568
2880409
2893250
2906091
2918932
2931773
2944614
2957455
2970296
2983137
2995978
3008819
3021660
3034501
3047342
3060183
3073024
3085865
3098706
3111547
3124388
3137229
3150070
3162911
3175752
3188593
3201434
3214275
3227116
3239957
3252798
3265639
3278480
3291321
3304162
3317003
3329844
3342685
3355526
3368367
3381208
3394049
3406890
3419731
3432572
3445413
3458254
3471095
3483936
3496777
3509618
3522459
3535300
3548141
3560982
3573823
3586664
3599505
3612346
3625187
3638028
3650869
3663710
3676551
3689392
3702233
3715074
3727915
3740756
3753597
3766438
3779279
3792120
3804961
3817802
3830643
3843484
3856325
3869166
3882007
3894848
3907689
3920530
3933371
3946212
3959053
3971894
3984735
3997576
4010417
4023258
4036099
4048940
4061781
4074622
4087463
4100304
4113145
4125986
4138827
4151668
4164509
4177350
4190191
4203032
4215873
4228714
4241555
4254396
4267237
4280078
4292919
4305760
4318601
4331442
4344283
4357124
4369965
4382806
4395647
4408488
4421329
4434170
4447011
4459852
4472693
4485534
4498375
4511216
4524057
4536898
4549739
4562580
4575421
4588262
4601103
4613944
4626785
4639626
4652467
4665308
4678149
4690990
4703831
4716672
4729513
4742354
4755195
4768036
4780877
4793718
4806559
4819400
4832241
4845082
4857923
4870764
4883605
4896446
4909287
4922128
4934969
4947810
4960651
4973492
4986333
4999174
5012015
5024856
5037697
5050538
5063379
5076220
5089061
5101902
5114743
5127584
5140425
5153266
5166107
5178948
5191789
5204630
5217471
5230312
5243153
5255994
5268835
5281676
5294517
5307358
5320199
5333040
5345881
5358722
5371563
5384404
5397245
5410086
5422927
5435768
5448609
5461450
5474291
5487132
5500073
5512914
5525755
5538596
5551437
5564278
5577119
5589960
5602801
5615642
5628483
5641324
5654165
5667006
5679847
5692688
5705529
5718370
5731211
5744052
5756893
5769734
5782575
5795416
5808257
5821098
5833939
5846780
5859621
5872462
5885303
5898144
5910985
5923826
5936667
5949508
5962349
5975190
5988031
6000872
6013713
6026554
6039395
6052236
6065077
6077918
6090759
6103600
6116441
6129282
6142123
6154964
6167805
6180646
6193487
6206328
6219169
6232010
6244851
6257692
6270533
6283374
6296215
6309056
6321897
6334738
6347579
6360420
6373261
6386102
6398943
6411784
6424625
6437466
6450307
6463148
6475989
6488830
6501671
6514512
6527353
6540194
6553035
6565876
6578717
6591558
6604399
6617240
6630081
6642922
6655763
6668604
6681445
6694286
6707127
6719968
6732809
6745650
6758491
6771332
6784173
6797014
6809855
6822696
6835537
6848378
6861219
6874060
6886901
6899742
6912583
6925424
6938265
6951106
6963947
6976788
6989629
7002470
7015311
7028152
7040993
7053834
7066675
7079516
7092357
7105198
7118039
7130880
7143721
7156562
7169403
7182244
7195085
7207926
7220767
7233608
7246449
7259290
7272131
7284972
7297813
7310654
7323495
7336336
7349177
7362018
7374859
7387700
7400541
7413382
7426223
7439064
7451905
7464746
7477587
7490428
7503269
7516110
7528951
7541792
7554633
7567474
7580315
7593156
7605997
7618838
7631679
7644520
7657361
7670202
7683043
7695884
7708725
7721566
7734407
7747248
7760089
7772930
7785771
7798612
7811453
7824294
7837135
7849976
7862817
7875658
7888499
7901340
7914181
7927022
7939863
7952704
7965545
7978386
7991227
8004068
8016909
8029750
8042591
8055432
8068273
8081114
8093955
8106796
8119637
8132478
8145319
8158160
8171001
8183842
8196683
8209524
8222365
8235206
8248047
8260888
8273729
8286570
8299411
8312252
8325093
8337934
8350775
8363616
8376457
8389298
8402139
8414980
8427821
8440662
8453503
8466344
8479185
8492026
8504867
8517708
8530549
8543390
8556231
8569072
8581913
8594754
8607595
8620436
8633277
8646118
8658959
8671800
8684641
8697482
8710323
8723164
8736005
8748846
8761687
8774528
8787369
8800210
8813051
8825892
8838733
8851574
8864415
8877256
8890097
8902938
8915779
8928620
8941461
8954302
8967143
8979984
8992825
9005666
9018507
9031348
9044189
9057030
9069871
9082712
9095553
9108394
9121235
9134076
9146917
9159758
9172599
9185440
9198281
9211122
9223963
9236804
9249645
9262486
9275327
9288168
9301009
9313850
9326691
9339532
9352373
9365214
9378055
9390896
9403737
9416578
9429419
9442260
9455101
9467942
9480783
9493624
9506465
9519306
9532147
9544988
9557829
9570670
9583511
9596352
9609193
9622034
9634875
9647716
9660557
9673398
9686239
9699080
9711921
9724762
9737603
9750444
9763285
9776126
9788967
9801808
9814649
9827490
9840331
9853172
9866013
9878854
9891695
9904536
9917377
9930218
9943059
9955900
9968741
9981582
9994423
10007264
10020105
10032946
10045787
10058628
10071469
10084310
10097151
10110092
10122933
10135774
10148615
10161456
10174297
10187138
10200079
10212920
10225761
10238602
10251443
10264284
10277125
10290066
10302907
10315748
10328589
10341430
10354271
10367112
10379953
10392794
10405635
10418476
10431317
10444158
10457099
10469940
10482781
10495622
10508463
10521304
10534145
10546986
10559827
10572668
10585509
10598350
10611191
10624032
10636873
10649714
10662555
10675396
10688237
10701078
10713919
10726760
10739601
10752442
10765283
10778124
10790965
10803806
10816647
10829488
10842329
10855170
10868011
10880852
10893693
10906534
10919375
10932216
10945057
10957898
10970739
10983580
10996421
11009262
11022103
11034944
11047785
11060626
11073467
11086308
11099149
11111990
11124831
11137672
11150513
11163354
11176195
11189036
11201877
11214718
11227559
11240400
11253241
11266082
11278923
11291764
11304605
11317446
11330287
11343128
11355969
11368810
11381651
11394492
11407333
11420174
11433015
11445856
11458697
11471538
11484379
11497220
11510061
11522902
11535743
11548584
11561425
11574266
11587107
11599948
11612789
11625630
11638471
11651312
11664153
11676994
11689835
11702676
11715517
11728358
11741199
11754040
11766881
11779722
11792563
11805404
11818245
11831086
11843927
11856768
11869609
11882450
11895291
11908132
11920973
11933814
11946655
11959496
11972337
11985178
11998019
12010860
12023701
12036542
12049383
12062224
12075065
12087906
12100747
12113588
12126429
12139270
12152111
12164952
12177793
12190634
12203475
12216316
12229157
12242098
12254939
12267780
12280621
12293462
12306303
12319144
12331985
12344826
12357667
12370508
12383349
12396190
12409031
12421872
12434713
12447554
12460395
12473236
12486077
12498918
12511759
12524600
12537441
12550282
12563123
12575964
12588805
12601646
12614487
12627328
12640169
12653010
12665851
12678692
12691533
12704374
12717215
12730056
12742897
12755738
12768579
12781420
12794261
12807102
12819943
12832784
12845625
12858466
12871307
12884148
12896989
12909830
12922671
12935512
12948353
12961194
12974035
12986876
13000000
13012841
13025682
13038523
13051364
13064205
13077046
13089887
13102728
13115569
13128410
13141251
13154092
13166933
13179774
13192615
13205456
13218297
13231138
13243979
13256820
13269661
13282502
13295343
13308184
13321025
13333866
13346707
13359548
13372389
13385230
13398071
13410912
13423753
13436594
13449435
13462276
13475117
13487958
13500799
13513640
13526481
13539322
13552163
13565004
13577845
13590686
13603527
13616368
13629209
13642050
13654891
13667732
13680573
13693414
13706255
13719096
13731937
13744778
13757619
13770460
13783301
13796142
13808983
13821824
13834665
13847506
13860347
13873188
13886029
13898870
13911711
13924552
13937393
13950234
13963075
13975916
13988757
14001598
14014439
14027280
14040121
14052962
14065803
14078644
14091485
14104326
14117167
14130008
14142849
14155690
14168531
14181372
14194213
14207054
14219895
14232736
14245577
14258418
14271259
14284100
14296941
14309782
14322623
14335464
14348305
14361146
14373987
14386828
14399669
14412510
14425351
14438192
14451033
14463874
14476715
14489556
14502397
14515238
14528079
14540920
14553761
14566602
14579443
14592284
14605125
14617966
14630807
14643648
14656489
14669330
14682171
14695012
14707853
14720694
14733535
14746376
14759217
14772058
14784899
14797740
14810581
14823422
14836263
14849104
14861945
14874786
14887627
14900468
14913309
14926150
14938991
14951832
14964673
14977514
14990355
15003196
15016037
15028878
15041719
15054560
15067401
15080242
15093083
15105924
15118765
15131606
15144447
15157288
15170129
15182970
15195811
15208652
15221493
15234334
15247175
15260016
15272857
15285698
15298539
15311380
15324221
15337062
15349903
15362744
15375585
15388426
15401267
15414108
15426949
15439790
15452631
15465472
15478313
15491154
15503995
15516836
15529677
15542518
15555359
15568200
15581041
15593882
15606723
15619564
15632405
15645246
15658087
15670928
15683769
15696610
15709451
15722292
15735133
15747974
15760815
15773656
15786497
15799338
15812179
15825020
15837861
15850702
15863543
15876384
15889225
15902066
15914907
15927748
15940589
15953430
15966271
15979112
15991953
16004794
16017635
16030476
16043317
16056158
16068999
16081840
16094681
16107522
16120363
16133204
16146045
16158886
16171727
16184568
16197409
16210250
16223091
16235932
16248773
16261614
16274455
16287296
16300137
16312978
16325819
16338660
16351501
16364342
16377183
16390024
16402865
16415706
16428547
16441388
16454229
16467070
16479911
16492752
16505593
16518434
16531275
16544116
16556957
16569798
16582639
16595480
16608321
16621162
16634003
16646844
16659685
16672526
16685367
16698208
16711049
16723890
16736731
16749572
16762413
16775254
16788095
16800936
16813777
16826618
16839459
16852300
16865141
16877982
16890823
16903664
16916505
16929346
16942187
16955028
16967869
16980710
16993551
17006392
17019233
17032074
17044915
17057756
17070597
17083438
17096279
17109120
17121961
17134802
17147643
17160484
17173325
17186166
17199007
17211848
17224689
17237530
17250371
17263212
17276053
17288894
17301735
17314576
17327417
17340258
17353099
17365940
17378781
17391622
17404463
17417304
17430145
17442986
17455827
17468668
17481509
17494350
17507191
17520032

```



An interrupt causes a forced interruption of the main program and directs the controller to execute a sub routine (Interrupt Service Routine or ISR) to service the interrupt. Different events can be programmed to generate an interrupt and an ISR is required to respond to each type of interrupt. Here Tim0_isr would be the subroutine name but in this example we have just used Tim0_isr: as a label which indicates where the program jumps to on interrupt. The last instruction of the interrupt routine must be a RETURN. In this example further interrupts will not be serviced until the return is executed.

Test 2 configures timer 0 with a prescaler of 64, which gives it a clock frequency of 250 kHz. The counter is 8-bits wide so without further programming it will generate an overflow interrupt every 256 clock cycles. We need the counter to interrupt every 250 clocks for an accurate 1 ms timebase so it is necessary to load the counter with the value 6 each time it overflows. A word variable called Ticks is incremented every time the counter overflows. When this variable reaches 1000 it indicates that one second has elapsed and the variable called Seconds is incremented. The value of either variable can be read by the main program. In this example the program sends the value of seconds to the terminal every second starting from zero at program start. It is necessary to allow the interrupts to occur by enabling the global interrupt (Enable Interrupts) and also allow the timer 0 overflow condition to generate an interrupt (Enable Timer0). The display shows the value of seconds:

```
0
1
2
3
```

All interrupt sources can be disabled by using Disable Interrupts.

Averaged measurements

Measurements made of analogue signal levels are often affected by a 50 Hz mains signal superimposed on the voltage level. The unwanted 50 Hz component can effectively be cancelled out by sampling the analogue voltage level several times during a complete cycle of the mains voltage (20 ms) and then averaging all the measurements.

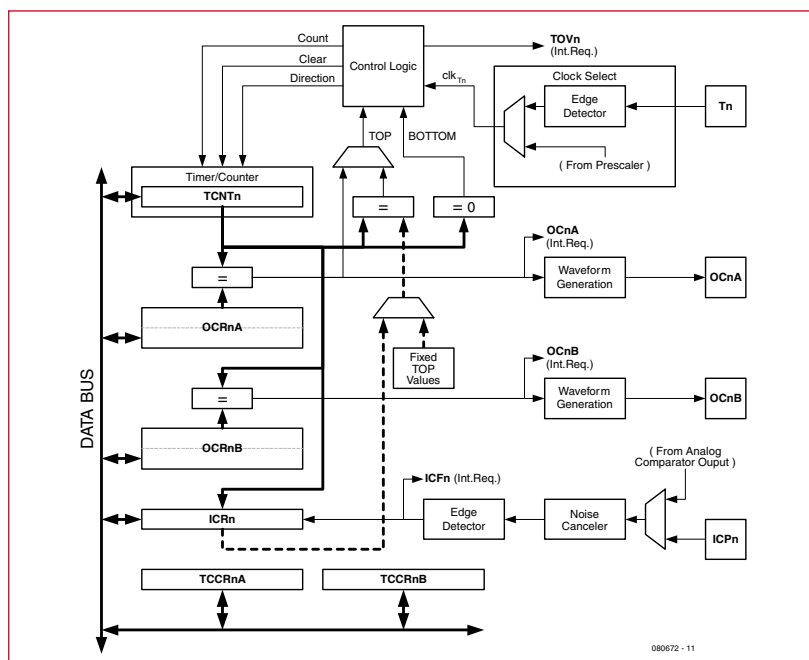


Figure 1. Block diagram of the timers.

Listing 1

Reading the timer registers

```
`Bascom ATmega88, Timer
$regfile = "m88def.dat"
$crystal = 16000000
Baud = 9600
Goto Test1

Test1:
Config Timer1 = Timer , Prescale = 256
`Start Timer1
Do
    Print Timer1
    Waitms 200
Loop
```

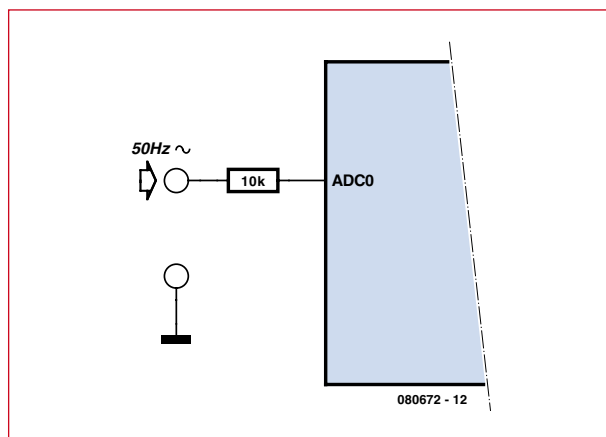



Figure 2.
Measuring an ac voltage.

For this exercise we will use a timer interrupt again to generate an accurate timebase. The average value is achieved by sampling the analogue signal 25 times in a 20 ms time window. The sampling interval is therefore 800 μ s. Timer 2 will be used with a prescale value of 64. Each time it overflows Timer2 is loaded with the value 56 so that the next overflow occurs 200 clocks later.

800 μ s is more than enough time to make the analogue measurement and calculate the sum and mean value. The variable Ticks is incremented each time a measurement is taken every interrupt. After 25 measurements the sum stored in AD0 is transferred to the variable AD0_mean. The main program averages the value and then sends it to the screen.

Averaging in this way gives such good suppression of the 50 Hz components that by using half wave rectification the system can be used to measure ac signals. The low voltage AC signal is connected to the ADC0 input via a 10 k

Listing 3

Measuring averages

```
Test3:
Dim Ad0 As Word
Dim Ad0_mean As Word
Config Adc = Single , Prescaler = 64 , Reference = Off
Config Timer2 = Timer , Prescale = 64
On Ovfl2 Tim2_isr
Enable Timer2
Enable Interrupts

Do
  Ad0_mean = Ad0_mean / 25
  Print Ad0_mean
  Waitms 100
Loop

Tim2_isr:
  '800  $\mu$ s
  Timer2 = 56
  Ticks = Ticks + 1
  Ad0 = Ad0 + Getadc(0)
  If Ticks > 24 Then
    Ticks = 0
    Ad0_mean = Ad0
    Ad0 = 0
  End If
  Return
```

protection resistor (**Figure 2**). The program now finds the average value of the positive half wave which is equal to half of the absolute average value of the sine wave. A typical sequence of measurements would be:

```
226
227
226
226
226
```

Although there is some variation the measured average value is mostly 226. This can be converted into a real voltage level: $5\text{ V} * 226 / 1023 = 1.10\text{ V}$. The measured alternating voltage therefore has an absolute average value of 2.20 V. For a sine wave this equates to an RMS value of 2.44 V and a peak to peak value of 3.46 V_{pp}. The relationship between the peak and RMS value of a sine wave is $\sqrt{2} = 1.414$. For arithmetic averaging the relationship of the peak value to the average value is $\pi/2 = 1.571$, so the absolute average value is 90.03 % of the RMS.

(080672-1)

Listing 2

Exact seconds using interrupts

```
Test2:
Dim Ticks As Word
Dim Seconds As Word
Dim Seconds_old As Word
Config Timer0 = Timer , Prescale = 64
On Ovfl0 Tim0_isr
Enable Timer0
Enable Interrupts

Do
  If Seconds <> Seconds_old Then
    Print Seconds
    Seconds_old = Seconds
  End If
Loop

Tim0_isr:
  '1000  $\mu$ s
  Timer0 = 6
  Ticks = Ticks + 1
  If Ticks = 1000 Then
    Ticks = 0
    Seconds = Seconds + 1
  End If
  Return
```

Downloads and further information:

The programming examples and more information for this course can be downloaded from the project page at www.elektor.com. As always we look forward to your feedback in the Elektor forum.

BASCOM AVR Course

(4) Counter and PWM

Burkhard Kainka (Germany)

We have already taken a look at timers in part 3 of the course. The ATmega timer/counters have far more to offer than just measuring time. Here we look at impulse counting, frequency measurement and PWM signal generation.

In the first exercise we set up timer 1 to count impulses over a period of one second. A look at the program **Listing 1** indicates that timer 1 is configured as a counter, counting on the falling edge of the input pulse and with a prescale value of 1.

The counter input is labelled T1 which for example on the Mega8 and Mega88 is pin PD5 (**Figure 1**). In this exercise we connect a low-voltage 50 Hz signal to this input via a 10 kΩ series resistor. A signal generator is not necessary here; the input impedance is relatively high so just touching the input resistor with your finger will inject a signal of sufficient level from the ambient mains field for measurement. In Europe the signal is 50 Hz, in the USA 60 Hz. The routine counts the number of pulses in 1 second so the screen shows:

```
0
50
100
150
201
251
```

After the fourth value a slight inaccuracy in the measured value creeps in. This is because the `Waitms 1000` instruction is not an exact time interval and also we have not taken into account the time necessary to output values to the display. To improve the accuracy of frequency measurements we go on in the next exercise to use timer interrupts.

Frequency measurement

The timer can reliably count external impulses with a repetition rate of up to 4 MHz. To make accurate frequency measurements we need a precise time window, in this example we use interrupts from two timers. Each time timer 1 overflows, the interrupt is serviced by `Tim1_isr` which increments the variable `Highword`. Without this variable the counter would only be able to measure frequencies up to 65535 Hz (**Listing 2**).

Timer 0 generates an exact time window of one second. When the variable `Ticks = 1`, timer 1 is reset and the measurement begins. Exactly 1000 ms later the counter value of timer 1 is copied to `Lowword` and then added to the number of overflows stored in `Highword` (multiplied by 65536) before storing the result in `Freq`. The main pro-

Listing 1 Impulse counter

```
Test1:
Config Timer1 = Counter , Edge = Falling ,
        Prescale = 1
Start Timer1
Do
    Print Timer1
    Waitms 1000
Loop
```

gram outputs `Freq` (in Hz) to the display every second. Initialising timer 1 in timer mode with a clock of 16 MHz (`Config Timer1 = Timer , Prescale = 1`) would display a frequency of 16000000 Hz. As a counter however, timer 1 can run at just a little more than a quarter of this frequency and its prescaler is synchronised to the processor clock. When you try to measure a frequency as high as 6 MHz for example the counter gating runs too slowly to register every edge of the input pulses so it misses some and shows a false reading of around 3 MHz. The design can be used to accurately measure frequencies up to and just beyond 4 MHz.

PWM outputs

Pulse Width Modulation (PWM) is a technique used in

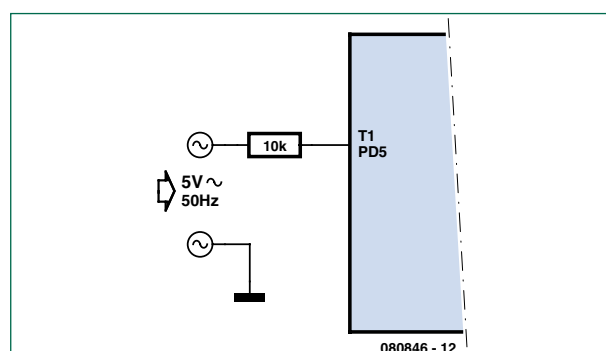


Figure 1.
A 50 Hz signal on T1.

Listing 2

Frequency measurements up to 4 MHz

```
Test2:
Dim Lowword As Word
Dim Highword As Word
Dim Ticks As Word
Dim Freq As Long
Config Adc = Single , Prescaler = 32 ,
    Reference = Off
Config Timer0 = Timer , Prescale = 64
Config Timer1 = Counter , Edge = Falling ,
    Prescale = 1
'Config Timer1 = Timer , Prescale = 1
On Ovfo Tim0_isr
On Ovfl Tim1_isr
Enable Timer0
Enable Timer1
Enable Interrupts

Do
    Print Freq
    Waitms 1000
Loop

Tim0_isr:
    '1000 µs
    Timer0 = 6
    Ticks = Ticks + 1
    If Ticks = 1 Then
        Timer1 = 0
        Highword = 0
    End If
    If Ticks = 1001 Then
        Lowword = Timer1
        Freq = Highword * 65536
        Freq = Freq + Lowword
        Ticks = 0
    End If
Return

Tim1_isr:
    Highword = Highword + 1
Return
```

many applications to provide a quasi-analogue control of power to a load without the need for a true D/A converter. Timers in the ATmega controllers can be used to generate

Listing 3 10-bit PWM

```
Test3:
Dim Pwm As Word
Config Timer1 = Pwm , Prescale = 8 , Pwm
    = 10 , Compare A Pwm = Clear Down ,
    Compare B Pwm = Clear Down

Do
    For Pwm = 0 To 1023
        Pwm1a = Pwm
        Pwm1b = 1023 - Pwm
        Waitms 5
    Next Pwm
Loop
```

Listing 4

Six PWM outputs produce an LED light 'wave'.

```
Test4:
Dim A As Single
Dim B As Single
Dim I As Byte
Dim K As Byte

Declare Sub Wave
Config Timer0 = Pwm , Prescale = 8 ,
    Compare A Pwm = Clear Down , Compare B
    Pwm = Clear Down
Config Timer1 = Pwm , Prescale = 8 , Pwm =
    8 , Compare A Pwm = Clear Down , Compare
    B Pwm = Clear Down
Config Timer2 = Pwm , Prescale = 8 ,
    Compare A Pwm = Clear Down , Compare B
    Pwm = Clear Down

Do
    For I = 1 To 60
        K = I
        Wave
        Pwm1a = Pwm
        K = I + 10
        Wave
        Pwm1b = Pwm
        K = I + 20
        Wave
        Pwm0a = Pwm
        K = I + 30
        Wave
        Pwm0b = Pwm
        K = I + 40
        Wave
        Pwm2a = Pwm
        K = I + 50
        Wave
        Pwm2b = Pwm
        Waitms 50
    Next Pwm
Loop

Sub Wave
    A = 6.1415 * K
    A = A / 60
    B = Sin(a)
    B = B + 1
    B = B * B
    B = B * 63
    Pwm = Int(b)
    If Pwm < 2 Then Pwm = 2
End Sub
```

PWM signals. Timer 1 has two independent PWM output channels with a resolution of 8, 9 or 10 bits.

Listing 3 shows how both channels Pwm1a and Pwm1b of timer 1 can be programmed to produce output signals with 10-bit resolution. The signals are output from OC1A (PB1) and OC1B (PB2). Their electrical characteristics are the same as other port pins so you can just hang an LED together with a series current-limiting resistor on the output or connect the output to a buffer like the ULN2003 fitted to the Elektor ATM18 AVR board. The program produces increasing brightness signal from channel A and decreasing brightness signal from channel B.

LED control using six PWM channels

The Mega88 provides six PWM outputs signals. Timers 0 and 2 both offer a resolution of eight bits. The individual outputs are on the following output pins:

OC1A on PB1
OC1B on PB2
OC0A on PD6
OC0B on PD5
OC2A on PB3
OC2B on PD3

In this last exercise we use all six PWM outputs, for the sake of symmetry in this application, timer 1 is configured with a resolution of only eight bits. The aim of this example (**Listing 4**) is to smoothly control the brightness of a row of LEDs such that a sinusoidal 'wave' of light travels along the row.

A loop with 60 brightness levels per LED is sufficient to produce a smooth transition between levels. The value of variable *l* is used in the sub wave to produce the light level value. It is first multiplied by 2π , divided by 60 and then its sine function is found. The result in the range ± 1 is then offset to the range 0 to 2. The eye's perception of brightness is nonlinear so to compensate, the value is squared. It now lies in the range 0 to 4 so multiplying by 63 converts to the 0 to 255 range (almost) of PWM values used to control the LEDs.

The steps at lower values of brightness are quite noticeable so the lowest possible level is limited to 2.

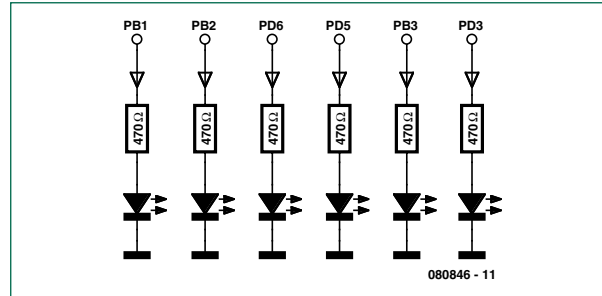


Figure 2.
Brightness control of six
outputs.

Using this calculation and the corresponding phase shift generated in the program produces an interesting lighting effect. The overall result is a wave of brightness moving along the line of LEDs. The LEDs can be arranged in a line or as a circle. It is possible to expand the line further by adding six, twelve or more LEDs.

(080846-1)

Program downloads and forum

As usual the programming examples and additional info can be downloaded from the project page at www.elektor.com/080846. We also welcome your feedback on the Elektor forum.

BASCOM AVR Course

(5)

Memory, switch polling and time management

Burkhard Kainka (Germany)

In the microcontroller embedded scene, complaints about systems having too much memory or too much processing power are rare if not non-existent — we never seem to have enough! Microcontrollers in particular have limited resources with no possibility of expansion, so it's important not to squander them by using inefficient programming practices.

Software engineers aim to produce efficient code. A simple routine like reading the value of a switch could be programmed in such a way that it uses up 100 % of the microcontrollers processing time. In this case there would be no capacity spare for the controller to perform any other tasks. It is important when designing any software that the processor resources are used efficiently. We expand on this theme here and give some pointers to how the microcontroller can be better employed.

RAM and EEPROM

In addition to the 8 kBytes of Flash memory the ATmega88 is fitted with 1024 bytes of RAM and 512 bytes of EEPROM. BASCOM uses the RAM to store variables and various stacks so how much memory is left over? To test memory allocation we will write some data into an array. The array dimension is given `A(500)`. This is handled as 500 individual bytes `A(1)` to `A(500)`. Note that there is no `A(0)`. The short test program given in **Listing 1** contains a loop which writes an incrementing data byte to memory. A second loop reads the memory and sends it to the PC.

A report file `Memory.rpt` is generated which gives an overview of how the memory has been used in the program. The file is in text file format and can be read using Notepad. The file shows memory size, exact location of all the variables and much more; very useful to see how much elbow room you have in reserve as you progress to writing larger programs.

Test 2 shows how data can be written to and read from EEPROM. In contrast to RAM the EEPROM will not lose its data when power is switched off. Data is written using the format `Writeeeprom, Variable, Memory address` and read using `Readeeprom, Variable, Memory address`. A wiped EEPROM memory location has the value `FF (255)`. From this it is possible to determine if any data has been programmed into the EEPROM. Test 2 (**Listing 2**) first writes 512 Bytes to the EEPROM, reads then displays them on the PC.

Reading the status of switches

Firmware running in stand-alone equipment will undoubtedly need to read the status of switches or pushbuttons so that the user can control the equipment. Reading the status of switches would seem at first sight to be quite a trivial process but there are a number of pitfalls. One problem is that we do not know when and for how long the button will be pressed so it is necessary to continuously read (or

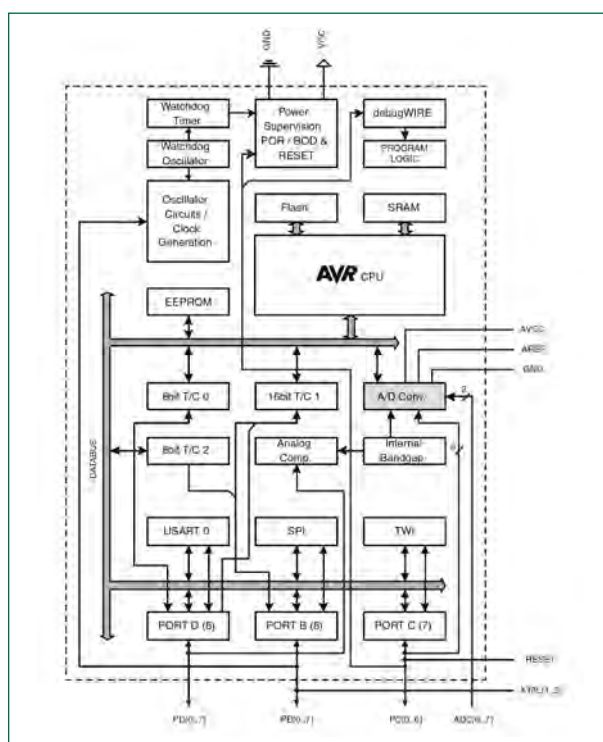


Figure 1.
ATmega88 Block diagram.

Listing 1 Data storage in RAM

```
Test1:
Dim A(500) As Byte
Dim N As Word
Do
    For N = 1 To 500
        A(n) = Low(n)
    Next N
    For N = 1 To 500
        Print A(n)
        Waitms 100
    Next N
Loop
```

Listing 2 The EEPROM

```
Test2:
    For N = 0 To 511
        Writeeprom N , N
    Next N
Dim D As Byte
Do
    For N = 0 To 511
        Readeprom D , N
        Print N , D
        Waitms 100
    Next N
Loop
```

Listing 3 LED control

```
Test3:
S1 Alias Pind.6
S2 Alias Pind.5
S3 Alias Pind.7
Out1 Alias Portd.2
Out2 Alias Portd.3
Config Portd = &B00001100
Portd.6 = 1
Portd.5 = 1
Portd.7 = 1

Out1 = 1
Do
    If S1 = 0 Then
        Out1 = 1
        Out2 = 0
        Print "1 on"
    End If
    If S2 = 0 Then
        Out1 = 0
        Out2 = 1
        Print "1 off"
    End If
    Waitms 50
Loop
```

Listing 4

PWM control

```
Test4:
Dim Pwmold As Integer
Pwma = 0
Do
    If S1 = 0 Then Pwma = Pwma + 1
    If Pwma > 1023 Then Pwma = 1023
    If S2 = 0 Then Pwma = Pwma - 1
    If Pwma < 0 Then Pwma = 0
    If S3 = 0 Then Pwma = 0
    Waitms 20
    Pwmla = Pwma
    If Pwma <> Pwmold Then
        Print Pwma
    End If
    Pwmold = Pwma
Loop
```

'poll') the switch status to ensure we do not miss a press. A systematic approach to software design is also important; it can create many problems if you need to add a switch poll routine to existing software, much better to design it in from the start where each function can be built up logically.

Another, more practical problem is that most switches suffer from contact bounce. When the contacts come together they do not switch cleanly but instead bounce, producing an output that looks like the button has been pressed several times off and on very quickly. It would therefore not be a good idea to use the switch input directly to clock a timer or counter. The bounce time is quite short, one common debouncing method is to filter out the bounce by reading the switch status say once every 10 ms.

In the next series of examples we use three pushbuttons connected on D5, D6 and D7. The corresponding port bits are set high and the data direction register sets these pins to inputs so that internal pull-up resistors are connected. An open circuit input will be read as a '1' and a '0' when the button is pressed. The port pins are given aliases so that you can use statements like: If S1 = 0 then (**Listing 3**).

Test 3 actually uses just two buttons to toggle two outputs. S1 switches the first output high and the second low while S2 toggles them back. Each key press sends a message to the PC screen. The polling is repeated after a 50 ms wait. When either button is pressed continuously, a message is sent to the serial interface every 50 ms but the port outputs do not change state.

Test 4 (**Listing 4**) uses two buttons to control the mark/space ratio of a PWM signal OC1A = PB1. One button increases the PWM value while the other decreases it. An oscilloscope shows the variation in mark/space ratio and an LED connected at the output will change in brightness. Switch debouncing is not necessary here because the routine only measures the time that the buttons are pressed.

Test 5 (**Listing 5**) uses two buttons to toggle the state of two LEDs. Each press of S1 causes the LED on Out1 to change state; likewise S2 controls the LED on Out2. Once a key

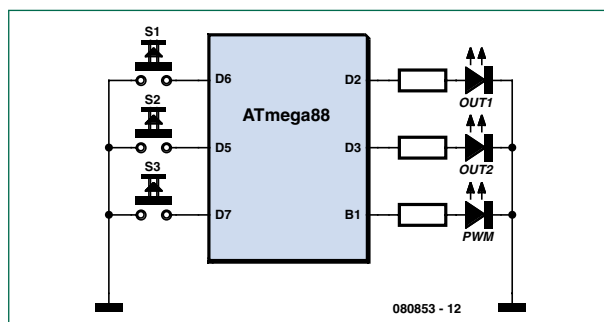


Figure 2.
Input and output

Listing 5 Two toggle flipflops

```
Test5:
Do
    If S1 = 0 Then
        If Out1 = 0 Then
            Out1 = 1
        Else
            Out1 = 0
        End If
        Waitms 10
    End If
Do
Loop Until S1 = 1
If S2 = 0 Then
    If Out2 = 0 Then
        Out2 = 1
    Else
        Out2 = 0
    End If
    Waitms 10
End If
Do
Loop Until S2 = 1
Waitms 100
Loop
```

Listing 6

Two counters

```
Test6:
Dim Count1 As Word
Dim Count2 As Word
Do
    If S1 = 0 Then
        Count1 = Count1 + 1
        Print "Count1 ";
        Print Count1
        Waitms 50
    Do
        Loop Until S1 = 1
    End If
    If S2 = 0 Then
        Count2 = Count2 + 1
        Print "Count2 ";
        Print Count2
        Waitms 50
    Do
        Loop Until S2 = 1
    End If
Loop
```

Listing 7 Switch polling using interrupt

```
Test7:
Dim Ticks As Byte
Dim Sw1 As Byte
Dim Sw2 As Byte
Dim Sw3 As Byte
Dim Sw4 As Byte
Dim Pwm1 As Integer
Dim Pwmlold As Integer
Dim Ledtimer As Byte
Dim Ledblink As Byte

Ledblink = 1
Enable Timer0
Enable Interrupts
Cls
Lcd 0

Do
    If Ticks = 1 Then Out1 = 1
    If Ticks = 5 Then Out1 = 0
Loop

Timer0isr:
    Ticks = Ticks + 1
    If Ticks = 1 Then
        If S1 = 0 Then Sw1 = Sw1 + 1 Else Sw1 = 0
        If Sw1 > 100 Then Sw1 = 100
        If S2 = 0 Then Sw2 = Sw2 + 1 Else Sw2 = 0
        If Sw2 > 100 Then Sw2 = 100
        If S3 = 0 Then Sw3 = Sw3 + 1 Else Sw3 = 0
        If Sw3 > 100 Then Sw3 = 100
    End If
    If Ticks = 2 Then
        If Sw1 = 3 Then
            Pwm1 = Pwm1 + 1
```

```
        If Pwm1 > 1023 Then Pwm1 = 1023
        End If
        If Sw1 = 100 Then
            Pwm1 = Pwm1 + 1
            If Pwm1 > 1023 Then Pwm1 = 1023
        End If
        If Sw2 = 3 Then
            Pwm1 = Pwm1 - 1
            If Pwm1 < 0 Then Pwm1 = 0
        End If
        If Sw2 = 100 Then
            Pwm1 = Pwm1 - 1
            If Pwm1 < 0 Then Pwm1 = 0
        End If
        If Pwm1 <> Pwmlold Then
            Print Pwm1
        End If
        Pwm1a = Pwm1
        Pwmlold = Pwm1
    End If
    If Ticks = 3 Then
        If Sw3 = 3 Then
            If Ledblink = 1 Then
                Ledblink = 0
            Else
                Ledblink = 1
            End If
        End If
    End If
    If Ticks = 4 Then
        Ledtimer = Ledtimer + 1
        If Ledtimer > 100 Then Ledtimer = 0
        If Ledtimer = 1 Then
            If Ledblink = 1 Then Out2 = 1
        End If
        If Ledtimer = 50 Then Out2 = 0
    End If
    If Ticks = 10 Then Ticks = 0
Return
```

press is detected the program switches the LED and loops until the switch is released. A 10 ms wait is used to filter any bounce otherwise the LED would change state on every edge of the switch bounce waveform, leaving the LED randomly on or off.

The same routine can be used to increment the values of two counters (Test 6). Each time a counter value changes, its value is sent to the PC.

Switch polling using timer interrupt

All of the preceding methods of switch polling do not use the processor resources efficiently, it spends its time either waiting or reading the switch inputs. In reality there will be more switches to read and other tasks for the firmware to take care of. The next stage is to take a more structured approach to software design so that resources are better managed. Test 7 (**Listing 7**) shows one method of how this can be achieved. Switch polling occurs in the background in a timer interrupt routine. The main program is now free to take care of other tasks.

For each button S1, S2 and S3 there is an associated variable Sw1, Sw2 and Sw3. While a button is not pressed its variable has the value zero. As long as a button is pressed the variable is incremented up to 100 where it stops. The variable indicates how long the key has been pressed, so you may for example wish to initiate some process only when its value reaches three. A long key press gives a value of 100.

The timer routine uses a counter to produce short time intervals `Ticks` which is incremented each time the timer inter-

rupts (it is reset when it reaches 10). The three switches are read only once every ten `Ticks` (when `Ticks = 1`). The interval takes care of switch debouncing and occurs often enough not to miss any press.

At other tick values different duties are performed. When `Ticks = 2` switch counters are read and a PWM signal is generated. When `Ticks = 3` the switch counter is read and `Ledblink` is toggled to switch a flashing LED. The LED output is produced when `Ticks = 4`. The sequential distribution of tasks gives the impression that all the activities are performed simultaneously. The processor still has ample processing power in reserve for many additional tasks. The main program switches output `Out1` high for five ticks and low for five ticks. An LED connected to this output appears slightly dim; the on/off repetition rate is so fast that you cannot see any flickering. The LED brightness is constant, indicating that the program is maintaining a 50:50 output clock.

The mark/space ratio of the PWM output is controlled by buttons Sw1 and Sw2. The software determines if there is a short button press or a long one. A short press changes the value by one, a longer press changes the counter value continuously. This allows the user to quickly reach the desired value.

(080853-1)

Downloads and further information

The programming examples and more information for this course can be downloaded from the project page at www.elektor.com. We also look forward to your feedback on the Elektor forum.

BASCOM AVR Course

(6)

A DDS generator using the ATmega32

Burkhard Kainka (Germany)

The first five instalments of this course have already covered many programming techniques with an emphasis on practical applications. This final instalment builds on this theme giving a detailed insight into all the software routines needed to make a simple DDS generator. We also have an offer for you, see the final page of this article!

To get a good overview of some of the possibilities of the BASCOM language and the ATmega controllers it is worth looking through the BASCOM AVR help pages, particularly all the Config options (**Figure 1**). Some of the topics we have already covered in this course and in other ATM-18 projects include:

- COM-interface, hardware and software
- Ports, input, input pull-ups, output
- A/D converter
- Timer and counter
- Timer interrupts
- PWM outputs
- RC5 input
- I²C master
- Servo impulse
- One-wire bus (elsewhere in this edition)

There is of course much more internal and external hardware that can be used. The ATmega family share the same basic core but more specialised applications call for a careful study of the datasheets to find the version best suited to the task.

This month we build an AF generator using the principles of DDS to produce the signal. The microcontroller interfaces to an LCD which is driven from the port pins.

The DDS generator produces a sine wave signal from a digital PWM output. Two pushbuttons increment or decrement the output frequency in steps of 10 Hz. The output frequency is shown on an LCD and also sent to the COM interface port. For this test a Mega32 type controller has been used, it has many I/O pins and is a popular choice. It would be simple to make changes to allow the program to run on other controllers from the ATmega family.

The principle

The block diagram in **Figure 2** gives an overview of the external components connected to the microcontroller. The PWM signal is output from pin OC1B, and passes through



Figure 1.
Bascom help for the config options.

a low-pass filter to produce a sine wave. The filter design is very simple but for test purposes is sufficient. A better solu-

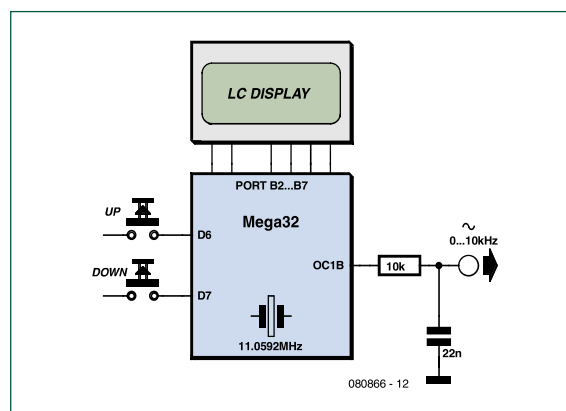


Figure 2.
Block diagram of the DDS generator.

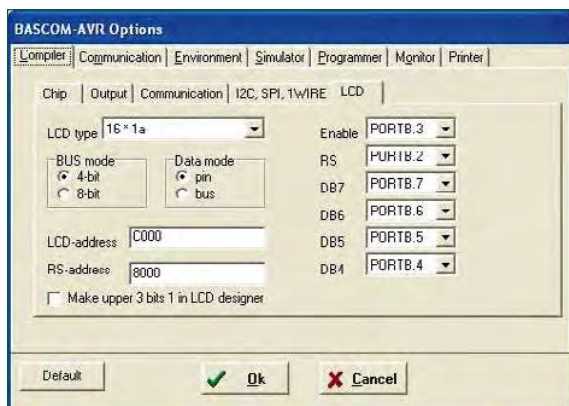


Figure 3.
The LCD setup page.

Listing 1

Initialising and writing to the LCD

```
Config Lcdpin = Pin , Db4 = Portb.4 ,
    Db5 = Portb.5 , Db6 = Portb.6 ,
    Db7 = Portb.7 , E = Portb.3 , Rs =
    Portb.2
Config Lcd = 16 * 2
Initlcd
Cls
Lcd "DDS"
```

Listing 2

Sine wave lookup table and frequency selection

```
For N = 1 To 256
    A = N - 1
    A = A * 3.1415
    A = A / 128
    B = Sin(a)
    B = B * 120
    B = B + 128
    Table(n) = Int(b)
Next N

Freq = 10
Do
    Locate 2 , 1
    Lcd Freq
    Lcd " Hz  "
    If Pind.6 = 0 Then
        Freq = Freq + 10
        Print Freq
    End If
    If Pind.7 = 0 Then
        Freq = Freq - 10
        Print Freq
    End If
    Waitms 10
    A = Freq
    '43200/65535
    B = A / 0.65918

    F = Int(b)
Loop
```

tion would use a (many-poled) filter with a much steeper response and a cut-off frequency of around 15 kHz. A simple piezo buzzer can be directly connected to the output without the need for any filter at all.

LC display

Whenever a design calls for an LCD to display lines of characters it can be achieved using six port pins for the drive signals. The LCD usually works in 4-bit mode with each data byte split into two before being sent to the display. Two control signals, E and RS, are also needed. In addition to these six signals we need three for the power supply and contrast setting.

The port pin assignments can be defined in the Menu Options/Compiler/LCD (**Figure 3**). A better method is to make the assignments in the source file (Config Lcdpin). This ensures that it runs successfully on different systems. It is also necessary to define the type of LCD used (Config Lcd = 16 * 2). When the system is first switched on one line of the LCD will be dark and the other light. After the display is initialised (Initlcd) the dark line becomes light. Characters (Lcd "Text") or a variable (LCD Freq) can now be sent to the display. After each character is written to the display the position is automatically incremented ready for the next character, but the second line does not automatically follow from the first. Writing to the second line it is necessary to define the position (Locate 2 , 5). The entire screen can be cleared at any time using the Cls command.

The example given in **Listing 1** writes the text 'DDS' to the first line. The frequency value is sent to the second line of the display (**Listing 2**) followed by the units 'Hz'. The displayed frequency value will not always have the same number of characters so it is necessary to include enough spaces to ensure that all characters previously displayed will be overwritten by the new value.

Sine table and frequency selection

A sine function lookup table must be written into memory for use by the DDS generator. The table size is 256 bytes. These represent the analogue values of the wave which are sent to the PWM output to produce the sine wave.

At start up the generator has an output frequency Freq = 10 Hz (**Listing 2**). The two pushbuttons PD6 and PD7 increment and decrement the output frequency in 10 Hz steps. The output frequency value is written to the two line display and each time the frequency is changed the new value is sent to the PC. Operation without an LCD is therefore possible. The generator can be used for example to tune an instrument; the standard 'A' note (440 Hz) produced on a tuning fork can be selected without problem. The frequency in Hz is scaled to give the variable F. Every change of F immediately affects the output frequency. This is made possible by the use of an interrupt routine.

Timer and DDS

The sine wave generator uses the DDS (Direct Digital Synthesis) principle with values of a sine waveform stored (as bytes) in a lookup table. A phase accumulator (the variable Accu) is increased by the value of the variable F to point to the next value in the lookup table. Only the high byte of the 16-bit Accu is used as a pointer to the table. When F has the value 1 it will therefore take 256 timer interrupts before the next value in the table is used and produces an output sine wave with a frequency of 0.65918 Hz. This is the resolution of the frequency generator. As the value of F

increases the pointer steps through the table more quickly. When its value reaches 256 the pointer will start to jump over individual values but the output will still be sinusoidal. At the highest frequency of 10 kHz only around four values are used to produce a complete sine wave cycle. The lowpass filter ensures that a good approximation to a sine wave will still appear at the output.

The program uses two timers. Timer 1 generates the 8-bit PWM signal. In this setup the PWM frequency is $11059200 \text{ Hz} / 256 = 43200 \text{ Hz}$. The 8-bit Timer 0 without any prescaler overflows at a rate of 43.2 kHz. This is therefore the rate at which the interrupt service routine is called, a new value is fetched from the lookup table and written to the PWM register before the next interrupt occurs (**Listing 3**).

Without any prescaler an 8-bit timer will interrupt every 256 clock cycles. Between interrupts the controller must not only execute all the instructions in the interrupt service routine, but also push all the working registers onto a stack and lastly pop them off again. In some cases the timing could be a little tight. It is important to be sure that there will be enough time to carry out all the activities. The simplest way to indicate how long the controller spends servicing the interrupt is to get it to set a port pin ($\text{Port}.0 = 1$) as it enters the ISR and reset it ($\text{Port}.0 = 0$) when it exits. With an oscilloscope probe on the pin we can now observe the mark/space ratio directly to see how much time is available. In the example here the pin is high for less than 50% of the time. The main routine can only execute its tasks when this waveform is low. Using a simple software delay like `Delaysms` will produce noticeably longer delay times than expected.

Two lines are 'commented out' in the interrupt routine. When these comment characters are removed the signal generator now has a sweep function. The frequency is incremented each time an interrupt occurs, the generator now sweeps from 0 to 10 kHz approximately three times per second. The oscilloscope display (**Figure 4**) shows the resulting output waveform after the low-pass filter. A piezo buzzer connected to the output will produce a characteristic twittering sound.

(080866-1)

Downloads and more info

Go to the project page at www.elektor.com/080866 for more information and the program downloads. We welcome your feedback in the Elektor forum.

BASCOM-AVR Reader Offer

Exclusive to Elektor readers, the download version of MCS BASCOM-AVR is now available at £ 55.00 (€ 69.00), a discount of more than 20% compared to the normal price of £ 71.00 (€ 89.00). As a bonus, pdf copies of all six BASCOM-AVR course instalments that appeared in Elektor are included with the download. The offer is valid from **19 January 2009 through 9 February 2009**. Further details at www.elektor.com/bascom-avr. US readers please check US\$ prices on website.

Listing 3 The DDS

```
Config Timer1 = Pwm , Prescale = 1 , Pwm = 8 ,
    Compare A Pwm = Clear Down ,
    Compare B Pwm = Clear Down
Config Timer0 = Timer , Prescale = 1
On Ovfl0 Tim0_isr
Enable Timer0
Enable Interrupts
Pwm1a = 127
Pwm1b = 0

Tim0_isr:
    'Timer 43.2 kHz at 11.0592 MHz
    Portb.0 = 1
    Accu = Accu + F
    N = High(accu)
    Pwm1b = Table(n)
    'F = F + 1
    'If F > 15000 Then F = 1
    Portb.0 = 0
Return
```

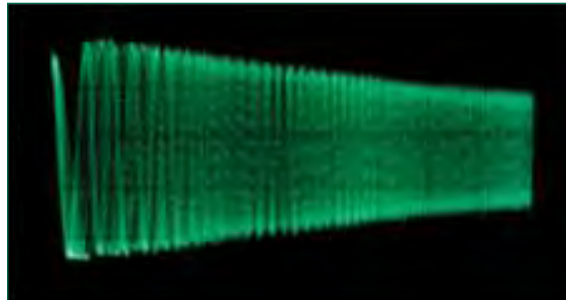


Figure 4.
Oscilloscope display of the swept output signal.