Digital Benchtop Power Supply (3)

part 3: the software

Design by R. Pagel

The digital benchtop power supply is controlled by a microcontroller programmed in PIC BASIC, while a Visual BASIC program is responsible for producing the control panel display on a PC.

Figure 1 shows the flowchart of the program in the microcontroller. At start-up a brief initialisation sequence runs, which resets the set values to zero and configures some of the microcontroller's pins. The next step, the measurement of the actual voltage and current values, already forms part of the main program loop. All the remaining parts of the program follow sequentially in this loop. One branch that can occur is when the push-buttons are being read. The procedure for reading the buttons is indeed as cumbersome as it (unavoidably) appears from the flowchart. The idea is to read each button in turn and, when one is found that is pressed, the microcontroller stores the corresponding key code. Finally, under `button pressed?' the microcontroller checks whether any button was in fact pressed. If so, a branch is taken to code which increases or decreases the appropriate set value, as long as the value remains within the permitted range. The new set point is then displayed. A half-second delay follows, before the push-buttons are scanned again. This provides an auto-repeat function. If no button is being pressed, the program branches back to the top of the main loop to measure the voltage and current again.

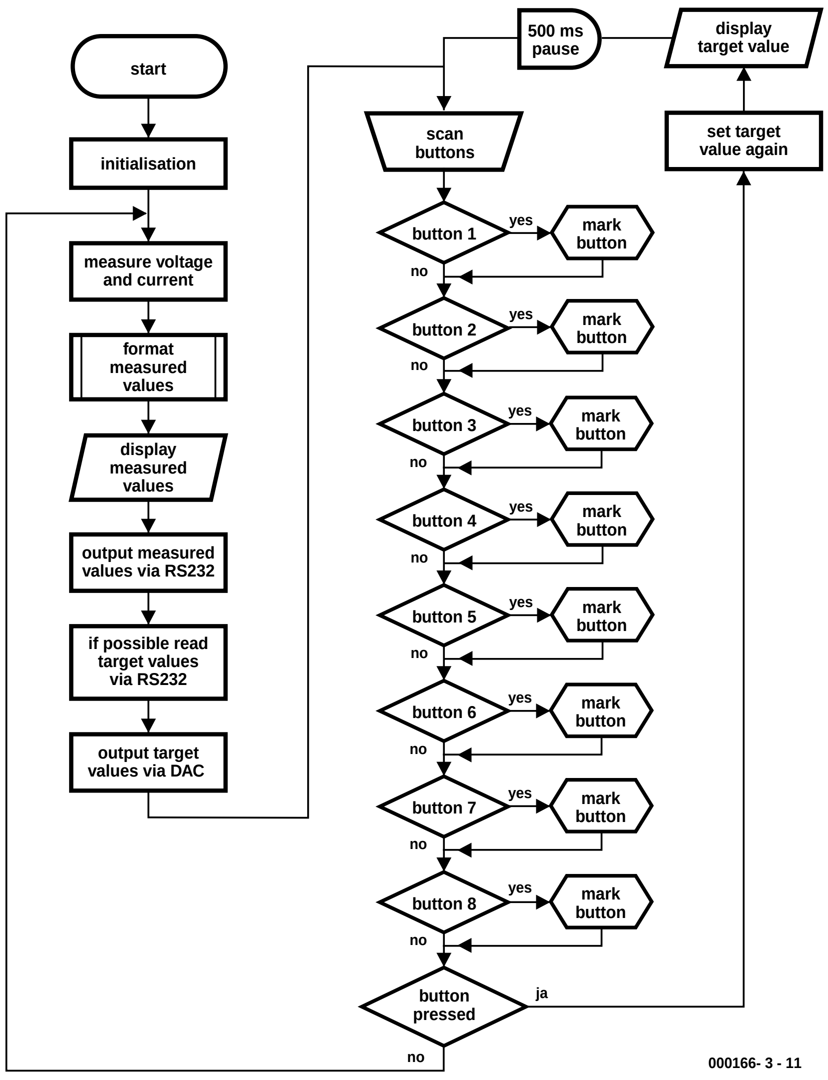

Figure 1. Flowchart for the microcontroller software.

BASIC Program

The source code listing for the microcontroller appears in **Figure 2**. The microcontroller program, written in PIC BASIC 1.3, can be downloaded from www.pic-basic.de.

PIC BASIC allows microcontroller programs to be written quickly and easily. It also

```
'D-PSU 25V, 2.5A or 20V, 1A

'attention: modifications to the program require that register
      numbers
'in the assembler subroutines are checked for changes!!!

'____________________________-
'declaring the variables
VarB Lh1, Lh2, Lh3, Lh5, Lh6, Lh7, Uvalue, Ivalue, y
VarB Buttonnumber, Accu, Callcounter, Bitpattern
VarW Meas_Voltage, Meas_Current

'____________________________-
'Main program
Init:
CV Uvalue, Ivalue  'set to 0 on each start'
Low A3           'ADC output at 0
High B2          'CTS: not ready to receive

Start:
'Measure voltage and current
'Using value 5??? allows ADC scale factor to be adjusted
'                              + - 20 equals approx. 1 digit
Low A4   'Mux to U
ADW A2, 5380, 0, Meas_Voltage  'Voltage measurement
Meas_Voltage = Meas_Voltage Shr 1   'equals / 2
High A4   'Mux to I
ADW A2, 5380, 0, Meas_Current  'Current measurement
Meas_Current = Meas_Current Shr 1   ' equals / 2   'line for
      2.5A
'Meas_Current = Meas_Current Shr 2   ' equals / 4   'line for
      1A

'Format measured values
Call Format

'Display measured values on LCD
LCD B5, " ", Lh1, Lh2, ",", Lh3, "V    ", Lh5, ",", Lh6, Lh7,
      "A "

'Send measured values over RS232
SerOut B3, 9600, "D", #Meas_Voltage, #Meas_Current, 13

'Allows a new target value to be received over RS232
Call RS232E

'Send target values over DAC
PWM A1, Uvalue, 64  'Set voltage (200 = 20V)
PWM A0, Ivalue, 64  'Set current (200 = 2A or 200 = 1A)


'Scan buttons

Entry:

Accu = %00010000  'Bit 4 High (reset by pressed button)
CV Callcounter, Buttonnumber
Call ButtonScan
Call ButtonScan
Call ButtonScan
Call ButtonScan
Call ButtonScan
Call ButtonScan
Call ButtonScan
Call ButtonScan
Branch Buttonnumber, Start, Button1, Button2, Button3, Button4,
      Button5, Button6, Button7, Button8


Button1:
If Uvalue > 240 Then Skip   'Line for 2.5A
'If Uvalue > 190 Then Skip   'Line for 1A
Uvalue = Uvalue + 10
Goto Display_Uvalue

Button2:
If Ivalue > 240 Then Skip   'Line for 2,5A
'If Ivalue > 190 Then Skip   'Line for 1A
Ivalue = Ivalue + 10
Goto Display_Ivalue
```

```
Button3:
If Uvalue < 10 Then Skip
Uvalue = Uvalue - 10
Goto Display_Uvalue

Button4:
If Ivalue < 10 Then Skip
Ivalue = Ivalue - 10
Goto Display_Ivalue

Button5:
If Uvalue > 249 Then Skip   'Line for 2,5A
'If Uvalue > 199 Then Skip   'Line for 1A
Inc Uvalue
Goto Display_Uvalue

Button6:
If Ivalue > 249 Then Skip   'Line for 2.5A
'If Ivalue > 198 Then Skip 2  'Line for 1A
Inc Ivalue
'Inc Ivalue  'Line for 1A (omit for 2.5A version)
Goto Display_Ivalue

Button7:
If Uvalue < 1 Then Skip
Dec Uvalue
Goto Display_Uvalue

Button8:
If Ivalue < 1 Then Skip   'Line for 2.5A
'If Ivalue < 2 Then Skip 2  'Line for 1A
Dec Ivalue
'Dec Ivalue  'Line for 1A (omit for 2.5A version)


Display_Ivalue:
'y = Ivalue Shr 1 'equals / 2  'Line for 1A (omit for 2.5A ver-
      sion)
LCD B5, "     ", #Ivalue, "0mA"  'Line for 2.5A
'LCD B5, "     ", #y, "0mA"  'Line for 1A
Pause 500
Goto Entry


Display_Uvalue:
LCD B5, "     ", #Uvalue, "00mV"
Pause 500
Goto Entry


'____________________________-

'Subroutines

'Depending on value in Callcounter, ButtonScan shifts one of
'eight bitpatterns to the pins of the HC164.
'Only the button at the pin with the 0 on it
'can pull PB4 Low. PB4 then indicates if a button was pressed
      or not,
'while Callcounter reveals the button identity


Sub ButtonScan
   LookUp Callcounter, %11101111, %11011111, %10111111,
      %01111111, %11111011, %11110111, %11111110, %11111101,
      Bitpattern
   EXPo B5, Bitpattern, 0  'only Button 0 of bit pattern can
      pull B4 Low
   Inc Callcounter
   PBI %00010000 = Accu        'read only bit 4 of Port B
   If Accu <> 0 then Skip        'skip when no button pressed
   Buttonnumber = Callcounter       'mark Button number
EndSub


'The Basic subroutine Read is called from
'assembler subroutine RS232E

Sub Read
   SerIn B0, 9600, #Uvalue, #Ivalue
   Uvalue = Uvalue Min 250           'limit to 25 Volt  'Line
```

Figure 2. Listing in PIC BASIC.

```
      for 2.5A
   'Uvalue = Uvalue Min 200           'limit to 20 Volt  'Line
      for 1A
   Ivalue = Ivalue Min 250           'limit to 2.5 Ampere
      'Line for 2,5A
   'Ivalue = Ivalue Min 200           'limit to 1 Ampere  'Line
      for 1A
   Y = 1          'Leave loop immediately
Endsub


'Assembler sub-routine Format employs the already available
'Resources for PB. It load the number registers Lh1-Lh8 with
      the
'ASCII values for Numbers 0-9 according to the values in the
'variables Meas_Voltage and Meas_Current.

'The auxiliary subroutine called Packer saves 8 bytes of pro-
      gram memory
'Packer calls machine code program SOSS°, which is contained in
      the
'PB compiler output, when the commands SerOut - #WordVar
'or LCD - #WordVar" was employed.
'It returns the decimal number equivalent of a wörd variable.
'It divides te value contained in HWERT2/R21 by the value from
      the
'jump table SOTT° (also contained in compiler output).
'The value(!) in the FSR has to be the ADD value
'of the jump table (Pos. 5 = 0, 4 = 2, 3 = 4, 2 = 6, 1 = Rest
      in R21).
'LWERT1 contains the ASCII code (characters 0-9) as the result.

Ass Format
      ;format voltage
   MOVF 24,W
   MOVWF HWERT2
   MOVF 23,W
   MOVWF 21
   MOVLW 2
   Call Packer
   MOVWF 27
   MOVLW 4
   Call Packer
   MOVWF 28
   MOVLW 6
   Call Packer
   MOVWF 29
      ;format current
   MOVF 26,W
   MOVWF HWERT2
   MOVF 25,W
   MOVWF 21
   MOVLW 2
   Call Packer
   MOVWF 30
   MOVLW 4
   Call Packer
   MOVWF 31
   MOVLW 6
   Call Packer
   MOVWF 32
   Return

Packer:      ;no repeating of lines; saves 8 bytes of program
      memory
   MOVWF FSR
   CALL SOSS°
   MOVF LWERT1,W
EndAss


'RS232E controls data reception at the interface. Each time it
      it called, the CTS line is pulled High for 1.5ms.
'If a character arrives via RxD within this period, the D-PSU
      goes into Receive mode i.e.
'subroutine Read is called. Next, 2 values with terminating CRs
'have to arrive at the interface before the controller is
      allowed
'to leave the subroutine

Ass RS232E
   CLRF 35        ;Clrf Y (= R35)
 RS232:
   BCF PB,2       ; CTS: ready to receive
   BTFSS PB,0     ; RxT pin test
   Call Read
   DECFSZ 35,F    ;
   GOTO RS232     ; repaet loop 256 times
   BSF PB,2       ; CTS: not ready to receive
EndAss
```

makes compiling the program and programming it into a chip easy. Further information on PIC BASIC, as well as the most up-to-date version of the program, can be found on the Internet at www.pic-basic.de. At the time of writing this article, the information on PIC Basic is only available in German. We hope that Mr. Pagel will eventually produce English translations.

Figure 3 shows in-system programming of the 1 A power supply using the PIC BASIC programmer.

First all the variables used in the program are declared. There are 13 byte-wide variables and two word-wide variables, occupying a total of 17 bytes of the microcontroller's RAM (and a further twelve bytes are reserved by PIC BASIC as a scratch area). Then follows the first part of the program: this is the part indicated in the flowchart by `initialisation'. The label **Start** marks the entry point for the main loop. The program is so thoroughly commented that a detailed description is not necessary here. A few remarks are, however, in order:

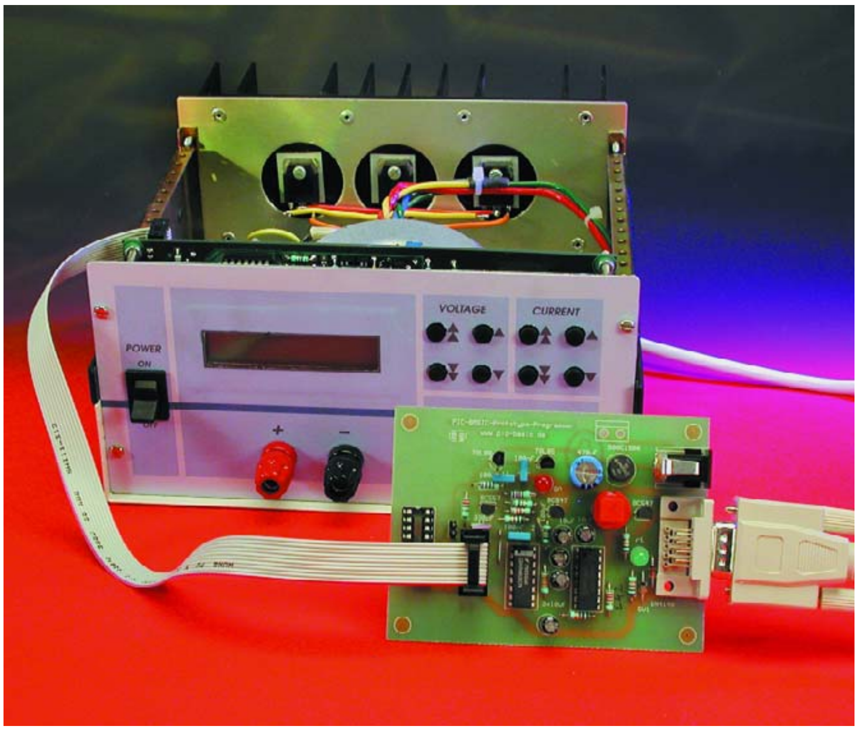

Figure 3. In-system programming of the power supply microcontroller.

A/D converter

The command `ADW A2, 5380, 0, Meas_Voltage` carries out an analogue-to-digital conversion and writes the measured value into the variable `Meas_Voltage` (measured voltage). The scale factor can be adjusted by changing the value `5380` above. The circuit is, however, designed so that this will not normally be necessary.

Assembly code subroutines

Under `Format measured values` the assembly code subroutine `Format` is called. This subroutine is 30 bytes long and uses some (PIC BASIC) subroutines, provided for the use of other BASIC commands, to format the measured values for display. Using this trick a large amount of precious program memory can be saved.

The eight-byte subroutine `RS232E` is also written in assembler. It sets the CTS signal high and waits for a period to check if data are being sent from the PC. If so, the assembler subroutine calls the BASIC subroutine `Read` which is responsible for actually reading the data in.

All the remaining parts of the program are written entirely in BASIC. The compiled code size for the complete program is either 1009 bytes or 1021 bytes (for the 2.5 A and 1 A versions respectively), and so just fits into the program memory of the PIC16F84. The program lines that need to be changed between the two versions of the power supply are marked in the BASIC program listing.

If it is desired to enable the watchdog timer in the microcontroller, the configuration word in the assembly output must be changed as follows:

```
CONFIG B'11111111110101'
```

Also, a `CLRWDT` instruction must be inserted at one point in the code, in the main loop and in the push-button scanning loop. The latter loop runs in just over 500 ms and the main loop runs in about 780 ms. With the values set in the Option register on power-up, a watchdog reset will occur after 2.3 s. This is enough time for both loops (indeed, even half the time would be adequate). The simplest place to put the `CLRWDT` instruction is immediately after the label `Entry`. The following code is then shifted down by one location, which makes no difference in this case.

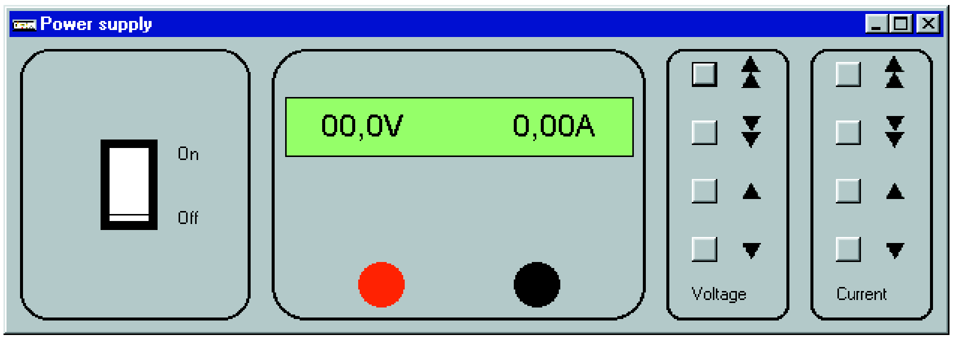

Figure 4. Power supply display on the PC's monitor.

The interface protocol

The data packet that the power supply sends out over its interface is structured as follows:

```
Duuuuuiiiii↵
```

First a `D` is sent, followed by five-digit values for voltage and current, and finally a carriage return character. The least-significant digit of the voltage value represents 10 mV, and that of the current value 1 mA. The two values are thus given to a resolution ten times higher than that shown on the LCD panel. The leading digit of the voltage and current values is always zero.

When set values are sent to the power supply, both voltage and current settings must be sent, one immediately after the other. In both cases up to 3 digits can be sent.

```
uuu↵ iii↵
```

After each value a non-digit character (for example a carriage return character) must be sent. For either version of the power supply a voltage value of 20 V must be sent as `200`. For a current of 1 A, a value of `100` should be sent in the case of the larger model, `200` in the case of the smaller model.

Visual BASIC program

The control program, written especially for this project, runs under Windows 95 or 98. Its control interface resembles the front panel of the power supply (**Figure 4**). Just as in reality, the voltage and current set values can be adjusted using the eight buttons. When the mouse button is released the values are sent to the power supply. The set and actual values are shown on a simulated LCD panel. If you click in a region of the main program window other than on the buttons, a settings window appears. Here the version of the power supply (2.5 A or 1 A) and the COM port (1 to 4) used for data communication can be configured. If a mouse is connected to COM1 it can happen that the program does not work correctly with COM3, but this is a common problem with PCs.

In the right-hand part of the settings window the names of a log file and of a control file can be specified. A click of the mouse on the adjacent `Start' button and a file is either read or written: the file contains the displayed readings along with a time stamp indicating when they changed (in the case of a log file), or when they are to be changed (in the case of a ready-prepared control file). In the simplest case the control file can be created from a log file by modifying the time stamps.

The following example one-line data record shows the format used for the log and control files:

```
#2000-08-20
14:35:53#,"04.9V","0.97A"
```

Between the two hash symbols ('#') we have the date (in international format) and the time when the indicated change occurred or is to occur. Within the record the time stamp and the two electrical values are separated from one another by commas. This allows for the processing and, for example, the graphical presentation of the contents of the file using a spreadsheet program.

(000166-3)