DS18(S)20 and DS18B20 tempsensors

2014-05-15

1 Contents

2 Purpose of this document 2

3 The examples output to uart 2

4 General characteristics of the DS18x20 device 3

5 Basic usage 3

5.1 Connecting the hardware 4

5.2 Detecting the device type 4

5.3 Reading the temperature value 4

5.3.1 Using the default temperature conversion time 6

5.3.2 Using the actual temperature conversion time 6

5.3.3 Temperature value format 7

5.4 CRC Checking of the temperature 8

5.5 Converting the temperature value to a string 9

5.6 Complete code example 10

6 Extended Usage 11

6.1 Parasite Power 11

6.2 The DS18x20 configuration 13

6.2.1 Setting the configuration 13

6.2.2 Getting the Configuration 13

6.3 Multiple devices on one 1-wire bus 14

6.3.1 The hardware connection diagrams 14

6.3.2 The ROM code 15

6.3.3 Getting the ROM code of a single connected device 15

6.3.4 Getting the ROM codes of multiple connected devices 15

6.3.5 CRC checking a ROM code 19

6.3.6 Using the ROM code 19

6.3.7 Using the non ROM library routines 20

6.4 Alarms 20

6.4.1 Setting the alarm levels 20

6.4.2 Checking for alarms 21
7 Appendixes 21
7.1 The DS1820 Library 22
7.2 The OW_Utilities library 25
7.3 Ds18x20 datasheets 25

2 Purpose of this document

- Describe useful knowledge for an mP (or mB or mC) user of the digital temperature sensor of the DS18x20 family (DS1820, DS18S20 and DS18B20). This document is however no replacement of the DS18x20 datasheet (see section 7.3)
- Describe, in the examples, the usage of the DS1820 library and the OW_Utilities library written in mikroPascal.

3 The examples output to uart

Most examples shown in this document send information (e.g. the outcome of a measurement) via the PIC's rs232 serial interface (uart) to the PC with a baud rate of 115200 baud. There it can be observed with e.g. the Usart Terminal tool of the mE compiler's IDE or HyperTerminal. Of course it is also possible to display the outcome on an LCD in stead of sending it via the uart.

If using the uart as output device, and assuming the PIC is of the P18F2620 type, the uart init code looks as follows:

```
begin
  { Main program }

  AdCon1 := $0f; // all inputs digital

  Uart1_init(115200);
  delay_ms(200);
  Uart1_write_text(#13 + #10); // CRLF
  Uart1_write_text('Started');
  Uart1_write_text(#13 + #10); // CRLF

  // the actual code goes here
  ...
end.
```

4 General characteristics of the DS18x20 device

The types **DS1820** and **DS18S20 are actually the same**, and will be treated as such in this document (mentioned further as DS18S20).

All three types are 1-wire devices, and can be driven both with external or parasite power (powered from dataline).

Type	Temp Range	Accuracy	Resolution (bits) [1]	Resolution °C	Max Temperature Conversion Time (ms)
DS18S20	-55..+125°C	±0.5°C[2]	9	0.5	750
DS18B20	-55..+125°C	±0.5°C[2]	9	0.5	94
			10	0.25	188
			11	0.125	375
			12	0.0625	750

As one can see, the accuracy of both device families is the same (±0.5°C), but the DS18B20 is 4 times faster than the DS18S20 for the 9 bits resolution.


The temperature conversion time in the above table is a maximum value, the actual conversion takes less (or equal) time. See section The temperature conversion time for more details (a.o. checking for actual “temp conversion done”).

5 Basic usage

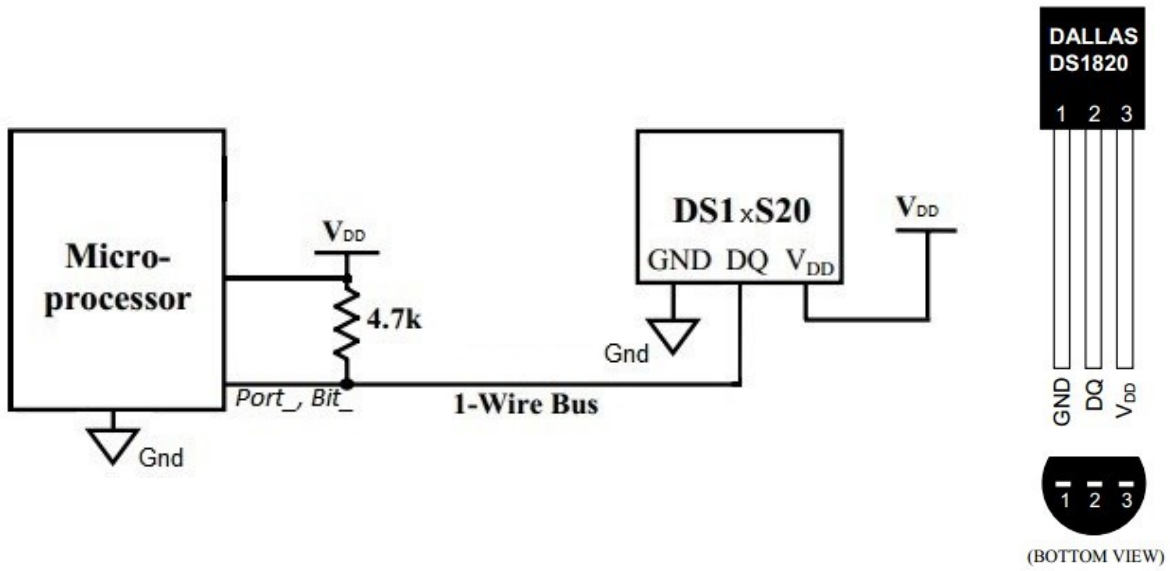
Simple usage of the DS18x20 usually consists of the following steps:

- Connecting the hardware (section 5.1)
- Reading the temperature value (section 5.2) out of the DS18x20, eventually followed by a CRC check (section 5.4)
- Converting the temperature value to a string (section 5.5) that can be displayed e.g. on an (ascii text based) LCD display.

Simple usage also assumes only one DS18x20 device is present on a one-wire bus, and the device has external power (see the Ds18x20 datasheet and diagram below).

[1] user programmable for the DS18B20, constant for the DS18S20
[2] only guaranteed in the range of -10°C to +85°C

5.1 Connecting the hardware



As one can see there is, besides the actual "1-wire bus" connection, only need for pull-up resistor of 4.7K plus a ground and a Vdd connection to the Ds18x20. The latter is called "External Power" in the datasheets (see section 7.3). The Vdd connection can by avoided by using the parasite Power possibility, see section 6.1.
In the examples show in this document the 1-wire bus is connected to PortA, bit 0 of the PIC.

5.2 Detecting the device type

This can be done using the procedure "DS1820_Family" from the DS1820 Library:

```
var Ch: char;

  Ch := DS1820_Family(PortA, B0);
  case Ch of
    'S': uart1_write_text('DS18S20');
    'B': uart1_write_text('DS18B20')
    else uart1_write_text('unknown');
  end;
  Uart1_write_text(#13 + #10);
```

5.3 Reading the temperature value

This is done in 3 steps:

1. Start the temperature conversion
2. Wait until the conversion in the DS18x20 device is finished (temperature conversion takes a considerable amount of time), see time, section 5.3.1 for more details

3. Read the temperature from the DS18x20 device

This is the code by using the DS1820 library:

```
var Temperature: integer;
    Strng: string[20]; // make it large enough

// Uart init code goes here (see The examples output to uart)
while true do
begin
  // 1. start temperature conversion
  DS1820_StartTempConversion(PortA, B0, false); // no waiting for temp
                                                // conversion in the
                                                // DS18x20

  // 2. wait until the conversion is finished
  delay_ms(750);

  // 3. read the temperature from the DS18x20 device
  Temperature := DS1820_ReadTemperature(PortA, B0);

  // 4. convert the temperature to a displayable string
  // and send the string to the output
end;
```

The temperature read with above code will be in integer variable “Temperature”. For the actual format of the temperature in this variable, see section Output format, section 5.3.3.
PortA and Bit B0 are the I/O port and its bit to which the DS18x20 device is connected (the “1-wire bus”).


Some remarks:

- above code assumes only one DS18x20 device is connected to the 1-wire bus
- Actual “waiting” for the DS18x20 device while it is converting the temperature is not necessary, the MCU can do something else in the mean time, as long there is e.g. 750 ms between starting the temperature conversion and reading the temperature from the DS18x20.


In stead of using DS1820_ReadTemperature the routine DS1820_ReadTemperature_Fast can be used. The advantage is that the latter is faster: only the necessary data (2 bytes) is read from the DS18x20 in stead of the whole data (9 bytes). One drawback: temperature CRC check can not be done, see section 5.4.

5.3.1 Using the default temperature conversion time

This is the method of simply "waiting" the minimum time needed for the temperature conversion between starting the conversion and reading the Ds18x20 temperature.

The maximum needed conversion times are:

Type	Resolution	Max conversion time (milliseconds)
DS18S20	--	750
DS18B20	9 bits	94
	10 bits	188
	11 bits	375
	12 bits	750

As one can see the needed conversion time depends on the DS18x20 type, and in case of the DS18B20, of the resolution used.


If reading the temperature before the conversion is done the previous converted temperature (or the DS18x20's default 85°C) will be read.

A drawback of this method is that usually the actual conversion time will be smaller than the maximum needed one, leading to a loss of time.

For a code example, see Reading the temperature value (section5.3)

5.3.2 Using the actual temperature conversion time

This is the method described in the DS18B20.pdf datasheet, section "Operation – Measuring temperature": *If the DS18B20 is powered by an external supply, the master can issue "read time slots" (see the 1-Wire Bus System section) after the Convert T command and the DS18B20 will respond by transmitting 0 while the temperature conversion is in progress and 1 when the conversion is done.*
This can be done by using the function "DS1820_TempConversionReady" from the DS1820 library.

Example:

```
var Temperature: integer;

while true do
begin
  // start temperature conversion
  DS1820_StartTempConversion(PortA, B0, false);

  // wait until the temperature conversion is actually over
  repeat until DS1820_TempConversionReady(PortA, B0);
```

```
  // read the temperature
  Temperature := DS1820_ReadTemperature(PortA, B0);

  // do something with the read "Temperature".
  ...
end;
```

The above code can be simplified, because the procedure "DS1820_StartTempConversion" can be called with its last parameter set to "true", which makes it wait until the actual temperature conversion is done:

```
var Temperature: integer;

while true do
begin
  // start temperature conversion
  DS1820_StartTempConversion(PortA, B0, true); // will wait until the
                                               // temperature conversion is
                                               // actually over

  // read the temperature
  Temperature := DS1820_ReadTemperature(PortA, B0);

  // do something with the read "Temperature".
  ...
end;
```


The method described above can not be used when using parasite power, see section 6.1.


In stead of using DS1820_ReadTemperature the routine DS1820_ReadTemperature_Fast can be used. The advantage is that the latter is faster: only the necessary data (2 bytes) is read from the DS18x20 in stead of the whole data (9 bytes). One drawback: temperature CRC check can not be done, see section 5.4.

5.3.3 Temperature value format

Both device types deliver 2 bytes as the temperature reading in integer format, with the fractional value integrated into the least significant bits.

5.3.3.1 The DS18S20

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
←	Integer part of the temperature reading													→	

Fractional part of the temperature reading ↑

Temperature weight of each bit (°C):

Sign bits	64	32	16	8	4	2	1	0.5

So, a value of 0000 0000 0011 0010 binary (33 hex) represents a temperature of 16 + 8 + 1 +0.5 = 25.5 °C.

5.3.3.2 The DS18B20

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
←	Integer part of the temperature reading										→	←			→
												Fractional part of the temperature reading↑			

Temperature weight of each bit (°C):

Sign bits	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625

So, a value of 0000 0001 1001 0001 binary (191 hex) represents a temperature of 16 + 8 + 1 + 0.0625 = 25.0625 °C.

The width of the fractional part is always 4 bits, irrespective of the device's resolution.

5.4 CRC Checking of the temperature

The temperature read from the DS18x20 device can be checked on validity with function "DS1820_CheckCRC" of the DS1820 Library. It should be used after "DS1820_ReadTemperature" (or its ROM code equivalent "DS1820_ReadTemperatureROM") and it returns zero if the CRC check is Ok, meaning the temperature read is valid.

Example:

```
var CRCOk: byte;
    Temperature: integer;
 ...

  Temperature := DS1820_ReadTemperature(PortA, B0);

  CRCOk := DS1820_CheckCRC;

  if CRCOk > 0 then // CRC error
  begin
    Uart1_write_text('Temperature CRC error');
    Uart1_write_text(#13 + #10);
  end
  else
  begin
    // process the valid temperature
  end;
```


The CRC check (see section 5.4) can not be done after using the “Fast” functions to read the temperature. Reason: This function only read the data of the DS18x20 partially, while the CRC check is on the whole DS18x20 data.

5.5 Converting the temperature value to a string

The purpose of this conversion is to be able to display the DS18x20 temperature on e.g. an LCD (or any other ascii string/character based device) or via serial communication.
The procedures for the DS18S20 and the DS18B20 is different. For the latter 2 possibilities are provided.

This is the code using the DS1820 library:
For the DS18S20:

```
var Strng: string[5]; // make it large enough
    ...
  // convert the temperature to a displayable string
  DS1820_TempToString(Temperature, Strng, ',');
```

For the DS18B20:
The conversion to a string for a temperature read from a DS18B20 is slightly different:

```
var Strng: string[5]; // make it large enough

  // Convert the temperature to a displayable string
  // Only one decimal of the temperature present in the string [3]
  DS18B20_TempToString(Temperature, Strng, ',');
```

[3] The string will always contain 1 fractional digit, irrespective of the DS18B20 resolution set

or (DS18B20, high resolution)

```
var Strng: string[9]; // make it large enough

  // convert the temperature to a displayable string
  // Four decimals of the temperature present in the string [4]
  DS18B20_TempToString_HR(Temperature, Strng, ',');
```

In all three cases the displayable result will reside in the variable “Strng”.

This string can then be sent to the uart using:

```
// and send the string to the output (uart in this case)
  Uart1_write_text(Strng);
  Uart1_write_text(#13 + #10);
```

5.6 Complete code example

Here is a complete code example with usage of the DS1820 Library functions and procedures used so far:

```
uses DS1820;

var Ch: char;
    var Temperature: integer;
    Strng: string[9]; // make it large enough for the DS18B20 output string
begin // main code

  // detect the device type
  Ch := DS1820_Family(PortA, B0);
  case Ch of
    'S': uart1_write_text('DS18S20');
    'B': uart1_write_text('DS18B20')
    else uart1_write_text('unknown');
  end;
  Uart1_write_text(#13 + #10);

  while true do
  begin
    // start temperature conversion
    DS1820_StartTempConversion(PortA, B0, true); // wait for actual
                                                 // conversion finished

    //read the temperature from the DS18x20 device
    Temperature := DS1820_ReadTemperature(PortA, B0);

    // Check the temperature validity
    CRCOk := DS1820_CheckCRC;

    if CRCOk > 0 then // CRC error
    begin
```

[4] The string will always contain 4 fractional digits, irrespective of the DS18B20 resolution set

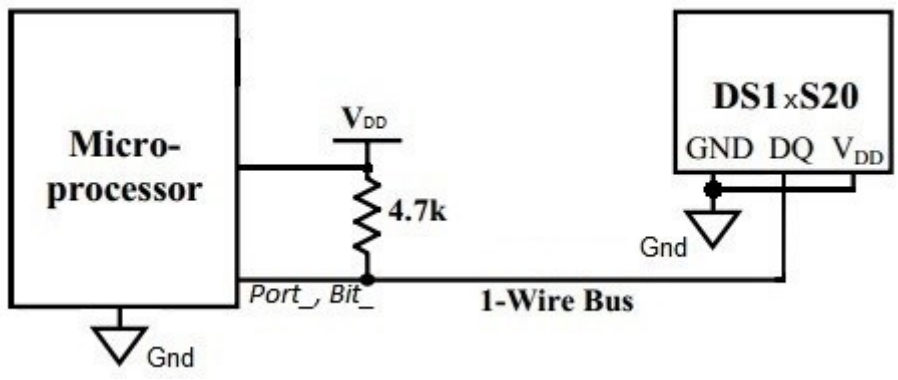
```
      Uart1_write_text('CRC error');
      Uart1_write_text(#13 + #10);
    end
    else
    begin // valid temperature read
      // convert the temperature to a displayable string
      case Ch of
        'S': DS1820_TempToString(Temperature, Strng, ',');
        'B': DS18B20_TempToString_HR(Temperature, Strng, ',');
      end;

      // and send the string to the output (uart in this case)
      Uart1_write_text(Strng);
      Uart1_write_text(#13 + #10);
    end;
  end;

end. // end of main code
```

6 Extended Usage

6.1 Parasite Power

The DS18x20 can function with its Vdd line connected to ground in stead of Vdd. In this case the DS18x20 is powered via the pull up transistor of 4.7KΩ. This saves 1 wire from PIC to Ds18x20 device, reducing the number of wires to 2 (but still called the 1-wire bus).

This connection diagram using parasite power:



The only issue here is that the DS18x20 consumes a rather high current during temperature conversion, which is not possible via the pull up resistor. To ensure that the device gets enough power via the 1-wire bus the PIC must pull that line high during the whole temperature conversion time.

Since the 1-wire line must be pulled high within 10us after the "DS1820_StartConversion" command, the normal mE OW_write routine can not be used to give this command (needs too much time), so a special routine "DS1820_StartTempConversion_**PP**" exists in the DS1820 Library.

Its usage:

```
DS1820_StartTempConversion_PP(PortA, B0, 750); // start temp conversion
                                               // with parasite power and
                                               // wait for 750 millisecs
// read the temperature from the DS18x20 device
Temperature := DS1820_ReadTemperature(PortA, B0);
```

This routine call above will start the temperature conversion, pull up the 1-wire line, wait 750 ms (its default conversion time), release the 1-wire line and exit. Always the default conversion time has to be used here because the 1-wire bus does not allow any testing during parasite power pull up.

There is also the possibility of skipping the wait time, by setting the wait value (last parameter) to zero. In this case the using program is responsible for not reading the temperature before the conversion is done.
e.g.:

```
DS1820_StartTempConversion_PP(PortA, B0, 0); // start temp conversion
                                             // with parasite power
                                             // no waiting

// wait until the conversion is finished
delay_ms(750);

// read the temperature from the DS18x20 device
Temperature := DS1820_ReadTemperature(PortA, B0);
```


Remark: Actual "waiting" for the DS18x20 device while it is converting the temperature is not necessary, the MCU can do something else in the mean time, as long there is e.g. 750 ms between starting the temperature conversion and reading the temperature from the DS18x20.


The "parasite power" conversion method can also be used for devices that are externally powered. The only drawback is that default conversion wait times have to be used.


There is a possibility to detect if the Ds18x20 device has external (Vdd) or uses parasite power: with the routine "DS1820_ReadPowerSupply" in the DS1820 Library. This functions returns zero if the Ds18x20 uses parasite power, and 1 if the device has external power.
Example:

```
if DS1820_ReadPowerSupply(PortA, B0) = 0
then Uart1_write_text('Parasite Power')
else Uart1_write_text('External Power');
Uart1_write_text(#13 + #10);
```

6.2 The DS18x20 configuration

The "configuration" of the DS18x20 are all values that the user can "program" into the device. These values are:

- The alarm temperature limit high (see Alarms, section 6.4 for its usage)
- the alarm temperature limit low (see Alarms, section 6.4 for its usage)
- for the DS18**B**20 only: the resolution at which the device should operate.

6.2.1 Setting the configuration

6.2.1.1 The DS18S20

The Ds18S20configuration can be set with the routine "DS1820_SetConfiguration" from the DS1820 Library.

Example:

```
DS1820_SetConfiguration(PortA, B0, +50, +5);
```

The value '+50' is the upper alarm limit, the '+5' is the lower one.

6.2.1.2 The DS18B20

The Ds18B20 configuration can be set with the routine "DS18B20_SetConfiguration" from the DS1820 Library.

Example:

```
DS18B20_SetConfiguration(PortA, B0, +50, +5, 9);
```

The value '+50' is the upper alarm limit, the '+5' is the lower one, and the '9' is the wanted resolution.

6.2.2 Getting the Configuration

6.2.2.1 Of the DS18S20

The configuration of an DS18S20 device can be read with routine "DS1820_GetConfiguration":
Example:

```
var THigh, TLow: short;
    Strng: string[10];
...
```

```
// read the configuration
DS1820_GetConfiguration(PortA, B0, THigh, TLow);

// and send it to the output to show
Uart1_write_text('THigh: ');
ShortToStr(THigh, Strng);
Uart1_write_Text(Strng +#13 + #10);
Uart1_write_text('TLow : ');
ShortToStr(TLow, Strng);
Uart1_write_Text(Strng +#13 + #10);
```

6.2.2.2 Of the DS18B20

This is done with the DS18B20_GetConfiguration routine.

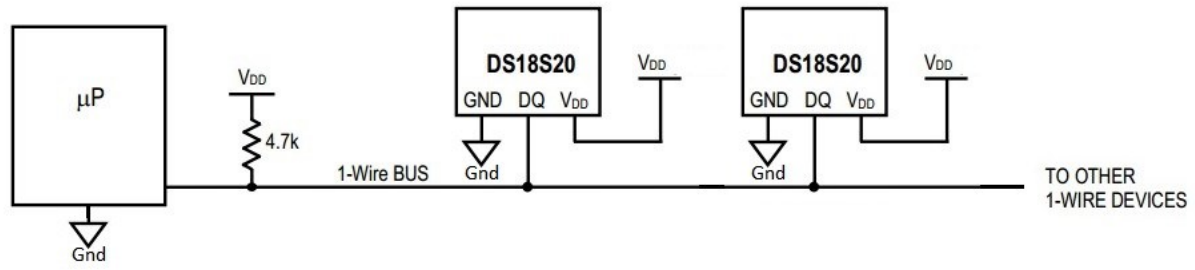
Example:

```
var THigh, TLow: short;
    Resolution: byte;
...
// get the DS18B20 configuration
DS18B20_GetConfiguration(PortA, B0, THigh, TLow, Resolution);

// and send it to the output to show
Uart1_write_text('THigh: ');
ShortToStr(THigh, Strng);
Uart1_write_Text(Strng +#13 + #10);
Uart1_write_text('TLow : ');
ShortToStr(TLow, Strng);
Uart1_write_Text(Strng +#13 + #10);
Uart1_write_text('Res  : ');
byteToStr(Resolution, Strng);
Uart1_write_Text(Strng +#13 + #10);
```

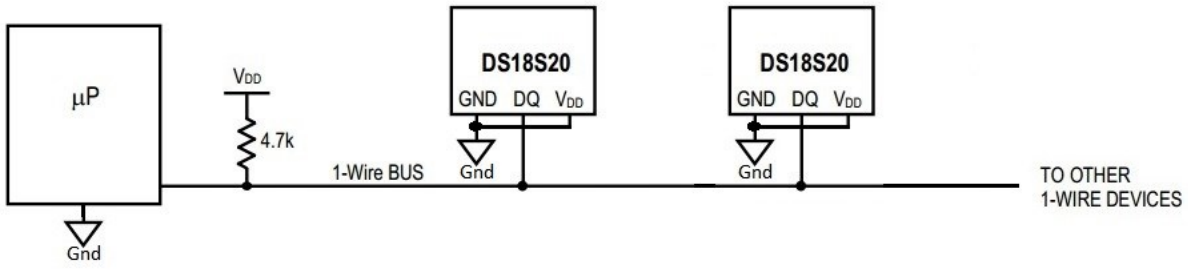
6.3 Multiple devices on one 1-wire bus

6.3.1 The hardware connection diagrams

6.3.1.1 External power



6.3.1.2 Parasite Power



6.3.2 The ROM code

When connecting more than 1 device on the 1-wire bus it is absolutely necessary (e.g. when reading the temperature from one of the devices) to identify the device accessed. This is done with the so-called ROM code.

The ROM code is **unique** for every DS18x20 device (in fact for every 1-wire device). Its size is 8 bytes. It includes a CRC byte, so CRC checking can be done easily, see section 6.3.5.
Using the ROM code of a DS18x20 device is always necessary when more than 1 device is connected to the same 1-wire bus. See more details in Multiple devices on one 1-wire bus (section 6.3).

6.3.3 Getting the ROM code of a single connected device

Assuming there is only 1 DS18x20 device on the 1-wire bus, its ROM code can simply be read with:

```
var RomCode: array[8] of byte;
...
DS1820_ReadROM(PortA, B0, RomCode);
```

After above call the Romcode of the DS18x20 device will reside in the variable “RomCode”.


For situations with more than 1 DS18x20 device on the 1-wire bus this method can not be used. See Multiple devices on one 1-wire bus (section 6.3).


The validity of the ROM code read can be done with a CRC check, see CRC checking of the ROM code, section 6.3.5.

6.3.4 Getting the ROM codes of multiple connected devices

6.3.4.1 With the “Search” method from Maxim

The OW_Utilities library contains procedures to detect all DS18x20 devices on a certain 1-wire bus. The routines are “OW_Search_First_ROM” and “OW_Search_Next”.

The mikroPascal code in the library is derived from the C code provided by Maxim, see here.


One important remark: these functions will give you a table with a number of ROM codes of devices present on the 1-wire bus, but no association of those codes with the physical devices. In other words: you do not know which ROM code belongs to which DS18x20 device.

Example of usage:

```
type Device = array[8] of byte;   // a romcode is 8 bytes long
var  Devices: array[5] of Device; // max 5 devices to find here
     Success: boolean;
     DeviceCount: byte;

  ...
  DeviceCount := 0;
  Success := OW_Search_First_ROM(PortA, B0);
  while (DeviceCount < 5) and Success do
  begin
    // the romcode is in variable ROM_NO, see unit OW_utilities
    memcpy(@Devices[DeviceCount], @Rom_No, 8); // copy it to our own table
    inc(DeviceCount);
    Success := OW_Search_Next(PortA, B0);
  end;
```

Both OW_Search_First_ROM and OW_Search_Next leave the ROM code found in variable "Rom_no" (provided the function returned true).

Above code fetches the ROM code of maximum 5 devices. The number of devices found resides in "DeviceCount", the ROM codes of the devices found resides in "Devices".

To complete the example, the ROM codes can then be shown via the uart with:

```
if (DeviceCount > 0)
then for I := 0 to (DeviceCount - 1 ) do
begin
  ShowRomCode(Devices[I]);
Uart1_write_text(#13 + #10);
end
else Uart1_write_text('No devices found' +#13 + #10);
```

The actual procedure to send a ROM code to the uart output:

```
procedure ShowRomCode(var Code_: array[8] of byte);
var Str: string[2];
    I: byte;
begin
  for I := 0 to 7 do
  begin
    bytetohex(Code_[I], Str);
    uart1_write_text(Str);
  end;
end;
```


The validity of the ROM code read can be done with a CRC check, see CRC checking of the ROM code, section 6.3.5.

6.3.4.2 With the Manual method

This method uses the "DS1820_ReadROM" and the "DS1820_CheckCRCRomCode" procedures. Since the first one only works well when only one DS18x20 device is connected to the 1-wire bus all devices have to be connected in sequence manually while using this method.

Due to the manual nature of this method it will be performed usually only once (e.g. during the setup of the temperature measurement system): the ROM codes detected can e.g. be written in the PIC's EEPROM, and reloaded on system startup.

Additionally, with this method it is obvious which ROM code is associated with a certain DS18x20 device on the 1-wire bus.

Example code:

```
type Device = array[8] of byte;   // a romcode is 8 bytes long
var  Devices: array[5] of Device; // max 5 devices to find here
     Success: boolean;
     DeviceCount, I: byte;

  DeviceCount := 0;
  while DeviceCount < 3 do // 3 devices expected
  begin

    // prompt to user
    ByteToStr(DeviceCount, Strng);
    Uart1_write_text('Insert Device '+ Strng + #13 + #10);

    // wait for DS18x20 becomes present
    I := 0;
    repeat
      Ds1820_ReadRom(PortA, B0, RomCode);
      if DS1820_CheckCRCRomCode(RomCode) = 0 // correct romcode
      then inc(I)
      else I := 0;
      delay_ms(10);
    until I = 5;  // correct rom code read 5 times with 10 ms interval

    memcpy(@Devices[DeviceCount], @RomCode, 8); // copy it to our own table

    inc(DeviceCount);

    // prompt to user
    Uart1_write_text('Remove Device '+ Strng + #13 + #10 +#13 + #10);

    // wait for a DS18x20 removed
    I := 0;
    repeat
      Ds1820_ReadRom(PortA, B0, RomCode);
      if DS1820_CheckCRCRomCode(RomCode) > 0 // incorrect RomCode
      then inc(I)
      else I := 0;
      delay_ms(10);
    until I = 5;  // incorrect rom code read 5 times with 10 ms interval

  end;
```

In above example code 3 devices are expected. It does not show the storage and the recall into/from the PIC's EEPROM, only the detection and the ROM code storage in table "Devices".

To complete the example, the ROM codes can then be shown via the uart with:

```
for I := 0 to (DeviceCount - 1 ) do
begin
  ShowRomCode(Devices[I]);
  Uart1_write_text(#13 + #10);
end;
```

For the actual procedure to send a ROM code to the uart output: see ShowRomCode.

See also next section (6.3.5) about ROM code CRC checks.

6.3.5 CRC checking a ROM code

The ROM code read from the DS18x20 device can be checked on validity with function "DS1820_CheckCRCRomCode". It returns zero if the CRC check is Ok, meaning theROM code read is valid.

Example:

```
var RomCode: array[8] of byte;
...

  // the ROM code to be checked is present in the variable "RomCode" here

  if DS1820_CheckCRCRomCode(RomCode) > 0 then
  begin // romcode error
    Uart1_write_text('ROM code CRC error');
    Uart1_write_text(#13 + #10);
  end
  else
  begin // valid romcode
    // use the valid rom code
  end;
```

6.3.6 Using the ROM code

Once the ROM code of a device is known its usage is very straightforward: use the DS1820 Library function X*xx**ROM*** in stead of the simple *Xxx* one:

Example:

```
var RomCode: array[8] of byte;
    Temperature: integer;
    Ch: char;
    Strng: string[9];
...

  //  the variable "RomCode" is supposed to hold a valid ROM code
```

```
DS1820_StartTempConversionROM(PortA, B0, true, RomCode);
// in stead of "DS1820_StartTempConversion(PortA, B0, true);"

Temperature := DS1820_ReadTemperatureROM(PortA, B0, RomCode);
// in stead of DS1820_ReadTemperature(PortA, B0);

Ch := DS1820_FamilyROM(RomCode);
// in stead of DS1820_Family(PortA, B0);
case Ch of
  'S': DS1820_TempToString(Temperature, Strng, ',');
  'B': DS18B20_TempToString_HR(Temperature, Strng, ',');
end;
```

The "ROM" version routines are:

DS1820_StartTempConversionROM	in stead of	DS1820_StartTempConversion
DS1820_ReadTemperatureROM	in stead of	DS1820_ReadTemperature
DS1820_ReadTemperatureROM_Fast	in stead of	DS1820_ReadTemperature_Fast
DS1820_FamilyROM	in stead of	DS1820_Family
DS1820_SetConfigurationROM	in stead of	DS1820_SetConfiguration
DS18B20_SetConfigurationROM	in stead of	DS18B20_SetConfiguration
DS1820_GetConfigurationROM	in stead of	DS1820_GetConfiguration
DS18B20_GetConfigurationROM	in stead of	DS18B20_GetConfiguration
DS1820_StartTempConversionROM_PP	in stead of	DS1820_StartTempConversion_PP

6.3.7 Using the non ROM library routines

There are a number of non ROM routines that can be used in a multiple device 1-wire bus system. When using these routines the outcome is usually that all DS18x20 devices on the bus react.

Those routines that are sensible to use:

DS1820_StartTempConversion: will start temp conversion in ALL ds18x20 devices

DS1820_StartTempConversion_PP: will start temp conversion in ALL ds18x20 devices, parasite power mode

DS1820_TempConversionReady: will return true if ALL devices are ready

DS1820_ReadPowerSupply: will return "1" if ALL devices have external power

Other non ROM routines have no sensible usage in a multi device environment.

6.4 Alarms

The possibility exists of setting maximum and minimum temperature alarm levels, and check relatively easy if the temperature measured by the device is equal to or outside the limits set.

6.4.1 Setting the alarm levels

This is done with the "DS1820_SetConfiguration" and the "DS18B20_SetConfiguration" routines, see Setting the configuration, section 6.2.1.

The Alarm level resolution is always 8 bits (1°C), for both DS18S20 and DS18B20 devices.

The alarm levels are called TH and TL in the datasheets (see section 7.3) and are “shorts” (signed bytes).

6.4.2 Checking for alarms

This is done by using “OW_Search_First_Alarm” and “OW_Search_Next” from the OW_Utilities library. The routines give back the ROM code of the devices in alarm state: the measured temperature is equal to one of the alarm levels or above the maximum level or below the minimum level.

Example:

```
var Success: boolean;
...
  // make all Ds18x20 do a temperature conversion
  DS1820_StartTempConversion(PortA, B0);

  // wait until all conversions are completed
  repeat until DS1820_TempConversionReady(PortA, B0);

  // check for any alarms (uses OW_Utilities)
  Success := OW_Search_First_Alarm(PortA, B0); // search first alarm
  while Success do
  begin
    uart1_write_text('Alarm: ');
    ShowRomCode(ROM_NO);
    uart1_write_text(#13 + #10);
    Success := OW_Search_Next(PortA, B0); // search next alarm
  end;
```

Both `OW_Search_First_Alarm` and `OW_Search_Next` leave the ROM code (of the DS18x20 in alarm state) found in variable “Rom_no” (provided the function returned true).

For the actual procedure to send a ROM code to the uart output: see ShowRomCode.

Important Remarks:

- A DS18x20 device can only signal an alarm state after a completed temperature conversion.
- After detecting an alarm state in a device (its ROM code being returned by one of the above procedures), the alarm state is cleared, and eventually raised again after the next temperature conversion.

7 Appendixes

7.1 The DS1820 Library

See LibStock: http://www.libstock.com/projects/view/104/tempsensors.

The signature of the procedures and functions:

```
// interface

{$IFDEF P24}
type PortType = word;
{$ELSE}
type PortType = byte;
{$ENDIF}

function DS1820_StartTempConversion(var Port_: PortType; Bit_: byte; Wait:
boolean): boolean;
  // Starts the temperature conversion of a DS1820 connected to Port "Port"."Bit".
  // If "Wait" is true, the function waits until the conversion is completed.
  // Returns true if success.

function DS1820_StartTempConversion_PP(var Port_: PortType; Bit_: byte; Wait:
integer): boolean;
  // Same as above, but with "parasite" power (power via the dataline during
conversion)
  // Here also the "Wait" time (in millisecs) has to be specified:
  //  minimum 750 for DS18(S)20 or DS18B20 in 12 bits resolution
  //  minimum 375 for DS18B20 in 11 bits resolution
  //  minimum 188 for DS18B20 in 10 bits resolution
  //  minimum 94 for DS18B20 in 9 bits resolution
  //  0 means no waiting time (has to be realized by the caller)

function DS1820_TempConversionReady(var Port_: PortType; Bit_: byte): boolean;
  // Returns True if a(all) temperature conversion(s) started with
"DS1820_StartTempConversion" or
  // "DS1820_StartTempConversionROM" is (are) completed, else False (at least one
conversion is ongoing).
  // This routine can be used in stead of a fixed waiting time.
  // IMPORTANT: Can NOT be used with parasite power!

function DS1820_ReadTemperature(var Port_: PortType; Bit_: byte): integer;
  // Reads the temperature out of the DS1820 connected to Port "Port"."Bit".
  // DS1820_CheckCRC can be performed to check temperature reading validity.

function DS1820_ReadTemperature_Fast(var Port_: PortType; Bit_: byte): integer;
  // Reads the temperature out of the DS1820 connected to Port "Port"."Bit".
  // Only the 2 temperature bytes are read from the DS18(B/S)20.
  // DS1820_CheckCRC can NOT be performed to check temperature reading validity.

function DS1820_StartTempConversionROM(var Port_: PortType; Bit_: byte; Wait:
boolean; var RomCode: array[8] of byte): boolean;
  // Starts the temperature conversion of the DS1820 connected to Port
"Port"."Bit", which has ROM code "RomCode"
  // If "Wait" is true, the function waits until the conversion is completed.
  // Returns true if success.

function DS1820_StartTempConversionROM_PP(var Port_: PortType; Bit_: byte; Wait:
integer; var RomCode: array[8] of byte): boolean;
```

```
  // Same as above, but with "parasite" power (power via the dataline during
conversion)
  // Here also the "Wait" time (in millisecs) has to be specified:
  //  minimum 750 for DS18(S)20 or DS18B20 in 12 bits resolution
  //  minimum 375 for DS18B20 in 11 bits resolution
  //  minimum 188 for DS18B20 in 10 bits resolution
  //  minimum 94 for DS18B20 in 9 bits resolution
  //  0 means no waiting time (has to be realized by the caller)

function DS1820_ReadTemperatureROM(var Port_: PortType; Bit_: byte; var RomCode:
array[8] of byte): integer;
  // Reads the temperature out of the DS1820 connected to Port "Port"."Bit", which
has ROM code "RomCode"
  // DS1820_CheckCRC can be performed to check temperature reading validity.

function DS1820_ReadTemperatureROM_Fast(var Port_: PortType; Bit_: byte; var
RomCode: array[8] of byte): integer;
  // Reads the temperature out of the DS1820 connected to Port "Port"."Bit", which
has ROM code "RomCode"
  // Only the 2 temperature bytes are read from the DS18(B/S)20.
  // DS1820_CheckCRC can NOT be performed to check temperature reading validity.

function DS1820_CheckCRC: byte;
  // Returns 0 if the result of the last call of "DS1820_ReadTemperature" or
"DS1820_ReadTemperatureROM" gives a correct CRC.
  // This function does NOT work after the "DS1820_ReadTemperature..._Fast" routine
calls.

procedure DS1820_ReadROM(var Port_: PortType; Bit_: byte; var RomCode: array[8] of
byte);
  // Returns the ROM code of the single DS1820 device connected to Port
"Port"."Bit" in array "RomCode".
  // Caution: only one DS1820 device should be connected to the one wire bus
"Port"."Bit".

function DS1820_Family(var Port_: PortType; Bit_: byte): char;
  // Reads the Romcode of the device connected to (Port_, Bit_) and Returns
  // 'S' for a DS18S20 or DS1820,
  // 'B' for a DS18B20,
  // '?' for an unknown type
  // Can only used with one DS18x20 on the one wire bus

function DS1820_FamilyROM(var RomCode: array[8] of byte): char;
  // Returns 'S' for a DS18S20,
  // 'B' for a DS18B20,
  // '?' for an unknown type
  // of the DS18x20 which has ROM code "RomCode"

function DS1820_CheckCRCRomCode(var RomCode: array[8] of byte): byte;
  // Returns 0 if the romcode in "RomCode" gives a correct CRC

procedure OW_Write_PP(var Port_ : PortType; Pin_, Data_ : byte);
  // Writes one byte of data via the OneWire bus and switches on parasite power
afterwards.
  // Parasite power will be switched off again with any of the "OW.." routine
calls.

function DS1820_ReadPowerSupply(var Port_: PortType; Bit_: byte): byte;
  // Returns the power supply type of the DS1820's connected to "Port_, Bit_":
```

```
  // 0 = at least one Ds1820 uses parasite power
  // 1 = all ds1820's are externally powered

procedure DS1820_SetConfiguration(var Port_: PortType; Bit_, TH, TL: short);
  // Writes the temperature limits TH and TL to the Ds18(S)20 scratchpad
  // TH is the upper temperature limit, TL is the lower temperature limit

procedure DS18B20_SetConfiguration(var Port_: PortType; Bit_, TH, TL: short;
Resolution: byte);
  // Writes the temperature limits TH and TL and the config byte to the Ds18B20
scratchpad
  // TH is the upper temperature limit, TL is the lower temperature limit
  // Resolution is 9,10,11 or 12 <<- translation of the resolution to the config
byte is done by the routine

procedure DS1820_SetConfigurationROM(var Port_: PortType; Bit_, TH, TL: short; var
RomCode: array[8] of byte);
  // writes the temperature limits TH and TL to the Ds18(S)20 scratchpad
  // TH is the upper temperature limit, TL is the lower temperature limit.

procedure DS18B20_SetConfigurationROM(var Port_: PortType; Bit_, TH, TL: short;
Resolution: byte; var RomCode: array[8] of byte);
  // writes the temperature limits TH and TL and the config byte to the Ds18B20
scratchpad
  // TH is the upper temperature limit, TL is the lower temperature limit
  // Resolution is 9,10,11 or 12 <<- translation of the resolution to the config
byte is done by the routine

procedure DS1820_GetConfiguration(var Port_: PortType; Bit_:byte; var TH, TL:
short);
  // Returns the temperature limits TH and TL from the DS18(S)20 scratchpad
  // TH is the upper temperature limit, TL is the lower temperature limit.

procedure DS1820_GetConfigurationROM(var Port_: PortType; Bit_:byte; var TH, TL:
short; var RomCode: array[8] of byte);
  // Returns the temperature limits TH and TL from the Ds18(S)20 scratchpad
  // TH is the upper temperature limit, TL is the lower temperature limit.

procedure DS18B20_GetConfiguration(var Port_: PortType; Bit_:byte; var TH, TL:
short; var Resolution: byte);
  // Returns the temperature limits TH and TL and Resolution from the Ds18(S)20
scratchpad
  // TH is the upper temperature limit, TL is the lower temperature limit
  // Resolution is 9,10,11 or 12 <<- translation of the config byte to the actual
resolution is done by the routine

procedure DS18B20_GetConfigurationROM(var Port_: PortType; Bit_:byte; var TH, TL:
short; var Resolution: byte; var RomCode: array[8] of byte);
  // Returns the temperature limits TH and TL and Resolution from the Ds18(S)20
scratchpad
  // TH is the upper temperature limit, TL is the lower temperature limit
  // Resolution is 9,10,11 or 12 <<- translation of the config byte to the actual
resolution is done by the routine

procedure DS1820_TempToString(Temp: integer; var S: string[5]; Sep: char);
  // Returns temperature "temp", read from an DS1820, as string in S
  // The resolution of the result is 0.5 degree Celsius (1 digit after the decimal
separation character).
  // "Sep" is the decimal separation character
```

```
procedure DS18B20_TempToString(Temp: integer; var S: string[5]; Sep: Char);
  // Returns temperature "temp", read from an DS18B20, as string in S
  // The resolution of the result is still 0.5 degree Celsius (1 digit after the
decimal separation character).
  // The resolution the DS18B20 is put in does not matter.
  // "Sep" is the decimal separation character

procedure DS18B20_TempToString_HR(Temp: integer; var S: string[9]; Sep: Char);
  // Returns temperature "temp", read from an DS18B20, as string in S, with 4
digits after the decimal separation char.
  // "Sep" is the decimal separation character.
  // "Resolution" (values 9, 10, 11 and 12) makes sure the "undefined" temperature
bits are set to zero.

implementation
```

7.2 The OW_Utilities library

This library contains functions to search for 1-wire devices (obtaining ROM codes) and for 1-wire devices in the alarm state.

The signature of the procedures and functions:

```
// interface
{$IFNDEF P24} // for P16 and P18
type TPort = byte;
{$ELSE}
type TPort = word;
{$ENDIF}

var ROM_NO: array[8] of byte;
// the rom code found (if any) will be stored in this array

function OW_Search_First_ROM(var Port_: TPort; Bit_: byte): boolean;
// Search for the first ROM number. Returns true of one is found. In this case the
found number is in "ROM_NO".

function OW_Search_First_Alarm(var Port_: TPort; Bit_: byte): boolean;
// Search for the first ROM number of a device in alarm. Returns true of one is
found. In this case the found number is in "ROM_NO".

function OW_Search_Next(var Port_: TPort; Bit_: byte): boolean;
// Search for the next Rom number or the next Rom number of a device in alarm,
depending on which "search_first" was used before.
// Returns true of one is found. In this case the found number is in "ROM_NO".

implementation
```

7.3 Ds18x20 datasheets

DS18S20.pdf. Also valid for the DS1820.
DS18B20.pdf.

In these datasheets also the one-wire bus behavior is explained as far as the Ds18x20 is concerned.

[end of document]