ARMexpress identifies iButtons®

Dr. Claus Kühnel

"Combining a 60 MHz ARM CPU with a BASIC compiler, our new ARMexpress DIP24 Module is super speedy and easy to program" - this statement from Coridium Corp., the developer of ARMexpress, made me curious and I wanted to play with this new part.

At first sight there seems to be a contradiction between the used ARM7 microcontroller and the DIP24 form factor of the module with its reduced number of I/O pins. The built-in support for I^2C, SPI, RS-232, and 1-Wire protocols is a good base for interfacing peripherals for process control at low pin count too..

This article shows first results from connecting iButtons to ARMexpress for identification.

1. iButtons and 1-Wire devices

1-Wire devices lower system cost and simplify design with an interface protocol that supplies control, signaling, and power over a single-wire connection (and GND). A variety of identification, sensor, control, and memory functions are available in traditional IC packages, ultra-small CSPs, and stainless-steel-clad iButtons.

1-Wire device is a general term and includes iButtons implicit. I use the term iButton always when the iButton is used explicit.

1.1. Some Basics

The 1-Wire Interface connects several 1-Wire Devices to a simple network (MicroLAN).

The interface is built by a drilled two-wire connection (DATA and GND). The pullup resistor is required absolutely. It guarantees the Hi level. A bus master is responsible for controlling the serial bit stream. Figure 1 shows the drivers of bus master and slave in a 1-Wire network.

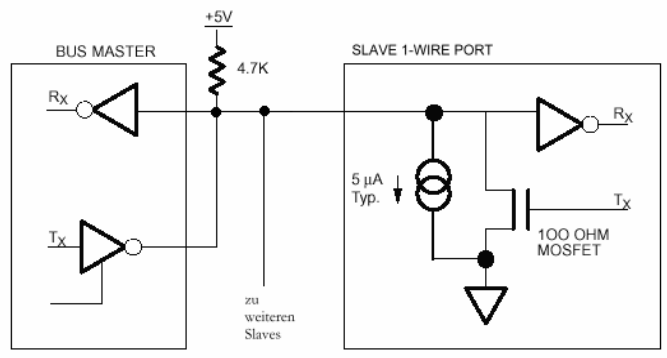

Figure 1 Bus Master and Slave in a 1-Wire network

For an interface built by such simple hardware the software protocoll must secure the communication.

Due to the reduced current consumption of CMOS technology it is possible to power the 1-Wire device in short communication breaks by the Hi level. This in the 1-Wire device saved energy is enough for the running data exchange until the next charge.

The serial data exchange is half-duplex at discrete time slots.

In each case the bus master, it is here the ARMexpress, starts the communication by sending a command to the connected 1-Wire device(s). Commands and data will be sent bit by bit starting with the LSB.

A sharp rise on the data line by the master synchronizes master and slave(s). A certain time after this rise (standard is 30 µs) the data line is polled depending of the transmission direction from master or slave to read a bit information (sample time).

This mode is known as data transmission in time slots. Each time slot is synchronized by a sharp H/L edge of the master. Therefore, pauses in the bit stream do not generates errors or other problems.

A data exchange can start first after connceting a 1-Wire device. The 1-Wire devices are not always soldered. We come back to this point very soon.

Several microseconds after the connection the 1-Wire device pulls the data line to GND to show the bus master the connection and the waiting for a command. This presence pulse can be requested by the master by sending a reset pulse.

Figure 2 explains the timing briefly. The ARMexpress hides the details for the programmer. So it is very easy to use the 1-Wire interface in an application.

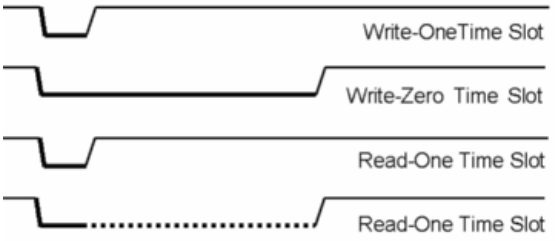

Figure 2 Timing Read/Write Operation

Figure 2 shows the actvities of the master with a streight, thick line. Each write or read operation begins with a falling edge of the master followed by a Lo pulse of about 15 µs.

For writing a „0“ the master hold the data line to GND. For writing a „1“ the master will be passiv and the pullup resistor pulls the data line to Hi.

Reading is quite similar. If the slave sends a „0“ to the master then the slave holds the data line to GND. This phase is marked by a dotted, thick line. If the master should read a „1“ then the slave is passiv and the pullup resistor generates the Hi level.

1.2. 1-Wire Devices

After explaining the basics of the data exchange between the bus master and the slave we want to known somethings about the 1-Wire devices itself.

Dallas MAXIM offers the iButtons and 1-Wire Chips, all equipped with the 1-Wire interface.

The iButtons are devices in stainless steel package called MicroCan. These iButtons look more like a big pill as an integrated circuit. Figure 3 shows such an iButton.

Figure 3 iButton

The MicroCan protects the iButtons as a typical devices used for automatisation purposes against external influences and is used as electrical contact at the same time. Besides numerous possibilities for contacting there is an iButton Mounting Clip for printed circuit boards (Figure 4).

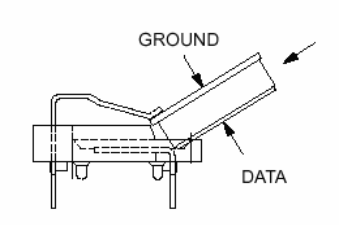

Figure 4 iButton Mounting Clip

The most chips used in iButtons are available as convential integrated circuit in a plastic package.

Each 1-Wire device has an unique 6-byte identification nummer (serial number) saved in a laser-programmed ROM area. The family code is an indication for the device type. The family code and the serial number describe each 1-Wire device unequivocal and identificable.

The ROM area has the same format in all 1-Wire devices. Byte 0 contains the family code describing the device type. The bytes 1 to 6 contain the unique serial number. Byte 7 contains a CRC-8 check sum usable for testing the correctness of the data transmission.

B7	B6	B5	B4	B3	B2	B1	B0
CRC-8	Serial Number						Family Code

Figure 5 shows an overview of the types of 1-Wire devices [1].

Family Code	Part () - iButton Package	Description (Memory size in bits unless specified)
01 (hex)	(DS1990A), (DS1990R), DS2401, DS2411	1-Wire net address (serial number) only
02	(DS1991), DS1425	Multikey iButton, 1152-bit secure memory
04	(DS1994), DS2404	4kb NV RAM memory and clock, timer, alarms
05	DS2405	Single addressable switch
06	(DS1993)	4kb NV RAM memory
08	(DS1992)	1kb NV RAM memory
09	(DS1982), DS2502	1kb EPROM memory
0A	(DS1995)	16kb NV RAM memory
0B	(DS1985), DS2505	16kb EPROM memory
0C	(DS1996)	Up to 64kb NV RAM memory
0F	(DS1986), DS2506	64kb EPROM memory
10	(DS1920), DS1820, DS18S20	Temperature with alarm trips
12	DS2406, DS2407	1kb EPROM memory, 2-channel addressable switch
14	(DS1971), DS2430A	256-bit EEPROM memory and 64-bit OTP register
18	(DS1963S)	4kb NV RAM memory and SHA-1 engine
1A	(DS1963L)	4kb NV RAM memory with write cycle counters
1C	DS28E04-100	4096-bit EEPROM memory, 2-channel addressable switch
1D	DS2423	4kb NV RAM memory with external counters
1F	DS2409	2-channel addressable coupler for sub-netting
20	DS2450	4-channel A/D converter (ADC)
21	(DS1921), (DS1921H), (DS1921Z)	Thermochron® temperature logger
22	DS1822	Econo digital thermometer
23	(DS1973), DS2433	4kb EEPROM memory
24	(DS1904), DS2415	Real-time clock (RTC)
26	DS2438	Temperature, ADC
27	DS2417	RTC with interrupt
28	DS18B20	Adjustable resolution temperature
29	DS2408	8-channel addressable switch
2C	DS2890	Single-channel digital potentiometer
2D	DS2431	1024-bit, 1-Wire EEPROM
30	DS2760	Temperature, current, ADC
33	(DS1961S), DS2432	1k EEPROM memory with SHA-1 engine
37	(DS1977)	Password-protected 32kB (bytes) EEPROM
3A	(DS2413)	Dual-channel addressable switch
41	(DS1922L), (DS1922T), (DS1923), DS2422	High-capacity Thermochron (temperature) and Hygrochron™ (humidity) loggers

*This list may not include all Dallas 1-Wire device types (families), just the ones directly supported by the Automatic Information Business Unit's (BU's) software libraries.

Figure 5 List of 1-Wire devices

1.3. Access to 1-Wire devices

The access to all 1-Wire devices is organized similar to the ISO/OSI-Model. But, not all layers of this modell are implemented. Table 1 shows the existing layers for 1-Wire devices.

ISO/OSI-Modell	*1-Wire device*
Application Layer	No
Presentation Layer	Yes
Session Layer	No
Transportation Layer	Yes
Network Layer	Yes
Link Layer	Yes
Physical Layer	Yes

Table 1 ISO/OSI-Model

The Physical Layer defines the electrical conditions, logic levels, and the timing for all 1-Wire devices.

The basic functions of the 1-Wire communication, as Reset, Presence Detection and Bit transfer, are defined in the Link Layer.

In the Network Layer the identification of the 1-Wire devices using the serial number is carried out. The commands of this layer point to ROM exclusively and are named as ROM Commands therefore (Table 2).

ROM COMMANDS	*FUNCTION*
Read ROM	Reads the complete ROM content (only possible when iButton onnected)
Match ROM	Addresses an iButton according to the 64-Bit ROM content
Skip ROM	Skip addressing (only possible with a connected iButton)
Search ROM	Search for an iButton in a network
Alarm Search	Search for iButtons (DS1920) in a network, which notify an alarm

Table 2 ROM Commands

The Transport Layer is responsible for the data exchange out of the ROM area. The next table shows a selection of the available Memory Commands (Table 3).

MEMORY COMMANDS	*FUNCTION*
Convert Temperature	Starts the temperature measurement
Read Scratch Pad	Reads all bytes from Scratch Pad Memory
Write Scratch Pad	Saves the temperature levels into the Scratch Pad Memory
Copy Scratch Pad	Copies the temperature levels into the EEPROM
Recall EE	Copies the temperature levels back to Scratch Pad Memory
Read Power Supply	Queries the power supply

Table 3 Memory Commands

Only for completeness, here is a hint to the file interface in the Presentation Layer. The iButton operating system uses this file interface. But, for our work with ARMexpress it is not relevant.

The next application sample demonstrates how to organize the access to iButtons.

2. Identification of iButtons

Each iButton can be identified according to its ROM data. Before reading the ROM content the iButtons must be connected reliable to the ARMexpress.

The program sample queries periodically the 1-Wire interface for an iButton connected. Figure 6 shows the output of the program in execution. The source code of the program 1WIRE_ID.BAS is placed on the end of this article.

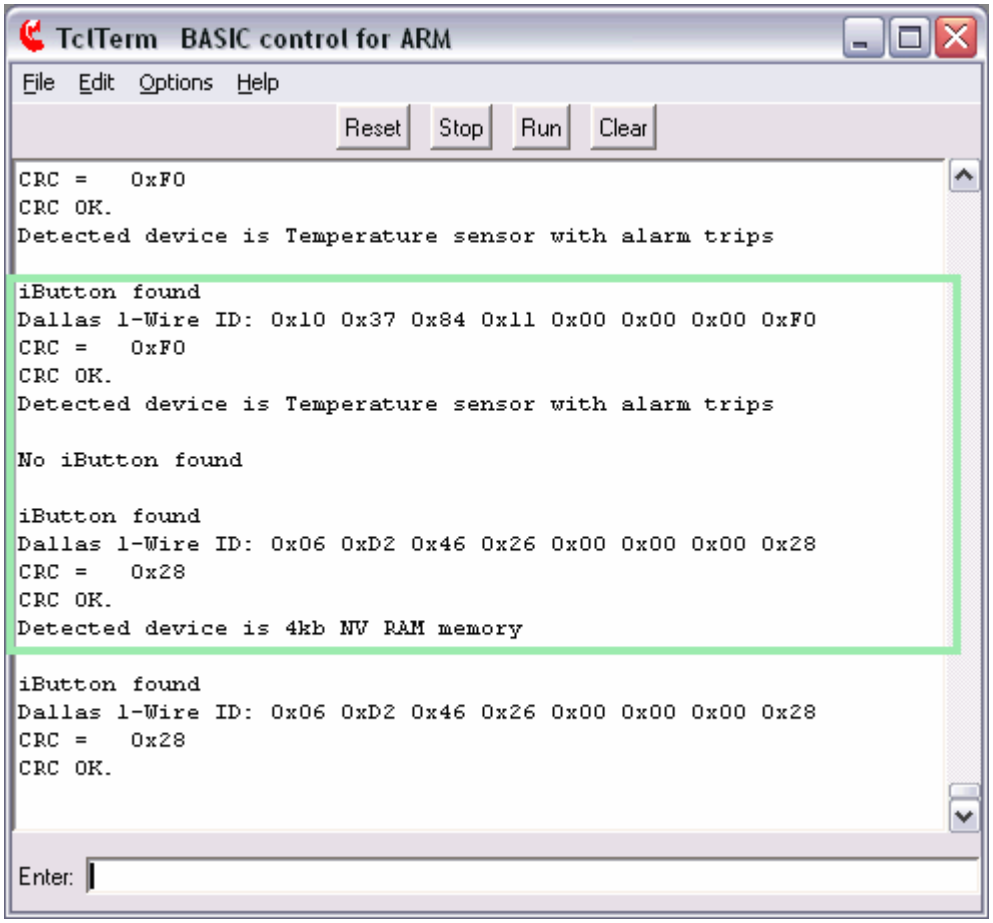

Figure 6 Messages of program 1-WIRE_ID.BAS

If an iButton was found then the program reads its ROM content and sends a hex dump to the terminal. Afterwards a CRC check calculates the CRC-8 over all read bytes except for the CRC byte and sends it to the terminal too. The data exchange was faultless if calculated and read CRCs are equal. A further message displays the result. The last activity is the interpretation of the family code and sending a message describing the detected type of iButton.

Was no iButton detected the message „No iButton found.“ is sent to the terminal.

In Figure 6 an area is marked. Here you can see at first an iButton with family code 0x10 found. The calculated CRC-8 was 0xF0 and signalized a correct data exchange, marked by "CRC OK." additionally. The interpretation of the family code 0x10 delivered a temperature sensor as type of the connected iButton.

Afterwards this iButton was removed and replaced by another. Before the replacement was finished the program detected the missing iButton and sended the message "No iButton found." to the terminal.

After the replacement the program detected an iButton with family code 0x06. The calculated CRC-8 was 0x28 and signalized a correct data exchange, marked by "CRC OK." again. The interpretation of the family code 0x06 delivered a 4 KB NV RAM as type of the connected iButton.

3. Program Development

Now you cold see the operation of the program sample for iButton identification. Coridium has TclTerm as communication and download tool in the package ready for download from the Coridium homepage.

You as programmer can select your preferred editor. It is a matter of taste, but I will show you the Crimson Editor as possibility for editing your source code. The Crimson Editor offers syntax-highlighting etc. and is free.

In the link directory of the ARMexpress User Group I placed a link for download of the Crimson Editor. Furthermore, I placed the required files for syntax-highlighting of ARMexpress Basic in the file directory.

If you like the format of the source code in the next chapter then you can use the template placed in the file directory of the ARMexpress User Group too.

To get a first impression of working with the Crimson Editor Figure 7 shows a part of the source code in the edit window.

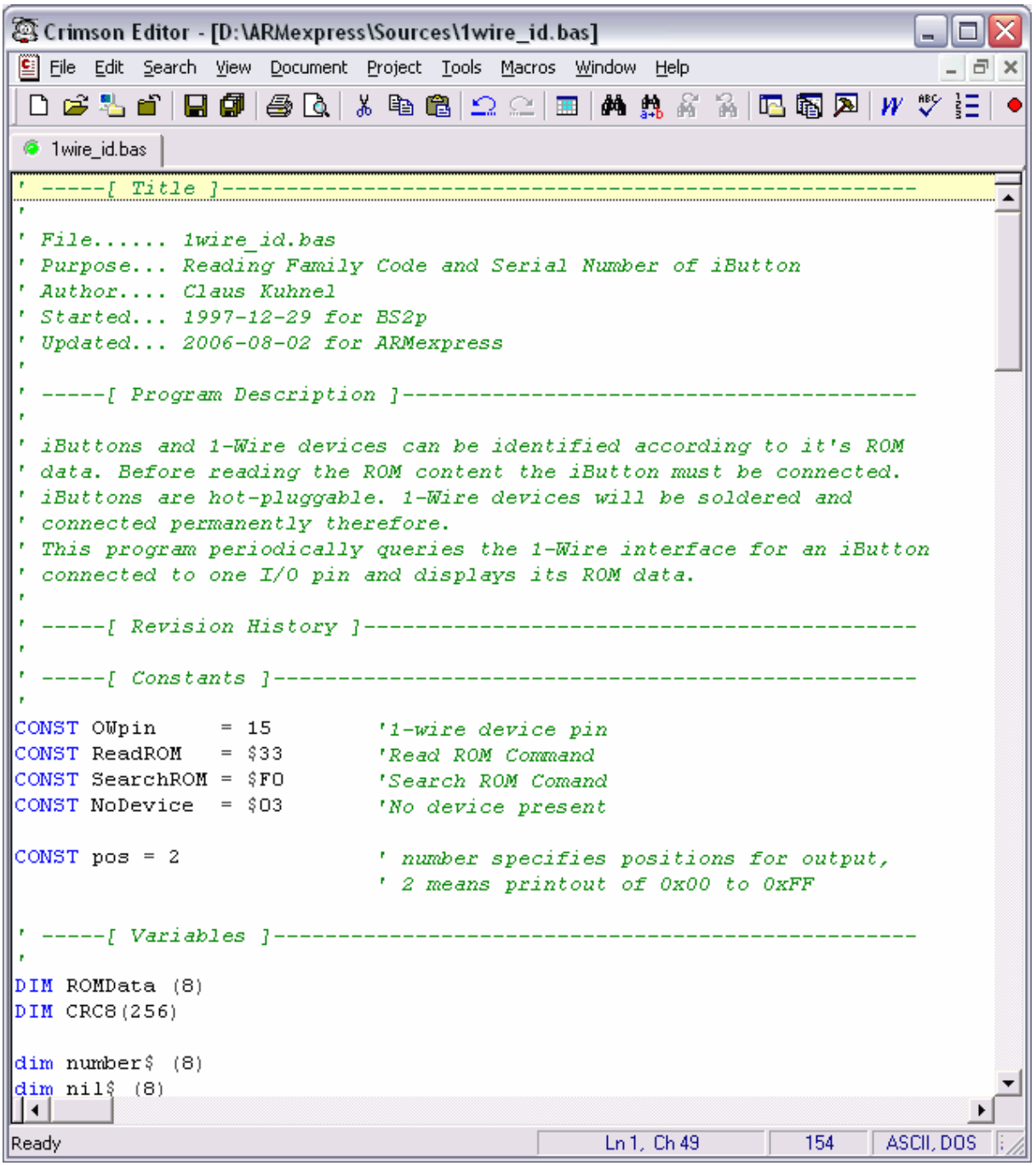

Figure 7 Source Code im Crimson Editor

4. Source code

```
' -----[ Title ]-----------------------------------------------------
'
' File..... 1wire_id.bas
' Purpose... Reading Family Code and Serial Number of iButton
' Author.... Claus Kuhnel
' Started... 1997-12-29 for BS2p
' Updated... 2006-08-06 for ARMexpress
'
' -----[ Program Description ]---------------------------------------
'
' iButtons and 1-Wire devices can be identified according to it's ROM
' data. Before reading the ROM content the iButton must be connected.
' iButtons are hot-pluggable. 1-Wire devices will be soldered and
' connected permanently therefore.
' This program periodically queries the 1-Wire interface for an iButton
' connected to one I/O pin and displays its ROM data.
'
' -----[ Revision History ]------------------------------------------
'
' -----[ Constants ]-------------------------------------------------
'
CONST OWpin         = 15         ' 1-wire device pin
CONST ReadROM       = $33        ' Read ROM Command
CONST SearchROM     = $F0        ' Search ROM Comand
CONST NoDevice      = $03        ' No device present

' -----[ Variables ]-------------------------------------------------
'
DIM ROMData (8)
DIM CRC8(256)

dim value$ (10)
dim number$ (10)

pos = 2                          ' number specifies positions for output,
                                 ' 2 means printout of 0x00 to 0xFF

' -----[ Initialization ]--------------------------------------------
'
restore

for i = 0 to 255                 ' load data into array
    read crc8(i)
next i

' -----[ Main Code ]-------------------------------------------------
'
do
    gosub blink                  ' marks the query by flashing a LED
    present=0
    devcheck = 0
    owout OWpin, present, [SearchROM] ' looks for iButton connected
    if (present=1) then
       print "iButton found"
       gosub displayROM
       gosub displayFamily
    else
       print "No iButton found"
    endif
    print
```

```
    wait(1000)
loop

' -----[ Subroutines ]-----------------------------------------------
'
displayROM:
    print "Dallas 1-Wire ID: ";
    owin  OWpin, ReadROM, [ROMData\8]

    for idx = 0 to 7
       value = ROMData(idx)
       gosub hexform
      print value$; " ";
    next idx

    CRCvalue = 0
    for idx = 0 to 6
      value = ROMData(idx)
      CRCvalue = CRC8(CRCvalue XOR value)
    next idx
    value = CRCvalue
    gosub hexform
    print
    print "CRC = ", value$

    if CRCvalue = ROMData(7) then
      print "CRC OK."
    else
      print "CRC wrong."
    endif
    pause(1000)
    return

displayFamily:
       print "Detected device is ";
       select case ROMData(0)
       case $01
              print "Serialnumber (only)"
       case $06
              print "4kb NV RAM memory"
       case $10
              print "Temperature sensor with alarm trips"
       case $14
              print "256-bit EEPROM memory and 64-bit OTP register"
       case $24
              print "Real-time clock (RTC)"
       case else
              print "Device with Family Code "; hex(ROMData(0))
       endselect
       return

hexform:
    number$ = HEX(value)
    value$ = "0000" + number$
    value$ = RIGHT(value$,pos)
    value$ = "0x"+value$
    return

blink:
    IO(8)=0
    wait(20)
    IO(8)=1
    return
```

```
' -----[ Data ]-----------------------------------------------------
'
' Table lookup method for generating a CRC-8 value on a byte-by-byte
' basis. At the start of each start of each message or datalogging, the
' CRC value must be set to zero.
' After each byte is received or written, the data value and the current
' CRC value is passed to the Crc8 subroutine for processing.
' The Crc8 will return the calculated CRC-8 value to that point.
' The CRC value returned after the last byte is processed is the final
' CRC-8 value.

CRC8_table:
DATA 0, 94, 188, 226, 97, 63, 221, 131, 194, 156, 126, 32, 163, 253, 31, 65
DATA 157, 195, 33, 127, 252, 162, 64, 30, 95, 1, 227, 189, 62, 96, 130, 220
DATA 35, 125, 159, 193, 66, 28, 254, 160, 225, 191, 93, 3, 128, 222, 60, 98
DATA 190, 224, 2, 92, 223, 129, 99, 61, 124, 34, 192, 158, 29, 67, 161, 255
DATA 70, 24, 250, 164, 39, 121, 155, 197, 132, 218, 56, 102, 229, 187, 89, 7
DATA 219, 133, 103, 57, 186, 228, 6, 88, 25, 71, 165, 251, 120, 38, 196, 154
DATA 101, 59, 217, 135, 4, 90, 184, 230, 167, 249, 27, 69, 198, 152, 122, 36
DATA 248, 166, 68, 26, 153, 199, 37, 123, 58, 100, 134, 216, 91, 5, 231, 185
DATA 140, 210, 48, 110, 237, 179, 81, 15, 78, 16, 242, 172, 47, 113, 147, 205
DATA 17, 79, 173, 243, 112, 46, 204, 146, 211, 141, 111, 49, 178, 236, 14, 80
DATA 175, 241, 19, 77, 206, 144, 114, 44, 109, 51, 209, 143, 12, 82, 176, 238
DATA 50, 108, 142, 208, 83, 13, 239, 177, 240, 174, 76, 18, 145, 207, 45, 115
DATA 202, 148, 118, 40, 171, 245, 23, 73, 8, 86, 180, 234, 105, 55, 213, 139
DATA 87, 9, 235, 181, 54, 104, 138, 212, 149, 203, 41, 119, 244, 170, 72, 22
DATA 233, 183, 85, 11, 136, 214, 52, 106, 43, 117, 151, 201, 74, 20, 246, 168
DATA 116, 42, 200, 150, 21, 75, 169, 247, 182, 232, 10, 84, 215, 137, 107, 53
```

Listing 1-Wire Identification (1WIRE_ID.BAS)

The source code listed here can be downloaded from ARMexpress User Group or author's homepage.

5. Links

Coridium Homepage	http://www.coridiumcorp.com
ARMexpress User Group	http://groups.yahoo.com/group/ARMexpress
Author's Homepage	http://www.ckuehnel.ch

6. Literature

[1] DALLAS MAXIM APPLICATION NOTE 155
1-Wire Software Resource Guide Device Description
2005-10-11

[2] Kühnel, C.; Zahnert, K.:
BASIC Stamp 2p
Commands, Features and Projects
Parallax: Rocklin (CA), 2003
ISBN 1-928982-19-0

[3] Kühnel, C.; Zahnert, K.:
BASIC Stamp 2nd Ed.
Newnes: Boston et al, 2000
ISBN 0-7506-7245-5