UM0715
User manual

SPC563MPrimerBasic

Introduction

This user manual describes how to use uBasicPrimer.

The uBasicPrimer is an extended BASIC interpreter executed in the SPC563M Primer.

SPC563M Primer is an USB dongle with a SPC563M device on board. It is designed to support from a PC most common automotive communication interfaces (CAN, LIN, SPI) in a simple tool.

Contents

1 How to install

In order to install the SPC563M Primer Basic you need to follow the procedure described below.

Install the SPC563M drivers:

- Download the SPC563M drivers package
- Unzip the package in a folder
- Enter the folder and run RLinkUSBInstall_quite.bat

Install the SPC563M Primer Basic tool:

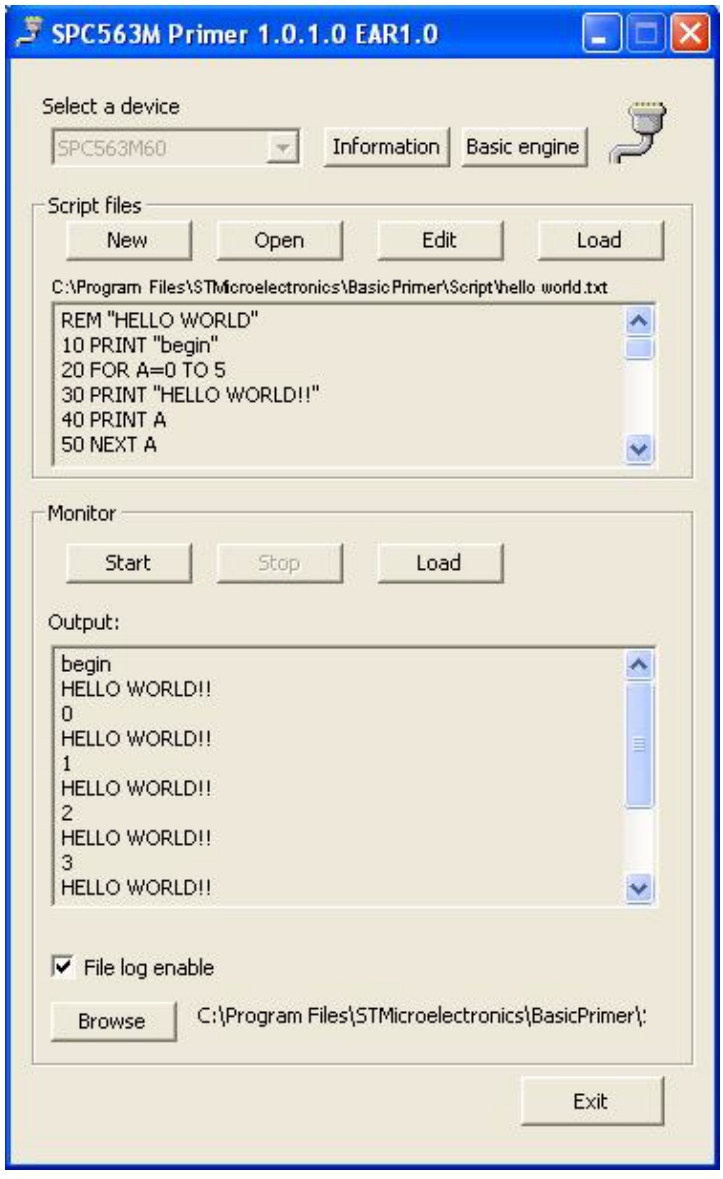
- Download the SPC563M Primer Basic package
- Unzip the package in a folder
- Enter the folder and run setup.exe
- Follow the indications provided by the wizard til the setup completes

2 Main features

2.1 GUI description

Figure 1 shows the uBasicPrimer GUI:

Figure 1. uBasicPrimer GUI



2.1.1 Information

Information button allows showing detailed information regarding the device connected.

2.1.2 Basic engine

Basic engine button inserts the basic interpreter inside the Primer. The basic interpreter will be loaded into the SPC563M flash memory.

2.1.3 Script files

Script File section allows script files management:

- New: it starts notepad to edit a new script
- Open: it allows selecting an existing script. It will be edit only in reading mode on the application GUI
- Edit: It starts the notepad with the script selected by “Open” button to modify the content. After the modification, to show the updated version of the script on the application GUI, the user has to push again “Open” button to reload the script
- Load: It allows flashing the script selected by “Open” button into the SPC563M memory

2.1.4 Monitor

This section allows enabling the output of the SPC563M Primer.

Start button allows starting the execution of the script loaded into the flash of the SPC563M Primer and the output will be show on the output window. Stop button arrests the execution of the script.

To save the output on a log file, select the file name using browser button and check file log enable.

“Quit” button allows closing the application.

3 How to use

Run the tool from the start menu (Start, Programs, STMicroelectronics, Component, Basic Primer).

First step is to load the BASIC engine into the Primer. You can do so by selecting the "Basic Engine" button.

From the GUI now you can:

- Create a new BASIC script or open an existing one by choosing "New" or "Open".
- Edit the BASIC script by choosing "Edit".
- Transfer the BASIC script to the SPC563M Primer by choosing "Load".
- Start/stop the execution of the application by choosing "Start" or "Stop".
- Quit the application by choosing "Quit"

The text window in the "Script files" section shows the contents of the script.

The text window in the "Monitor" section shows the output of the script.

4 Basic command

The section below list detailed commands.

Table 1. Basic commands

Name	Remarks

DIM	Use DIM to create a named list of numbers or strings. Arrays can have up to five dimensions. There must be no space between the name of the dimensioned variable and the opening parenthesis.
	`10 DIM name$(100)`
	creates an array of 100 names.
	`10 DIM map(width, height)`
	creates a 2d array of width * height entries, maybe representing grid squares on a map.
	`20 LET map(1,10) = 2.0`
	sets the top rigth element to 2.0.
	`30 LET map(width, height)`
	is the bottom right element. If you try to access out of range elements the computer will throw an error.
	It's possible to resize an array at any point by calling DIM on it again. If the array is one dimensional, elements will be preserved. If the array has higher dimensions then the elements will be scrambled.
	It's allows to initialize arrays when you dimension them.
	Example
	`10 DIM days$[7] = "Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun"`
	this instruction will declare an array of days of the week. This method is useful for defining data.
	For 2d arrays, the first dimension is the lowest (x) dimension, so:
	`DIM name$(2, 4) = "Fred", "Bloggs",`
	`"Joe", "Sixpack"`
	`"Homer", "Simpson"`
	`"John", "Doe"`
	is the correct order.
	Dimensioned variables are intimately connected with FOR ... NEXT loops. Use the loop counter to index into your array.
	Usage
	`DIM id(numeric, numeric)`
	`10 DIM array(10)`
	creates a single-dimensioned array of 10 numerical elements.
	`10 DIM dictionary$(2, N)`
	creates a 2-dimensional array of N * 2 strings.
	`10 DIM factorial(10) = 1!, 2!, 3!, 4!, 5!, 6!, 7!, 8!, 9!, 10!`
	reates a list of the first ten factorials.

Table 1. Basic commands (continued)

Name	Remarks

LET	The LET statement assigns a value to a variable. If the variable does not exist it is created.
	`10 LET x = 10`
	`10 LET name$ = fname$ + “ “ + sname$`
	Plain variables like x or length are always numerical, string variables like name$ always end with a dollar sign. It is illegal to try to assign a variable of the wrong type.
	LET will not create or increase the size of a dimensioned variable.
	`10 DIM array(2,2)`
	`15 REM Legal`
	`20 LET array(1,2) = 10`
	`25 REM Illegal out of bounds`
	`30 LET array(1,3) = 0`
	The form
	`10 LET x = x + 1`
	is legal. It is even legal is x has not been created (it is initialised to zero).
	Usage
	`LET id = numeric`
	`LET id$ = string`
	`10 LET x = 10`
	`10 LET x = x + 1`
	`10 LET a$ = CHR$(13)`

Table 1. Basic commands (continued)

Name	Remarks

FOR	The FOR statement consists of three parts, the initial set-up value, a TO value, and an optional STEP value.
	`10 DIM array(100)`
	`20 FOR I = 1 TO 100`
	`30 PRINT I`
	`40 NEXT I`
	will print hundred values. The variable in the NEXT statement must be the same as that in the matching FOR.
	FOR loops may be nested to a maximum depth of 32.
	`10 DIM chess(8,8)`
	`20 FOR I = 1 TO 8`
	`30 FOR J = 1 TO 8`
	`40 chess(j,i) = 1.0`
	`50 NEXT J`
	`60 NEXT I`
	The step value does not need to be 1, and may be negative. For instance:
	`10 FOR I = 1 TO 10 STEP 2`
	`20 PRINT I`
	`30 NEXT I`
	`40 FOR I = 10 TO 1 STEP ?0.3`
	`50 PRINT I`
	`60 NEXT I`
	The initial, to, and step values are calculated once on entering the FOR loop, they are then constant.
	`10 LET x = 10`
	`20 FOR I = 1 TO x STEP x/5`
	`30 PRINT I`
	`35 REM Next line has no effect`
	`40 LET x = x + 1`
	`50 NEXT I`
	If the TO value is lower than the initial value (or higher if the STEP value is negative) then the loop does not execute. Control passes to the first matching NEXT.
	It is important not to jump out of FOR ... NEXT loops, or get the nesting order wrong, otherwise control flow will become confused. To terminate a loop prematurely, set the counter to the TO value, and jump to the matching NEXT.
	Usage
	`FOR id = numeric TO numeric STEP numeric`
	`...`
	`NEXT id`

Table 1. Basic commands (continued)

Name	Remarks

GOTO	GOTO executes a jump to another line. The line number is usually a constant, but GOTO x is supported.
	Usage
	`GOTO numeri0`
	
	`10 GOTO 100`
	`10 GOTO x`

IF...THEN	The IF ... THEN construct allows to make decisions.
	If the test condition is true, then control jumps to the line indicated after the THEN keyword. If false, control passes to the next line. No statements other than a line number may appear after the THEN keyword, though the form:
	`IF y < 10 THEN x`
	is supported .
	The relational operators are =, <> (not equal), >, >=, < and <=. They can be applied to strings or to numerical expressions.
	The AND and OR logical operators can also be used.
	Usage
	`IF relational THEN numeric`
	
	`10 IF x < 10 THEN 100`
	`10 IF a$ <> “OK” AND a$ <> “YES” THEN x`

NEXT	For description see FOR.
	Usage
	`NEXT id`
	`10 FOR I = 1 TO 10`
	`20 PRINT I`
	`30 NEXT I`

REM	This statement is used for adding comments to programs.
	It allows multi-line comments, as long as the first character of every continued line is a space.
	Usage
	`REM any comments`
	
	`10 REM Demonstration program by Malcolm McLean`
	
	`10 REM This is an extremely long comment, which is spread over two`
	
	`10 REM PRINT “This PRINT statement is commented out”.`

Table 1. Basic commands (continued)

<table>
<tr><th>Name</th><th>Remarks</th></tr>
<tr><td>STEP</td><td>STEP is evaluated once when the FOR ... NEXT loop is entered It is by default 1, but can be any value, positive or negative. For further details see FOR.
Usage
FOR id = numeric TO numeric STEP numeric

10 FOR i = 1 TO 100 STEP 10
10 FOR i = 100 TO 1 STEP 1
10 FOR i = min TO max STEP delta</td></tr>
<tr><td>GOSUB</td><td>This construct calls a subroutine at the line number <line number>.
Control returns to the statement after the GOSUB command when a RETURN is encountered.
Usage
GOSUB line_number

100 GOSUB 30</td></tr>
<tr><td>RETURN</td><td>Return to the statement after the calling GOSUB command.</td></tr>
<tr><td>SPII</td><td>Init the SPI driver.
Parameter
N_SPI
SPI number (0: SPI_B and 1: SPI_C).
N_BIT
Programmable serial frame size of 4 to 16 bits.
FREQ
SPI peripheral frequency. Setting possible frequency values listed below:
<table>
<tr><th>Value</th><th>Fequency (MHz)</th></tr>
<tr><td>0</td><td>25</td></tr>
<tr><td>1</td><td>16.7</td></tr>
<tr><td>2</td><td>10</td></tr>
<tr><td>3</td><td>7.14</td></tr>
<tr><td>4</td><td>12.5</td></tr>
<tr><td>5</td><td>8.33</td></tr>
<tr><td>6</td><td>5</td></tr>
<tr><td>7</td><td>5.56</td></tr>
</table>
CS_ACTIVE
0: HIGH, 1: LOW
Usage
SPII N_SPI N_BIT FREQ CS_ACTIVE

100 SPII 1 16 6 1</td></tr>
</table>

Table 1. Basic commands (continued)

Name	Remarks

SPIX	Sending a data buffer via SPI and receive the reply.
	Parameter
	EN_PRINT
	(0 disable; 1 Enable) enable the print of the data on the monitor via PConsole application.
	N_SPI
	SPI number (0: SPI_B and 1: SPI_C).
	DATA
	message data containing the data to be send. (the size of the data length must be equal to number of bit N_BIT set on SPI_INIT).
	CS_INIT
	up to 4 Chip Select selectable:0,1,2,3.
	CONT_EN
	if 1 CS remains assert after data sending, 0 CS remain deassert.
	Usage
	`SPIX EN_PRINT DATA N_SPI CS_INIT ACTIVE CONT_EN`
	
	`20 SPII 1 8 5 1`
	`30 SPIX 1 0 0XA5 1 0`

CAN_INIT	Init the CAN driver with a specified baudrate in Kb/s.
	Usage
	`CAN_INIT baudrate`
	
	`10 CAN_INIT 500`

CAN_TX	Send a buffer with a specified identifier via CAN.
	Parameter
	CAN_ID
	CAN message identifier (max 11 bits).
	BUFFER_LENGTH
	message buffer length in byte (max 8 bytes).
	BUFFER
	message buffer containing the data to be send.
	Usage
	`CAN_TX CAN_ID BUFFER_LENGTH BUFFER`
	
	`10 LET id=0x10`
	`20 LET length=0x8`
	`30 DIM msg(8)=1,2,3,4,5,6,7,8`
	`40 CAN_INIT 500`
	`100 CAN_TX id length msg`

Table 1. Basic commands (continued)

Name	Remarks

ON_CAN_RX	When a CAN message is received will be called a subroutine at the line number indicated in the parameter line_number.
	The code returns to the normal statement (after the ON_CAN_RX command) when a RETURN is encountered.
	Parameter
	`ID`
	contain the CAN ID.
	`LENGTH`
	contain the data buffer length.
	`MSG`
	contain the CAN data.
	`LINE_NUMBER`
	contain the line number where jumping .
	Usage
	`ON_CAN_RX ID length msg line_number`
	
	`10 LET id=0`
	`15 LET length=0`
	`20 DIM msg(8)`
	`30 PRINT id`
	`40 FOR I=1 TO length STEP 1`
	`50 PRINT msg(I)`
	`60 RETURN`
	`...`
	`200 ON_CAN_RX id length msg 30`

DELAY	Allow to insert a delay. The parameter indicates the delay in us (Max value is 4294 sec.).
	Usage
	`DELAY time`
	
	`10 DELAY 100`

Table 1. Basic commands (continued)

<table>
<tr><th>Name</th><th>Remarks</th></tr>
<tr><td>GPIOI</td><td>Initializes the GPIO.
Parameter
GPIO_NUMBER
select the GPIO among the following table:
<table>
<tr><th>Value</th><th>Primer Pin</th><th>Functionality</th></tr>
<tr><td>179</td><td>2</td><td>EMIOS 0</td></tr>
<tr><td>181</td><td>4</td><td>EMIOS 2</td></tr>
<tr><td>183</td><td>6</td><td>EMIOS 4</td></tr>
<tr><td>187</td><td>8</td><td>EMIOS 8</td></tr>
<tr><td>104</td><td>1</td><td>GPIO 104</td></tr>
<tr><td>103</td><td>3</td><td>GPIO 103</td></tr>
<tr><td>102</td><td>5</td><td>GPIO 102</td></tr>
<tr><td>105</td><td>7</td><td>GPIO 105</td></tr>
<tr><td>106</td><td>9</td><td>GPIO106</td></tr>
<tr><td>107</td><td>11</td><td>GPIO 107</td></tr>
<tr><td>108</td><td>13</td><td>GPIO 108</td></tr>
<tr><td>190</td><td>LED 1</td><td>-</td></tr>
<tr><td>193</td><td>LED 2</td><td>-</td></tr>
<tr><td>191</td><td>LED 3</td><td>-</td></tr>
</table>
GPIO_I/O
set the GPIO as I/O: 1 input, 2 output
Usage
GPIOI GPIO_NUMBER GPIO_I/O

GPIOI 179 2</td></tr>
<tr><td>GPIOH</td><td>Assert the GPIO indicated into the parameter.
Parameter
GPIO_NUMBER
set the GPIO = 1.
Usage
GPIOH GPIO_NUMBER

GPIOH 179</td></tr>
</table>

Table 1. Basic commands (continued)

Name	Remarks

GPIOL	Deassert the GPIO indicated into the parameter.
	Parameter
	`GPIO_NUMBER`
	set the GPIO = 0.
	Usage
	`GPIOL GPIO_NUMBER`
	
	`GPIOL 179`

PRINT	Print a numeric or string to host via USB.
	Parameter
	`Output statement`
	numeric or string.
	Usage
	`PRINT x`
	`PRINT “hello world”`
	
	`10 LET A=8`
	`100 PRINT A`

SET_TIMER	Set the timer.
	Parameter
	`TIMER`
	timer used (1, 2 or 3).
	`TIME`
	time in us.
	Usage
	`SET_TIMER TIMER TIME`
	`10 SET_TIMER 1 100`

START_TIMER	Start the timer.
	Parameter
	`TIMER`
	timer used (1, 2 or 3).
	`MODE`
	continuos (1) or one shot (0).
	Usage
	`START_TIMER TIMER MODE`
	
	`10 SET_TIMER 1 100`
	`20 START_TIMER 1 0`

Table 1. Basic commands (continued)

Name	Remarks

ON_TIMER	When an event timer is occurred will be called a subroutine at the line number indicated in the parameter line_number.The code returns to the normal statement (after the ON_TIMER command) when a RETURN is encountered.
	Parameter
	`TIMER`
	timer used (1, 2 or 3).
	`LINE_NUMBER`
	line number where jumping.
	Usage
	`ON_TIMER TIMER LINE_NUMBER`
	
	`10 SET_TIMER 1 100`
	`20 START_TIMER 1 0`
	`30 ON_TIMER 1 100`
	`...`
	`100 ...`
	`110 RETURN`
	`...`

5 Expression

5.1 Expressions

Table 2. Expression

Expression	Symbol
+	Sum.
-	Subtraction.
*	Multiplication.
/	Division.
MOD	MOD calculates the modulus of a number. Both sides of the expression should be of the same sign. x MOD 0 is an error. Division by zero is also an error.
!	Factorial.
POW(x,y)	X^Y `POW( x + y, 2)` raise x + y to the power 2.
INT(X)	Force an expression to be the nearest exact integer.
LEN($A)	Return the length of the string A.

5.1.1 Precedence

The + and - operators have lower precedence than *, / and MOD (modulus), which have equal precedence and are evaluated left to right. ! (factorial) has the highest precedence.

Example:

```
(x + y) * 2
```

add x to y and multiply by two.

5.2 String expressions

All strings are stored internally in ASCII format, as NUL terminated arrays. Use of extremely large strings is likely to slow down the program, since most operations involve internal copying of strings.

A string literal consists of one or more concatenated quotes. A string can be spread over several lines, but the newline character is not allowed inside quotes. To enclose a quotation mark in astring, use double quotes.

```
10 LET A$ = ?And God said ??Let there be light?? ?
?and there was light.? ?And God saw the light, that it was good.?
```

is an example of a legal string. Note that the start of the second line contains white space at the beginning to tell the interpreter it is a continuation of the previous line.

To add a newline or other control character, use the CHR$() function. Note that CHR$(0) will prematurely terminate the string. Use the ASCII() function to perform numerical manipulation on characters.

```
e.g LET B$ = CHR$( ASCII(B$) + 1) will set B$ to the next letter of
the alphabet.
```

The ?+? operator will concatenate strings.

```
10 PRINT ?Fred? + ?Bloggs? + CHR$(42) + x$
```

Will print FredBloggs* followed by the contents of x$.

Functions with names ending in ?$? always return strings. Parentheses are not optional.

5.3 Relational expressions

Relational expressions are used only in IF ... THEN statements to make conditional jumps.

A relational expression evaluates to either true or false. The allowed operators a =, <> (doesn't equal), >, >=, <, <=.

With expressions the comparison is numerical, and with strings it is alphabetical. Both sides of a relational operator must be of the same type.

Relational expressions can contain the keywords AND and OR. Order of evaluation is left to right, but parentheses should always be used to disambiguate mixed expressions.

Examples of use:

```
10 IF (x <= 5 AND x > 0) OR x = 10 THEN 100
```

6 Errors

Sometimes PrimerBasic will terminate with an error message. Usually these are due to typing mistakes or logic errors in the basic program. Occasionally they may be caused by the computer running out of resources, by illegal input, or by internal errors in the PrimerBasic interpreter.

Following a list of the errors that can be occurred.

- **Can't read program**

  You have called PrimerBasic with something it cannot recognise as a PrimerBasic program at all, for instance with a text file containing a nursery rhyme.
- **Program lines not in order**

  Lines have to be in numerical order. If lines are out of order, you will receive this error.
- **Line not found**

  You have tried to jump to a non existent line.
- **Syntax error line**

  This means that the interpreter has encountered a line it cannot understand. It is a catch all error, incorporating things such as identifiers starting with digits, or lines not terminated with a newline.
- **Out of memory**

  The SPC563M Primer has run out of memory. This may occur when you try to dimension a huge array, or it may occur at any time if the computer is low on resources, since PrimerBasic uses memory internally. Be particularly careful when dimensioning arrays with variables.
- **Identifier too long**

  An identifier (variable name) is allowed to be only 31 characters long, including the $ for a string identifier. For dimensioned variables the number is one less.
- **No such variable**

  You have attempted to use a variable that has not been initiated.
- **Bad subscript**

  You have tried to access a dimensioned array beyond its dimensioned size.
- **Too many dimensions**

  You have tried to dimension an array with more than five dimensions.
- **Too many initialilers**

  In initialising a dimensioned array, you have tried to list more values than you have space for.
- **Illegal type**

  You have tried to use a string variable as the counter for a for loop.
- **Too many nested fors line**

  Maximum depth of FOR .. NEXT loops is 32. Exceeding this limit is probably due to problems with jumping out of FOR ... NEXT loops.
- **For without matching next**

  You have declared a FOR statement but not a matching NEXT.

- **Next without matching for**

  You have declared a NEXT statement without a matching FOR.
- **Divide by zero**

  You have attempted to divide by zero. This is a mathematical error.
- **Type mismatch**

  You have entered a string expression where PrimerBasic was expecting a numeric expression, or a numeric expression where it was expecting a string.
- **Input too long**

  Input lines can be a maximum of 1023 characters long. Lines longer than this are almost certainly either errors or malicious attempts to exploit the system, so they are rejected.
- **Bad value**

  There has been an internal overflow. Usually this is caused by trying to calculate with ridiculously large value like 10 trillion.
- **ERROR**

  Unspecified error has occurred.

7 Script examples

7.1 Hello world

The following example prints the string “Hello world” continually.

To execute this test, copy and paste the following code in a txt file and load it how already described.

```
10 PRINT "Hello world"
20 GOTO 10
```

7.2 FOR cicle

The following example prints the even values of the variable A.

To execute this test, copy and paste the following code in a txt file and load it how already described.

```
10 FOR A=0 TO 100 STEP 2
20 PRINT A
30 NEXT A
```

7.3 Leds blinking

The following example blinks the leds L1, L2 and L3 sequentially simulating a binary counter. From one configuration and another is inserted a delay of 500ms.

To execute this test, copy and paste the following code in a txt file and load it how already described.

```
10 GPIOI 190 2
15 GPIOI 191 2
20 GPIOI 193 2
30 GPIOH 190
40 GPIOH 193
45 GPIOH 191
48 DELAY 500000
50 GPIOL 190
60 GPIOH 193
70 GPIOH 191
80 DELAY 500000
90 GPIOH 190
100 GPIOL 193
110 GPIOH 191
120 DELAY 500000
130 GPIOL 190
140 GPIOL 193
150 GPIOH 191
```

```
160 DELAY 500000
170 GPIOH 190
180 GPIOH 193
190 GPIOL 191
200 DELAY 500000
210 GPIOL 190
220 GPIOH 193
230 GPIOL 191
240 DELAY 500000
250 GPIOH 190
260 GPIOL 193
270 GPIOL 191
280 DELAY 500000
290 GPIOL 190
300 GPIOL 193
310 GPIOL 191
320 DELAY 500000
370 GOTO 30
```

7.4 CAN monitoring

The following example prints CAN message received.

To execute this test, copy and paste the following code in a txt file and load it how already described.

```
20 CAN_INIT
30 ON_CAN_RX 100
40 GOTO 40
100 CAN_RX_PRINT
120 RETURN
```

8 Revision history

Table 3. Document revision history

Date	Revision	Changes
08-Jun-2009	1	Initial release.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com