Application note on REALIZER

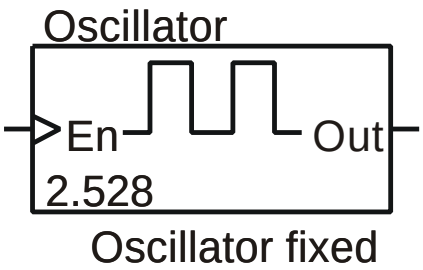
Title: Application note on REALIZER
Subject: Creating your own symbol
Date: 23 januari 2001
Our ref.: 0376901
Author: Mark Bruijn

Introduction

This document describes how to create your own symbol (and your own macro-code) with Realizer.

Most of the time you will use the symbols that are delivered with Realizer. But if you need a symbol for a function that's not included in the libraries of Realizer, you can create your own symbol. Before creating your own symbol, please take into account that writing your own macro-code is not that easy: you have to know about the assembly-language of your microcontroller and the macros you can use with your compiler. So the code that you write depends on your microcontroller and compiler. In other words: if you write code for a certain microcontroller, you can not use this code for other microcontrollers.

The symbol that is described in this document is an oscillator with an enable input: as soon as the enable input becomes true, the oscillator starts running. The moment the enable input becomes low, the oscillator stops running.



Disclaimer

The information in this document is believed to be accurate and reliable. However, Actum Solutions assumes no responsibility for any consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use.

No license is granted by implication or otherwise under any patent or patent rights of Actum Solutions.

Specifications mentioned in this document are subject to change without prior notice. This publication supersedes and replaces all information previously provided.

Actum Solutions products are not authorized for use as critical components in life support devices or systems without express written approval of Actum Solutions.

© 2000 Copyright Actum Solutions, All rights reserved.

Realizer® is a registered trademark of Actum Solutions.

All other trademarks mentioned herein are the property of their respective companies.

Please visit our web site at http://www.actum.com for more information. You can also send us an e-mail at info@actum.com or contact us directly:

Actum Solutions
Industriestraat 9A
1704 AA Heerhugowaard
Netherlands
Tel. +31 (0) 72 576 2555
Fax. +31 (0) 72 576 2559

Oscillator with enable input

The symbol that is described in this document is an oscillator with an enable input. The oscillator with enable can be used to control for example a buzzer. You want to hear this buzzer for 0.5 sec, then 0.5 sec silence, then 0.5 sec buzzing, then silence, etc. When you activate the buzzer, you want to hear it immediately for 0.5 sec, and that's where you need the enable input. If you use a normal oscillator for this with an AND-gate at the output, the oscillator starts running the moment your application initialises. When enabling the AND-gate, there is a chance that just at that moment the oscillator is low for 0.5 sec. After the 0.5 sec, the oscillator will be high and you'll hear the buzzer. So with this solution there can be a delay before hearing the buzzer.

The oscillator with the enable input will start at the moment the enable input is activated. In this way there's no delay: the oscillator starts with a high level at it's output the moment the enable input is activated.

Overview of creating your own symbol

Creating your own symbol (including writing your own macro-code), includes the following steps:

1. Design a new shape for your symbol, including input and output pins and references to the macro-code.
2. Create the file containing the macro-code. Realizer supplies the heading of the macro-functions for you and you must program the definitions.
3. Test your symbol.
4. Put your symbol in a library, so you can use your symbol in other projects.

These steps will be described in more detail in the following paragraphs.

A new shape

You can create a new symbol inside the project you're working on now, or you can start a new project for this. As you want to test your symbol, starting a new project is recommended.

After starting the new project, you select your microcontroller, for example the ST72101. On the empty schematic, you click the right mousebutton and select New-Symbol from there. The window 'select the symbol properties' will appear. Supply the following data (see the end of this document for an explanation):

- symbol name: oscfe
- select a symbol: user defined symbol
- select a shape: rectangle
- number of input pins: 1
- number of output pins: 1

Click button 'Next' to continue to the window 'select the input pin properties' and supply the following data:

- pin1, label: En
- type: Bit, Clock input

Click button 'Next' to continue to 'select the output pin properties' and supply the following data:

- pin2, label: Out
- type: Bit, Output

Click button 'Next' and the window 'select the code properties' appears. Type the following:

- ICODE macro name: ioscfe
- CODE macro name: oscfe
- OCODE macro name: (empty)

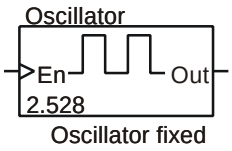
Click button 'Next' and then 'Finish' to let Realizer generate the symbol for you. You're now inside the symbol editor where you can edit your symbol. You have to place the following attributes:

- tag:NAME, value: ?, point of effect: symbol
- tag:TIME, value: ?, point of effect: symbol
- tag:DEVICE, value: TIMER, point of effect: symbol.

After placing the above attributes, you can check the point of effect by selecting one of the attributes. The statusbar then displays some of the properties of the selected attribute, including the point of effect (it should be 'symbol oscfe'. If not, then change the point of effect. The point of effect must be the symbol itself, and not one of the pins!) If you like, you

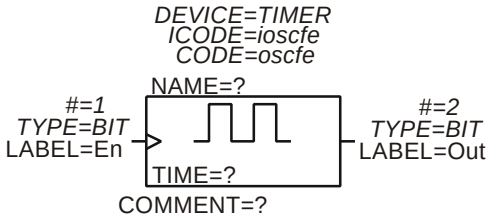
© Actum Solutions 0376901

can change the properties of the attributes, e.g. to make the attribute 'DEVICE=TIMER' invisible or to display only the value of attribute 'NAME=?'. You can enlarge the ghostbox of the symbol and change the properties of the rectangle to make your symbol somewhat larger. You can add some lines to the symbol by clicking the button 'start wiring'. You can add an attribute with 'tag: COMMENT, value: ? and point of effect: symbol'. In the end your symbol should look



like this:

Now you can save your symbol to the local library by clicking the button 'Save the scheme



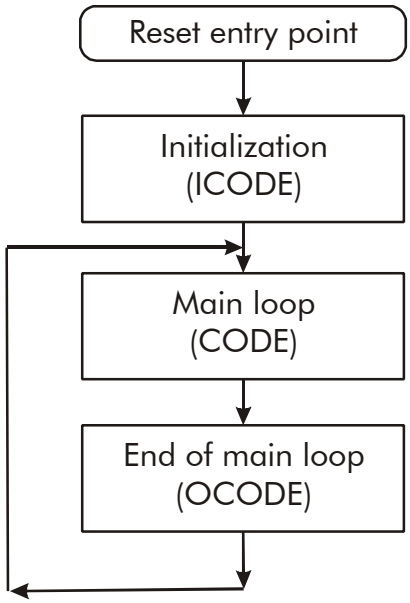
in the active window'.

Now copy the heading of your symbol (in fact, the heading of the macro-code that defines your symbol) to the clipboard by clicking button 'Copy the heading to the clipboard'. After that you can start writing the code for your symbol.

A list of each attribute you can use inside the symbol editor can be found at the end of this document.

The macro-code

When analysing your project, Realizer generates code that has the following structure:



Each symbol can have code (a macro-function) that is used at these different locations. That's why you can write 3 different functions for your symbol and that's why you can specify up to 3 names for these functions. You specify these names when creating the new symbol, in window 'select the code properties'. The name you specify for ICODE is the name of the function that is used at the initialisation of your application. The function with the name that you specify at CODE is used somewhere in the main loop of your application, and the function that you specify at OCODE is used at the end of the main loop (after all other calculations are done).

The heading for the functions is generated by Realizer and is copied to the clipboard. You can start a text-editor and paste the generated heading into the file with macro-code that you are writing. You do this as follows:

© Actum Solutions 0376901

Start an ASCII text-editor and start editing a new file, for example 'oscfe.mac' in your project directory. Paste the contents of the clipboard into your new file. You now have the following code:

```
ioscfebbb macro En,nEn,tEn,pEn,pnEn,ptEn,
    Out,nOut,tOut,Ctime,Ttime,clock,Tclock
  local labnr

  #if {t2}
    ; t2 connected

  #endif

  ; Example: unconditional jump to E&labnr
  jp E&labnr

E&labnr:
  mend

oscfebbb macro En,nEn,tEn,pEn,pnEn,ptEn,
    Out,nOut,tOut,Ctime,Ttime,clock,Tclock
  local labnr

  #if {t2}
    ; t2 connected

  #endif

  ; Example: unconditional jump to E&labnr
  jp E&labnr

E&labnr:
  mend
```

The generated heading depends not only on the name of your symbol (e.g. 'oscfe'), but on the type of your input- and output-pins too. The generated parameters for the oscillator with enable are:

En, **nEn**, **tEn**: the value of the En-pin ('En'), the bitnumber in which the value of En is stored ('nEn', only when the type is BIT) and the type of the En-pin.

pEn, **pnEn**, **ptEn**: the previous value of En (value of En in the previous loop).

Out, **nOut**, **tOut**: the value of the Out-pin, including bitnumber and type.

CTime and **TTime**: the value of the parameter 'Time' and the type of this parameter.

clock and **Tclock**: a variable especially for time-related symbols. By setting and reading this variable, you can use the elapsed time in your macro-function.

A list of the all the types of input- and output-pins of a symbol and the heading that is generated out of it, is included at the end of this document.

The macro-code should be changed into the following:

```
ioscfebbb macro En,nEn,tEn,pEn,pnEn,ptEn,
    Out,nOut,tOut,Ctime,Ttime,clock,Tclock
  local labnr

  #if {tOut}
       ; Out connected
       bitres Out, nOut
  #endif
  mend

oscfebbb macro En,nEn,tEn,pEn,pnEn,ptEn,
    Out,nOut,tOut,Ctime,Ttime,clock,Tclock
  local labnr

  #if {tOut}
       ; Out connected
       bittjt pEn, pnEn, A&labnr
       bittjf En, nEn, E&labnr
       ; pEn=0 and En=1: start oscillator
       bitset clock,7
       jp E&labnr

A&labnr:
       bittjf En, nEn, B&labnr
       ; pEn=1 and En=1: continue oscillator
       bittjf clock,7,E&labnr
       copyww Ctime,Ttime,clock,Tclock
       bitres clock,7
       invbb Out, nOut, tOut, Out, nOut, tOut
       jp E&labnr

B&labnr:
       ; pEn=1 and En=0: stop oscillator
       bitres Out, nOut

  #endif

E&labnr:
  mend
```

The macro-code above uses macros that are defined in the library files of Realizer. For example 'copyww' is not an assembly-command but a call to a macro that copies one variable into another. This macro can be found in the library 'st72lib.inc' in the library directory of Realizer (by default 'C:\Program Files\Realizer Gold\Lib'). Which library files are used depends on the microcontroller you select. For example, when you select the ST72101, the following library files are used:

- st72101.inc
- st72lib.inc
- st72xx.inc

© Actum Solutions 0376901

These files are listed in editbox 'Include files' at the hardware settings (menu-item 'Project-Hardware settings', tab 'General'). Examine these files to learn how to use the macros.

Test your symbol

Now that you've created both the shape (using the symbol editor) and the code (using a text editor) of your symbol, you're ready to test it. If you're still in your text editor then save your text and exit the editor. If you're inside the symbol editor then close the symbol and go back to the schematic editor of Realizer. You now see the root scheme of the new project.

Open the local library (button 'Add a local symbol to the active scheme' in the toolbar) and place the new symbol ('oscfe') on your schematic. Connect an input ('digin') and output ('digout') to the oscillator, connect the input and output with a pin of the microcontroller and your schematic is finished. But now Realizer doesn't know yet where to find your macro-code. You have to add your file in the list with include files (menu-item 'Project-Hardware settings', tab 'General'). After doing this, you can analyse and simulate your project. You've created your own symbol!

Put your symbol in a library

At this point, you can use your symbol in your current project. But how about using the symbol in other projects? Before you can do this, you have to put your symbol in a library. Start a new scheme in your current project (for example 'MyLib.sch') and use copy/paste to copy your new symbol from the schematic you already have to the new schematic. Now the new schematic only contains your new symbol. Now save the new schematic as a library file ('.lib') instead of a scheme, for example ('MyLib.lib'). Please save this library file in the default library directory of Realizer, so you can access it easily from other projects. Make a copy of the file containing your macro-code into the library directory too, copy the file to 'MyLib.mac' for example.

Now you're ready to start a new project and try if you can use your new symbol in it: start a new project (in a new directory), select the same microcontroller that you wrote the macro-code for (ST72101), and open your library (in the project window, double-click libraries and select your own library 'MyLib.lib'). Open the library: your symbol ('oscfe') is in there. Place the symbol. Before you can analyse your project, you have to add the file with your macro-code to the list of include files (menu-item 'Project-Hardware settings', tab 'General'). Now finish your project (add and connect some I/O symbols) and analyse and simulate it: you can use your symbol in other projects too!

© Actum Solutions 0376901

Creating a new symbol

When creating a new symbol with Realizer (right mousebutton, 'New-Symbol'), you have to supply some symbol properties. This paragraphs describes the properties you supply when creating a new symbol.

Properties of the symbol

window 'select the symbol properties':

symbol name (oscfe)

The name of your symbol. It must be unique for each new symbol you create in this project.

select a symbol (user defined symbol)

You can choose between 'subscheme symbol' and 'user defined symbol'. When choosing 'user defined symbol', you can write macro code for your new symbol. When choosing 'subscheme symbol' you create a new shape for a new subscheme symbol.

select a shape (rectangle)

The default shape you start with. You can adjust this shape later.

number of input pins (1)

The number of input pins of your symbol.

number of output pins (1)

The number of output pins of your symbol.

Properties of the inputpins

window 'select the input pin properties':

label (En)

The name of the pin.

type (Bit, Clock input)

First, you specify the data-type that you can use with the pin. This can be:

BIT	0,1
UBYTE	0 .. 255
SBYTE	-128 .. 127
UINT	0 .. 65535
SINT	-32768 .. 32767
LONG	-2147483648 .. 2147483647
WORD	UBYTE..LONG

When setting the data-type of an input pin with the attribute 'TYPE', a wire connected to this pin must be of this same type. When setting the data-type to 'WORD', a wire connected to this pin can be of type 'UBYTE'..'LONG'.

After specifying the data-type, you select 'input' or 'clock input'. A 'clock input' is edge sensitive, an 'input' is not. As the 'enable' input of the oscillator with enable, is edge sensitive, the type of this input is 'clock input'.

Properties of the output pins

window 'select the output pin properties':

label (Out)

The name of the pin.

type (Bit, Output)

The data-type of the output pin. Available are:

BIT	0,1
UBYTE	0 .. 255
SBYTE	-128 .. 127
UINT	0 .. 65535
SINT	-32768 .. 32767
LONG	-2147483648 .. 2147483647
MAX 1,2,3	max. type of pins 1, 2 and 3
MIN 1,2,3	min. type of pins 1, 2 and 3

You can define the data-type of the output pin by using the 'TYPE=BIT' .. 'TYPE=LONG' attributes. You also can define the data-type to be the maximum (or minimum) of the data-types of certain input pins. Then you use for example 'TYPE=MAX 1,2': the data-type of the output pin will be the maximum of the data-types connected to pinnumbers 1 and 2. This option is not available in the window 'select the output pin properties', but you can change the 'TYPE' attribute later.

After specifying the data-type you select 'Output' or 'Passive Output'. When selecting 'Output', the pin is a driving source. In Realizer, you can not connect two pins that both are a driving source (an 'Output'). If you want to be able to connect your output with an output that is a driving source, you select 'Passive Output'.

© Actum Solutions 0376901

The code properties
window 'select the code properties':

ICODE (ioscfe)

The name of the macro that is executed when initializing the symbol. This happens during initialisation of the application.

CODE (oscfe)

The name of the macro that is executed from the main loop of the application.

OCODE (ooscfe)

The name of the macro that is executed from the end of the main loop of the application.

You can leave ICODE, CODE or OCODE empty. This means there is no macro-code for that specific functionality (e.g. initialising). When creating a user defined symbol, you must specify ICODE or CODE or OCODE. The macro-functions that you have to write in your text-file, are the functions that you specify here. The names of the macro-functions are the same as the names you use here, appended with the type of the pins. For example, if you specify the name 'oscfe' and you have 2 pins of type bit, then the macro-function you have to write will have the name 'oscfebb'. You don't have to worry about this too much, as the exact name of your macro-function can be generated automatically by the symbol editor of Realizer.

Additional attributes

So far about the attributes you specify while using the wizard before entering the symbol editor. Once inside the symbol editor, you can change the attributes, you can add attributes and you can delete attributes. A list of the attributes that can be used inside the symbol (the symbol is the point of effect):

Symbol Attributes	Description
ICODE	This attribute defines a macro which is executed once, for initialising the symbols' properties.
CODE	This attribute defines the macro which is executed in the main loop of Realizer.
OCODE	This attribute defines the macro which is executed at the end of the main loop.
VALUExxxx	If the symbol has any attributes starting with 'VALUE', then the values of these attributes are added after the pin parameters. If there are more 'VALUE' attributes defined, an alphabetical order is used.
DEVICE	By adding a 'DEVICE' attribute the macro is extended with additional information. The extensions are added after the pin parameters and value attribute parameters. Available for user defined symbols are: INPUT OUTPUT TIMER COUNTER SHIFT

DEVICE (continued)	When the '**DEVICE=INPUT**' or '**DEVICE=OUTPUT**' attributes are used the macro call is extended with the parameters for the input or output ports. The attribute 'NAME' is required and is used to find the connected hardware port. The value of the attribute 'COMMENT' is used in the report file. When the '**DEVICE=TIMER**' attribute is used, the macro parameter list is extended with a time value, a time variable and a system tick variable. The time value is optional constant value (attribute 'TIME=2:00.00'). The type of the time variable is determined by the constant time value or by the attribute 'TIMEPIN=x', where x is an input pin number. It will result in the following macro call extensions: Time,Ttime,timer,Ttimer : if TIME attribute exists timer,Ttimer : if TIMEPIN attribute exists When the '**DEVICE=COUNTER**' attribute is used, the macro parameter list is extended with an additional variable and his type. This variable is used as an internal variable (the counter value). The type of the internal variable is defined by the 'COUNTPIN=x' attribute, where x = a pin number. When the '**DEVICE=SHIFT**' attribute is used, the macro parameter list is extended with an additional variable and his type. This variable is used as an internal variable (the shift value). The type of the internal variable is defined by the 'SHIFTPIN=x' attribute, where x = a pin number.
TABLE	When the 'TABLE' attribute is used, the macro parameter list is extended with a reference to a ROM table, the number of records in the ROM table and the default value of the table: table, nrOfrecs, defval Depending on the value of the attribute 'TABLETYPE', the value can be INDEX or LOOKUP, an indexed table is generated or a lookup table is generated. The 'TABLE' attribute defines the filename of the table.
ALLOCATEBLOCKIN	This attribute is used in combination with the attributes 'BLOCKSIZEINUNITS' and 'UNITTYPE', and allocates a memorypool (a table having 1 column) in either RAM or EEPROM. The value of the attribute 'ALLOCATEBLOCKIN' defines the type of memory where the memorypool is allocated in, and can be either RAM or EEPROM. Using this attribute results in an additional parameter in the macro parameter list. The list is extended with a parameter 'm1' that represents the starting address of the memory pool.
BLOCKSIZEINUNITS	This attribute is used in combination with the attributes 'ALLOCATEBLOCKIN' and 'UNITTYPE', and allocates a memorypool (a table having 1 column) in either RAM or EEPROM. The value of the attribute 'BLOCKSIZEINUNITS' defines the number of rows in the table. The parameter list of the macro is extended with parameter 's1', representing the number of variables in the memory pool (the table).

UNITTYPE	This attribute is used in combination with the attributes 'ALLOCATEBLOCKIN' and 'BLOCKSIZEINUNITS', and allocates a memorypool (a table having 1 column) in either RAM or EEPROM. The value of the attribute 'UNITTYPE' defines the type of the variables that are stored in the table, and can be 'BIT', 'SBYTE' up to 'LONG'. The parameter list of the macro is extended with parameter 't1', representing the type of the variables in the memory pool (the table).

Inside the symbol editor, Realizer can generate the heading of your macro-code. The heading that is generated, depends on the attributes that you use in your symbol. The following examples lists some of the attributes and the heading that is generated. The parameters in the generated heading are in the following order:

1. pins
2. previous state of pins
3. non-volatile pins
4. dynamic globals
5. devices
6. tables
7. values
8. memorypool

The following examples of symbol definitions describe the headings that are generated:

pins:	pin 1: clock input, type=bit, label=En pin 2: output, type=bit, label=Out
attributes:	CODE=mycode DEVICE=TIMER TIME=?
heading:	mycodebbb macro En,nEn,tEn, pEn,pnEn,ptEn, Out,nOut,tOut, Ctime,Ttime,clock,Tclock

mycodebbb: The name of the macro-function uses the value of attribute CODE, appended with the type of the pins. Pin 1 is of type bit, so a 'b' is appended to the name. Because pin 1 is a clock input, the previous value of pin 1 is included in the heading, and another 'b' is appended to the name. Pin 2 is of type bit also, and the third 'b' is appended to the name.

En,nEn,tEn: The value of the 'En'-pin is in parameter 'En'. As the 'En'-pin is of type bit, the bitnumber in which the value of En is stored, is a parameter too (parameter 'nEn'). The type of the 'En'-pin is the third parameter ('tEn'). This can be one of the following: BIT, UBYTE, SBYTE, UINT, SINT, LONG or WORD.

pEn, pnEn, ptEn: The previous value of the 'En'-pin. 'pEn' is the value, 'pnEn' is the bitnumber and 'ptEn' is the type. By comparing this variable (the previous value) with the actual value, you can create edge-sensitive inputs.

Out, nOut, tOut: The value of the 'Out'-pin (value, bitnumber, type).

Ctime, Ttime: The attribute DEVICE=TIMER (without using the attribute TIMEPIN, but with using the attribute TIME), results in 2 additional parameters: 'time' and 'clock'.

© Actum Solutions 0376901

'time' is a constant value: the value of the TIME attribute. 'Ctime' is the value itself and 'Ttime' is the type of the value.

clock, Tclock: These parameters define the value ('clock') and type ('Tclock') of the system tick variable. By comparing the 'clock' variable with the 'time' variable, you can create timers, oscillators, etc.

pins:	pin 1: input, type=bit, label=In pin 2: output, type=uint, label=Out
attributes:	CODE=mycode DEVICE=COUNTER COUNTPIN=2 VALUE=123
heading:	mycodebw macro In,nIn,tIn, Out,tOut, d3,t3, c4

mycodebw: This time a 'b' and a 'w' are appended to the name of the macro. The 'b' because of pin 1: this pin is of type bit, and the 'w' (word) because of pin 2: the pin is of type uint. For any pin that is not of type bit, a 'w' is appended to the name.

In, nIn, tIn: The value ('In'), bitnumber ('nIn') and type ('tIn') of pin 'In'.

Out, tOut: The value ('Out') and type ('tOut') of pin 'Out'. Because pin 'Out' is not of type bit, the bitnumber is not included in the parameters.

d3, t3: The attributes DEVICE=COUNTER and COUNTPIN=2 append a parameter to the heading that is used as the (internal) counter value. 'd3' is the value and 't3' is the type of the internal counter. The type is determined by the pin that the COUNTPIN references to (pin 2, pin 'Out').

c4: The attribute VALUE appends another parameter to the heading, representing the value of the attribute VALUE (123). The type of this parameter is not included, because the type is always the same (it is a constant).

pins:	pin 1: output, type=long, label=Out
attributes:	CODE=mycode DEVICE=INPUT NAME=?
heading:	mycodebw macro Out,tOut, p2,t2

mycodew: The name, appended with the type of pin 1 (type=long, a 'w' is appended).

Out, tOut: The value ('Out') and type ('tOut') of pin 'Out'.

p2, t2: The name ('p2') and type ('t2') of the connected register.

© Actum Solutions 0376901

pins:	pin 1: output, type=ubyte, label=Out
attributes:	CODE=mycode TABLE=mytable.tab TABLETYPE=INDEX
heading:	mycodebw macro Out,tOut, c2,c3,c4

mycodew: The name of the macro is appended with a 'w' (type of pin 1).

Out, tOut: The value and type of pin 'Out'.

c2, c3, c4: Using attribute TABLE results in these additional parameters. These parameters references to the table that Realizer reads from the file into ROM. c2 is the reference to the start of the table. c3 is the number of records in the table. c4 is the default value of the table.