Introduction to ScicosLab\Scicos

Lombardi Lorenzo

September 1, 2011

Index

Premise

The operating system on which were carried out the examples in the document is Windows XP SP3 on which were installed versions 4.4b8 of ScicosLab and version 1.5.1 of Erika Enterprise, code-named "Giacomo", with the ScicosLab Pack 9.3 . Different OSes and / or later versions of the software may bring to significantly differ result with what follows, however I hope that the tutorial will be useful the same.

1.1 ScicosLab

ScicosLab is an Open Source software package whose development is carried out by the METALAU[1] group enclosing some researchers belonging to INRIA[2] and ENPC[3] based on version 4 of Scilab and containing within it an important toolbox called Scicos.
Scilab is a software for numerical computation that includes, within libraries, hundreds of useful features for different uses: from the mathematical calculation to the simulation of control systems. It also offers the opportunity to design the personal functions in C, C ++ or *Scilab*.
Scicos is a toolbox with a graphical interface that allows modeling of the block diagrams that can simulate dynamic systems of various nature, also for this toolbox is possible to design blocks adapted to the specific needs in C or *Scilab* language.

1.2 Introduction to ScicosLab/Scicos

Generally it is advisable to create a new folder that encompasses all of the files in the project interested by the actual work: once created this will be set as the working directory of ScicosLab for the current session (you can still change the workbook without having to restart the program)

Open ScicosLab. To place the working directory to the desired folder, type the command

cd("Path folder");

The quotes that enclose the path of the folder used to avoid problems related to the presence of any spaces in the path
(es.: C:\Documents and Settings\folder)

To start Scicos, in the top bar, under "Application" menu, select "Scicos" (or run the command 'Scicos ();' -Without quotes! - In ScicosLab)

This will open a new window. In this window, in the top menu, go on the "Palette" menu and select "pal tree": a third window will appear with various expandable folders containing all blocks ("palette") usable in Scicos. Open the
menu "sources" and position itself on the block "GEN_SQR"

Fig. 1.1: GEN_SQR. block

1 METhodes, Algorithmes et Logiciels pour l' Automatique, http://www-rocq.inria.fr/metalau/
2 Istitut National de Recherche en Informatique et en Automatique, http://www.inria.fr/
3 Ecole Nationale des Ponts es des Chaussèes, http://www.enpc.fr/english/int_index.htm

(scrolling the mouse over the various blocks present in the menu the name of the block are displayed)

Drag the "GEN_SQR" block into the window of Scicos via a single click of the left mouse button on it, hold it until it is positioned in the desired position.

You can move a block already present on the plan with the same system, or by right-clicking the mouse on the block and select "Move".
Go to the menu "Sinks" and place the block "cscope" with the same system described for the block "GEN_SQR".

Place the mouse on the output arrow of the block GEN_SQR, click with the left mouse button and hold it down, draw a line from this block the entrance to the block Cscope

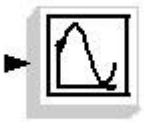

Fig. 1.2: CSCOPE block.

Place the mouse on the output arrow of the block GEN_SQR, click with the left mouse button and hold it down, draw a line from this block the entrance to the block Cscope

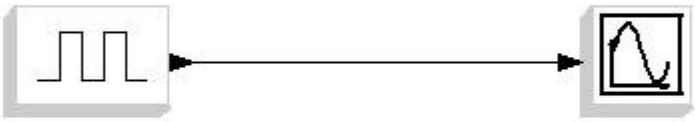

Fig. 1.3: Simple example of connection between blocks.

Now, on the top bar of the Scicos window, go to "Simulate" menu and select "Run". This will open a fourth window named "ScicosLab Graphic" and what will not be exactly the square wave that we expected:

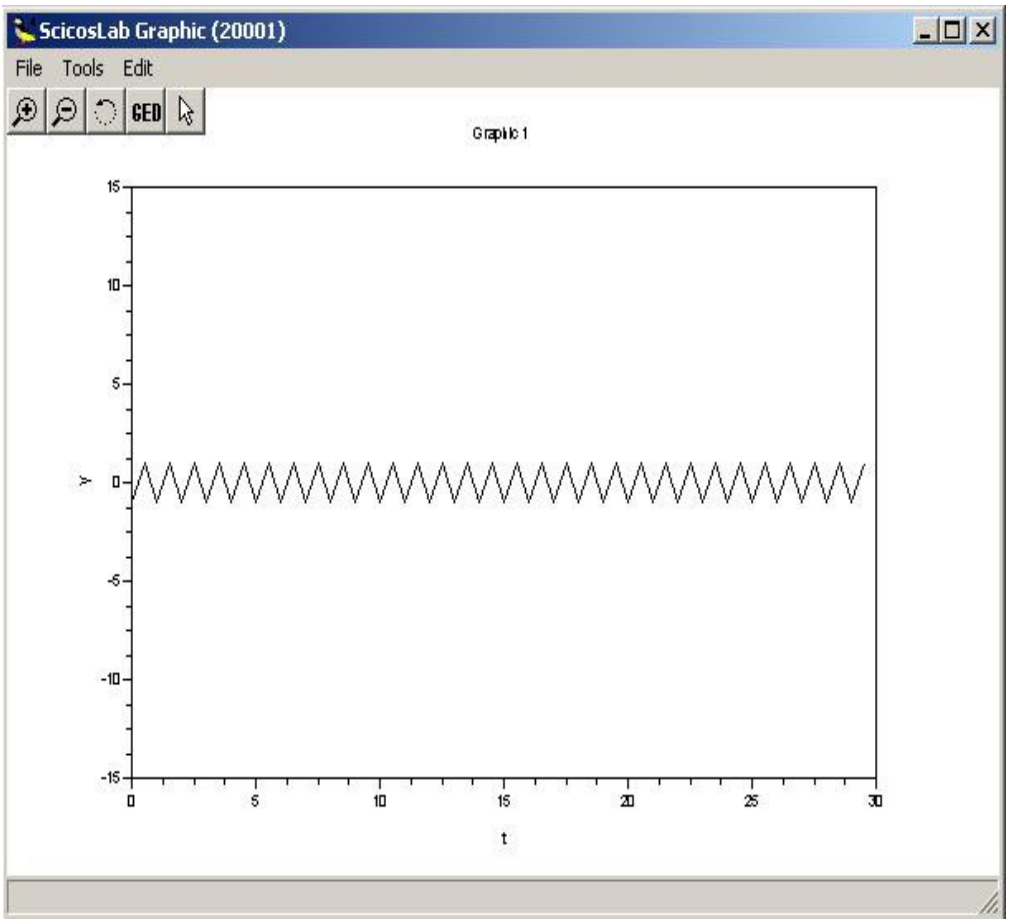

Fig. 1.4: Graph of the block C SCOPE resulting from the diagram of Fig. 3.

The sawtooth displayed an "edge effect" arising from the manner in which the scope combines two samples by default. The mechanism is 'a little' complicated to explain and is not one of the objectives of this introduction, but rather to see how to solve this problem.

Close the graph window and, returning on Scicos, drag the block diagram "CLOCK_c" from the "Sources" menu:

Fig. 1.5: CLOCK_c block.

Then go on the block C and SCOPE, double clicking with the left mouse button on it (Or by a single right click and select the "Open / Set") to change in the setup window of the block that will open item "*Accept herited events*" from one to zero as shown in Fig. 1.6.

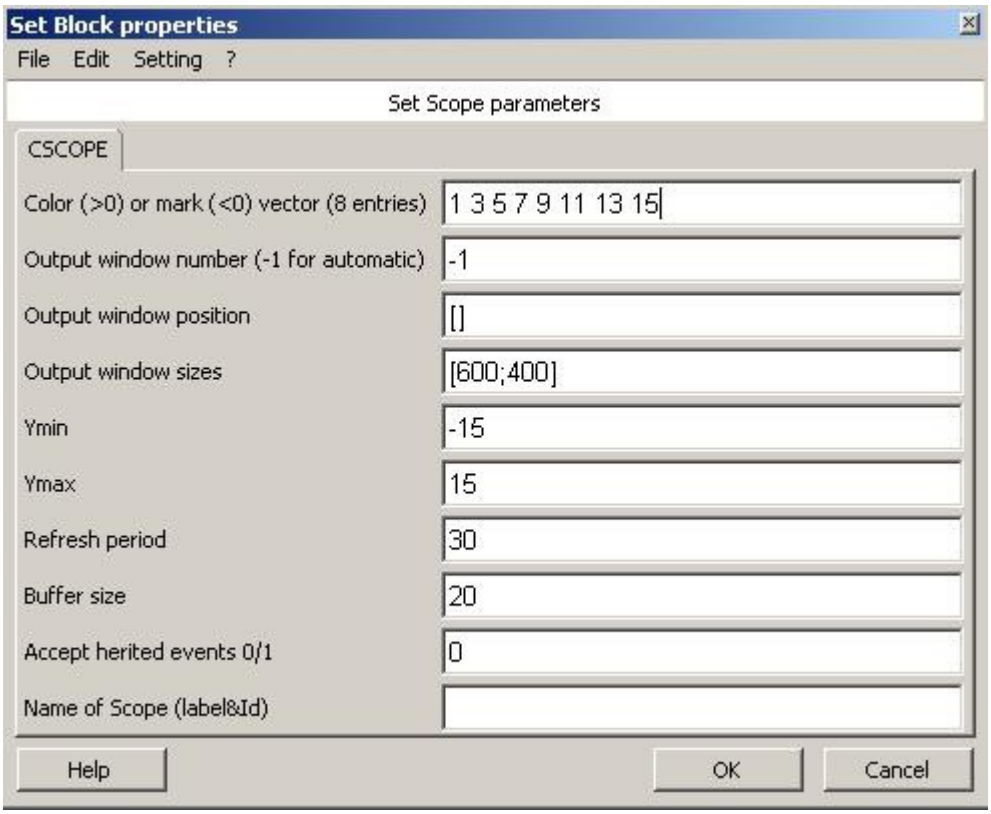

Fig. 1.6: Setting window of block CSCOPE.

In this way on the top of the block CSCOPE appear a small red arrow: Move the mouse on the output arrow of the block CLOCK_c and drag a line (red) from here to the new red arrow input appeared above the block CSCOPE:

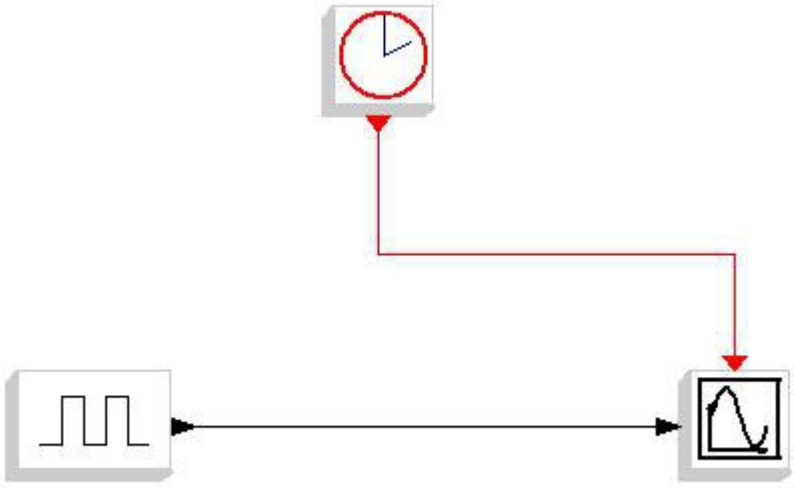

Fig. 1.7: Diagram comprising the synchronization block CLOCK c.

To create segmented line, while you are tracing make a single click with the mouse at the point where you want to change direction.

At this point, executing the command "Simulate -> run" again, the graph will show the square wave desired:

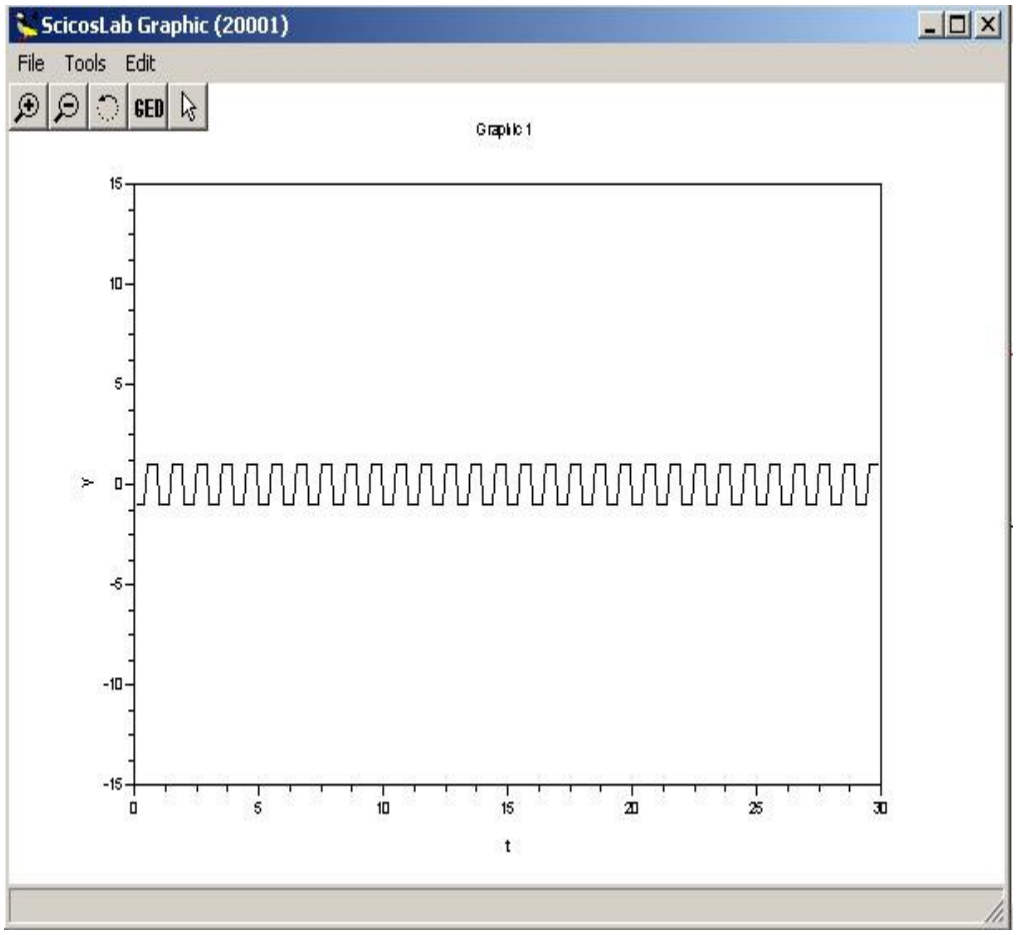

Fig. 1.8: Graph of the block C SCOPE resulting from the diagram of Fig. 1.7.

The block CLOCK_c force blocks connected to it to work at the frequency set by him: double clicking with the left mouse button on this block will open the window for setting the options block CLOCK_c: the best way to understand how it works is to do various tests and see what happens.

Try changing the "Period" by varying it between 0.001 and 1, remembering that a greater frequency of calculation does *Not always* mean greater accuracy of simulation, but rather if it rises too high you risk overloading the system with unnecessary calculations that in the best case do not bring any benefit to the process of calculation and simulation.

Try also to replace the block "GEN_SQR" with other blocks in the "Soucres" menu, for example with the block "GENSIN_f" (to clear the block "GEN_SQR" highlight it by clicking once with the right mouse button and select ' Delete ":the cancellation of a block also results in the disappearance of all the connections that this has with the other blocks in the chart)

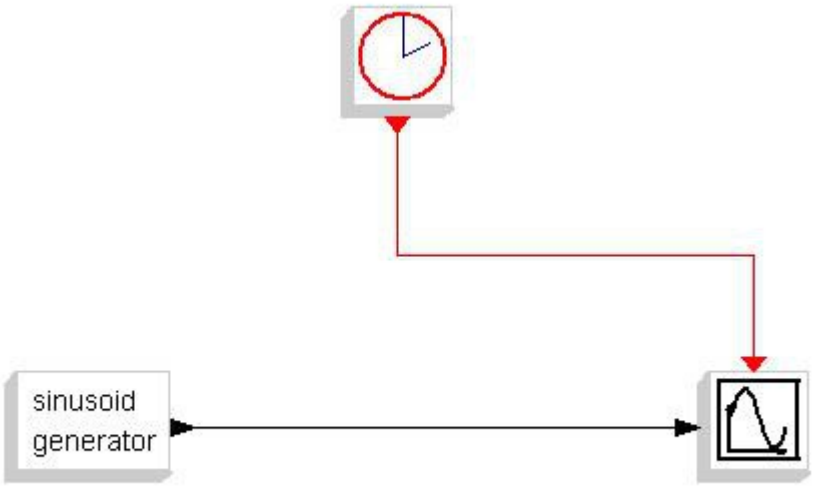

Fig. 1.9: Diagram of Fig. 1.7 with sine wave generator.

Like the block GEN_SQR (and many others), the block GENSIN_f allows you to set the desired properties (specifically amplitude, frequency and phase) opening the window of setting the block properties by double click of the left mouse button on the block under consideration.

To set the duration of the simulation, in Scicos, on the menu "Simulate -> setup" and change the "Final integration time".

With the "Realtiming scaling" option you can select the speed of the simulation: 0 = maximum speed permitted by the system in use, 0.5 = graph evolves speed *Doubled* in respect to how would the real system simulated, 1 = simulation evolves in real time , 2 = slowed by a factor of 0.5 and so on.

While you are in a simulation, you can stop it using the "Stop" command in the menu bar at the top of the Scicos window, you can then restart the simulation from the stop point, restart from the origin or end it by the "Simulate -> run" command.

To change the size of the axes of the graph, open the settings window of the block Cscope and edit the fields "Ymin", "Ymax" and "Refresh period" that indicates the length of the X axis of the graph: be careful that this does not exceed the "Final integration time" set in the parameters of the simulation!

To change the indications on the axes instead, for example to be able to use the chart created in a presentation or similar, click the menu "Edit -> Figures properties" on the ScicosLab Graphic window ,and, in the window of options that will appear, change the "Label" field present in the "Label Options" menu of the various tabs on the right under "Object Properties" menu that appears when you select the "Axes" menu on the left. In this way is possible to modify the instructions given on the chart axes, plus the title of the graph itself contained in the field "Name of Scope (label & Id)." (see Fig. 1.6) This will be useful when we will have schemes with many graphics.

To save a chart, the window ScicosLab Graphic, run the command "Save" under the "File" menu in the top left, in this way (remembering to add the extension ".scg" at the end of the name because this is not done

automatically by ScicosLab) you can save the chart in format ScicosLab Graphic so you can re-open later for further analysis (eg using the command "Zoom", the two magnifiers present under the file menu, which allow zoom in on a specific section of the graph shown within a rectangle that is created by holding down the left mouse button and dragging within the limits of the graph).

To load a saved chart, once opened a window ScicosLab Graphic, run the command "Edit -> Erase figures" and then "File -> Load" loading the desired graph. Using the "File -> Export" command you can save graphics also with other extensions (GIF, BMP, etc.)

The variables in use can be explicitly inserted between the blocks's parameters of the scheme or be indicated indirectly through CONTEXT of the schema.

As an example consider the following scheme, composed of the blocks already seen GENSIN_f, GEN_SQR, CLOCK_c and CSCOPE, and by the block PRODUCT in the menu "Non_linear":

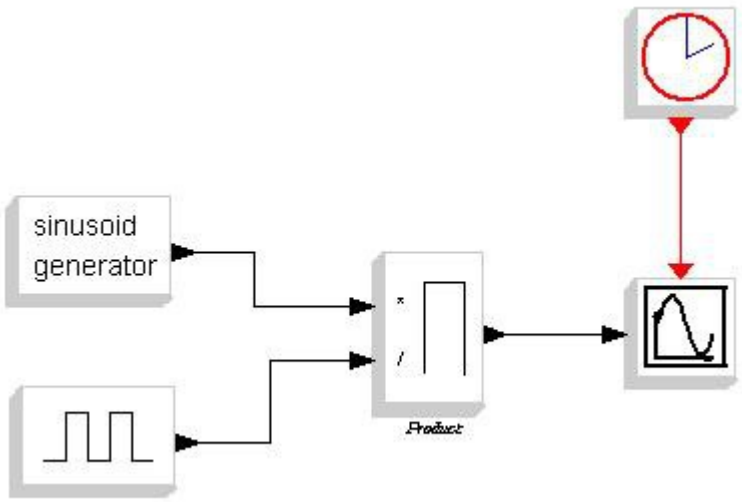

Fig. 1.10: Diagram with magnitudes indicated by context..

The result of the simulation, after having properly set the calculation time of the block CLOCK_c, is shown in Fig. 1.11.

Should be noted that to start a simulation is necessary that all the inputs of the blocks included in the diagram are connected to avoid the uncertainty of their value, while the same is not binding as regards the outputs..

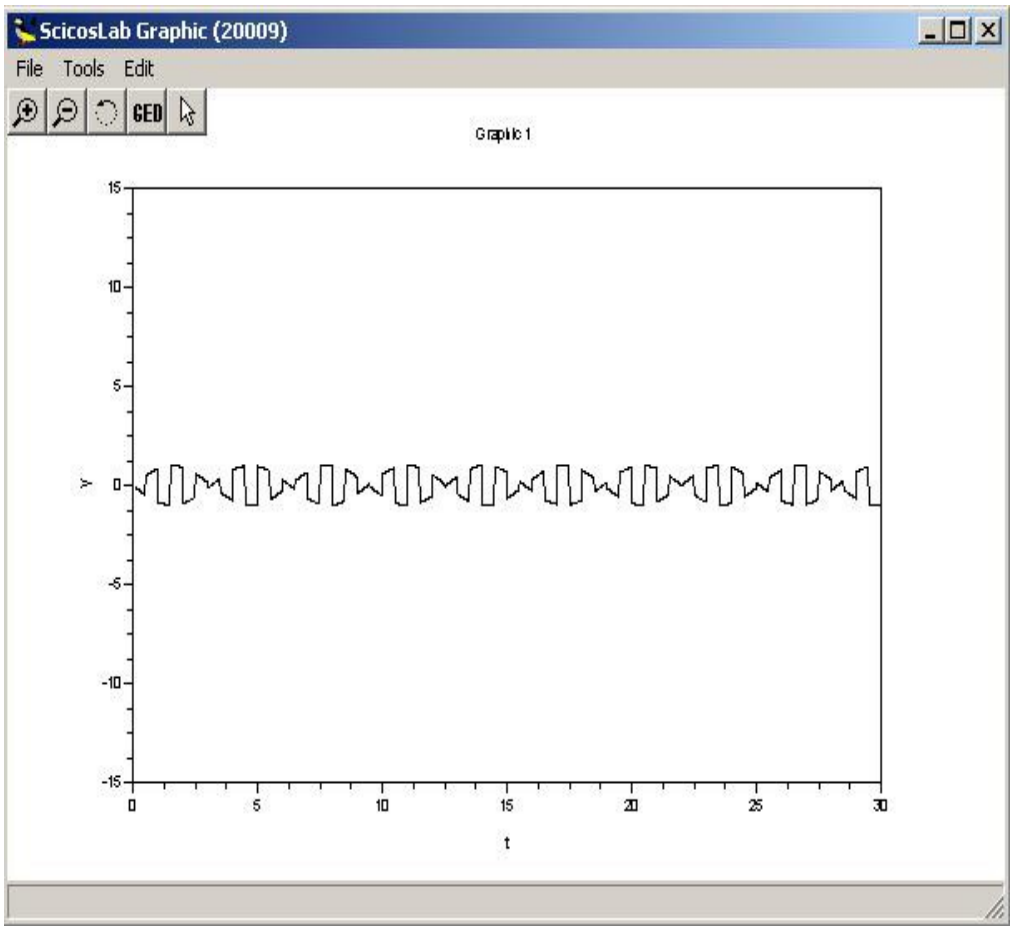

Fig. 1.11: Graph of the Block C SCOPE resulting from the diagram of Fig. 1:10.

Let's go back now to the Scicos graphic window and go to the menu "Diagram -> Context" accessible from the menu bar at the top of the Scicos window: will open a text window in which to enter the parameters of interest.

In the present case we insert the following:

```
a  = 2
f1 = 2
b  = 3
f2 = 4
```

Tab. 1.1: Context of the diagram of Fig. 1:10.

Click "OK" at the bottom and go back on the Scicos Graphic: open the setting window of the block GENSIN_f with a double click of the left mouse button and insert in the "Magnitude" field the value "a" defined in the context and in the field "Frequency (Rad / s) ", the value" f1 "as seen in Fig. 1:12.

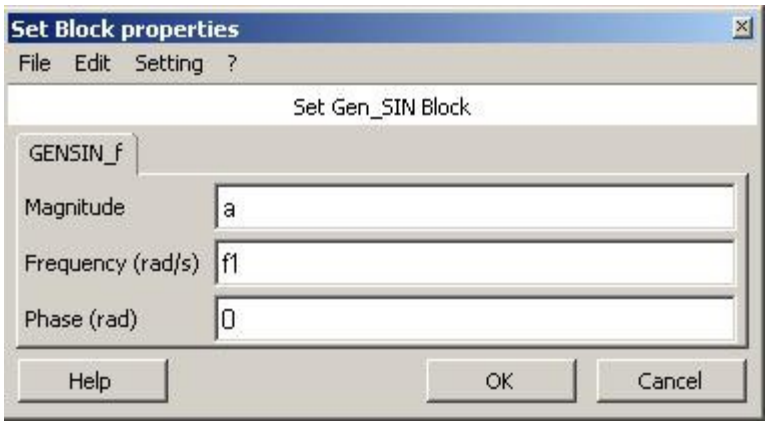

Fig. 1.12: Setting window of block GENSIN_f.

In this way the amplitude of the sine wave is set to the value indicated by the variable "a" in the context and the frequency at the value f1. Then open the settings window of the block GEN_SQR and modified values "Minimum value" and "Maximum value" respectively "-b" and "+ b", then set the field "Period (sec)" to the value "1 / f2 ".

Try again simulation to see the effects of the changes made to the chart.

The use of the context in this way may seem superfluous, but when you have complex patterns, composed of many blocks nested one inside each other in which the same variable may be repeated within several blocks, or in which the same mathematical expression is repeated in more than one parameter, the context allow to change this variable or this expression just one time without having to go and search for each block where this is used at the risk of losing some and allowing you to change the scheme faster and speed up the process tuning of the simulation itself.

But what does it mean to have "more blocks nested inside each other"? When you have Very large patterns, it is often possible to divide them into functional sections that perform specific purposes, in this case to simplify the understanding of the whole schema is useful to enclose these sections in special blocks, called "superblocks", so that the overall schema is composed of functional superblocks connected to one another.

To create a superblock, click with the left mouse button on the Scicos diagram in a blank space and, without releasing it, draw a rectangle that includes all the blocks you want to be included in the superblock. For example, going back to the first scheme, highlight All blocks excluding the block CLOCK_c (you can also enter this one in the superblock, but for our purposes is better to not include it) in order to have the situation shown in Fig. 1:13.

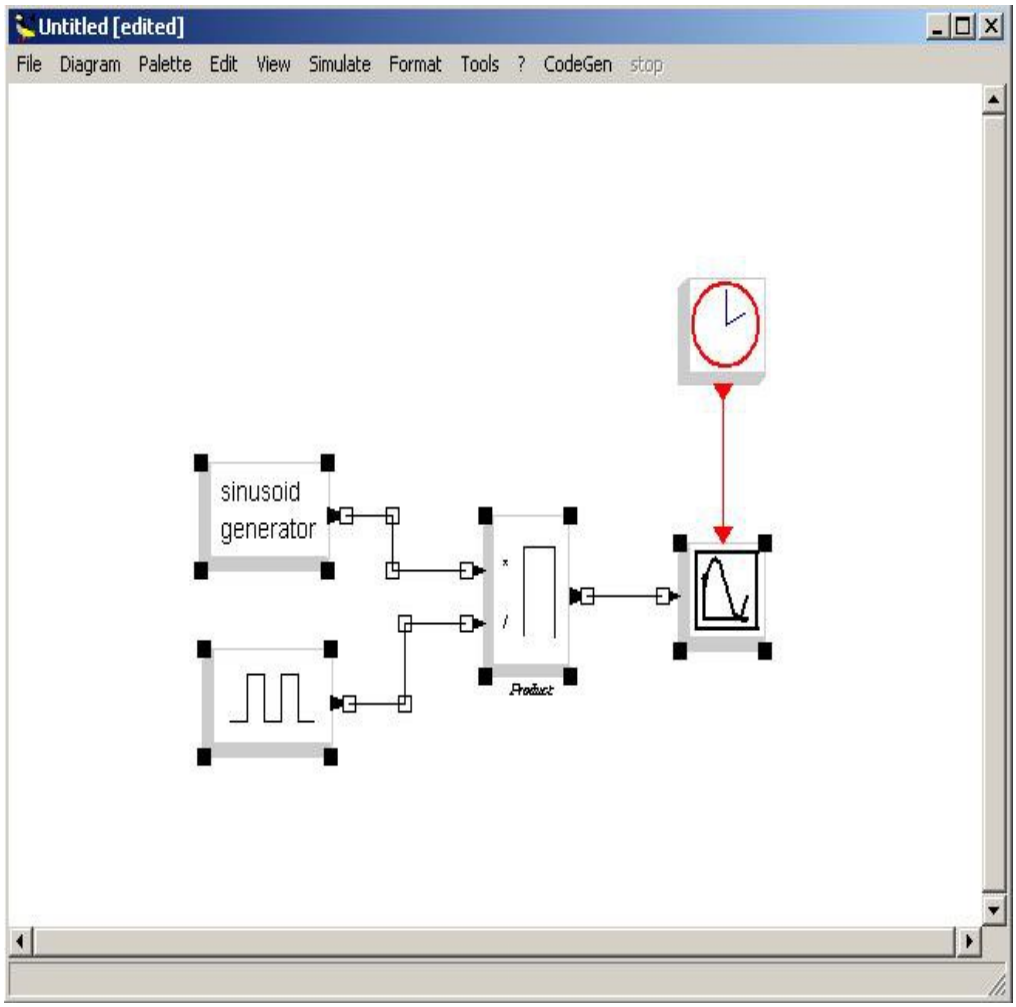

Fig. 1.13: Highlight of the blocks of the diagram of Fig. 1:10 to be included in the superblock.

Then click with the right mouse button and select the "Region to superblock". We will have a scheme similar to that shown in Fig. 1.14.

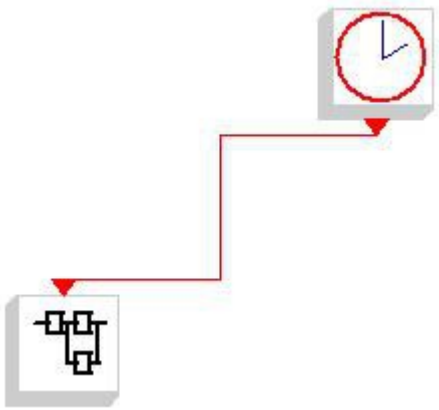

Fig. 1.14: Diagram of Fig. 1:10 enclosed in a superblock.

To obtain a more orderly pattern you can delete the red link that connects the block CLOCK_c to the superblock created (Go on it with the mouse and click the right button and then select the "Delete") to move the

block nearest and reconnect the two in a less segmented way, without losing any functionality. In fact, by re-executing the command "Simulate -> run" the graph obtained will be identical to that obtained previously.

Double-clicking with the left mouse button on the superblock, or by right clicking on it and selecting the "Open / set", you can open a second scheme Scicos comprising the blocks contained in the superblock.

The irregular red pentagon with in the number one represents the point in which, within the superblock, enters the external control signal coming from block CLOCK_c.

If the superblock had more inputs, including of non-activation type , we would have had other pentagons (Blacks, for signals) while if the superblock had had some outputs we would have had some similar pentagonal blocks indicating the outputs from the superblock.

You can still add new inputs and outputs even when the superblock has already been created. To add an input to the superblock, add the block "IN_f" available in the "Sources" menu

Fig. 1.15: IN_f block.

For not changing too much our example, open the setting window of the block PRODUCT and change the field "Number of input or sign vector" from "[1, -1]" to "[1; -1; 1]" : this adds an input to the multiplier block.

The operation that is performed on the inputs is a multiplication if the input is marked with a "1" and a division if "-1". The order in which the inputs appear outside the block are, from top to bottom, in the same order in which they are listed in the input vector from left to right.

In this way the previous connections are unaffected, but due to the different arrangement of the inputs on the side of the block PRODUCT the connection lines may no longer be straight.

To change the path of a connection make a single click over it with the left mouse button: in the points where the line changes direction appear some white rectangles; select them with the mouse pressing the left button and, without releasing it, is possible to move the selected angle to the desired position.

Place the mouse on the tip of the block IN_f and draw a line from this to the new entrance to the block PRODUCT. Add to the diagram the block "OUT_f" available in the "Sinks" menu

Fig. 1.16: OUT_f block.

Place the mouse in the middle of the line joining the block PRODUCT to the block Cscope and, by double clicking with the left mouse button (or, alternatively, by pressing the "l" ("Link") on the keyboard) to connect this line

to the OUT_f block. The end result will be similar to that shown in Fig. 1.17.

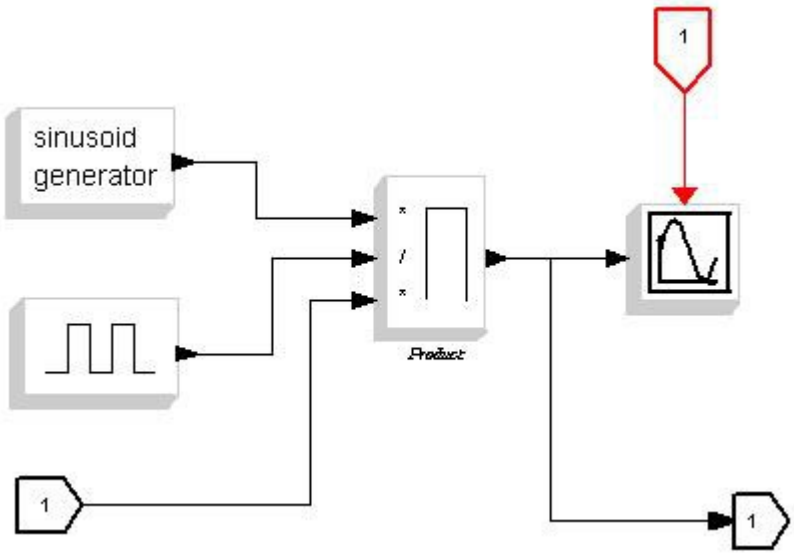

Fig. 1.17: Inner view of the superblock of Fig. 1:14 with addition of inputs and outputs.

Now, closing the Scicos Graphic window that encloses the blocks within the superblock and returning back to the main schema, the superblock will show, in addition to the activation input, a signal input and an output, for now unused, as shown in Fig. 1.18.

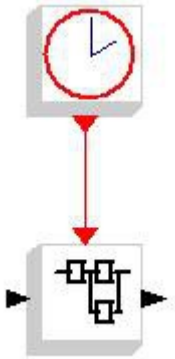

Fig. 1.18: Diagram of Fig. 1:14 with input and output to the superblock.

Add in this scheme a block "CONST_m" available in the menu "Sources" and another block PRODUCT, then set the field "Constant" of the block CONST_m at two and connect it to the created input of the superblock. Connect an extension from this new input to the input of the block PRODUCT, while the other input will carry the signal coming out of the superblock. Then add to the graph also the block "CMSCOPE" available in the menu "Sinks", which allows you to view more than one graph on the same screen, and activate the input event by setting to zero the value of the option "Accept herited events 0 / 1 "as done previously for the block cscope.

By default the block CMSCOPE accepts two inputs and then displays two graphs: inside its settings window you can still add new entries and then display multiple graphs in one screen by setting appropriately these parameters.

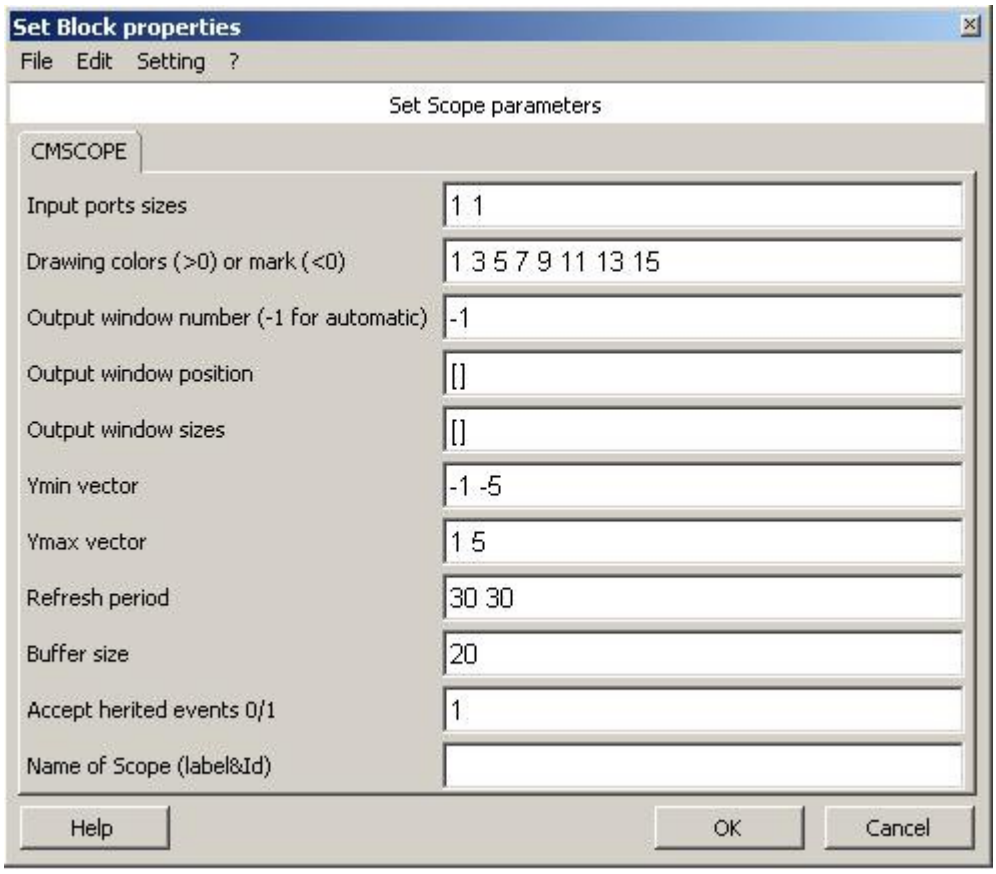

Fig. 1.19: Setting window of the block C SCOPE (Default settings)

To have a greater number of inputs you have to specify the type, for each of them, in the parameter "input port sizes". An "1", as we shall see later, corresponds to an input of real scalar type.

For each input here declared must then be specified the size of the axes of the graph by the parameters "Ymin vector", "Ymax vector" and "Refresh period", that have the same meanings of the block CSCOPE.

Returning to our example, connect to the inputs of the block CMSCOPE the output signal from the superblock and that coming from block PRODUCT just added, as shown in Fig. 1.20.

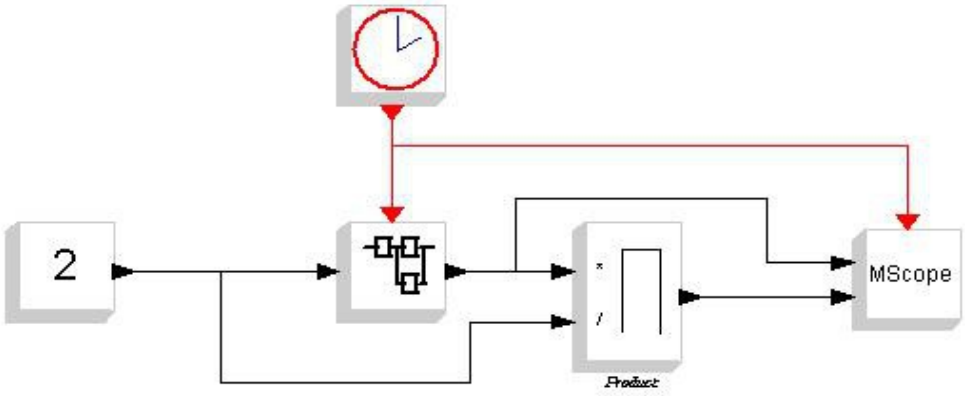

Fig. 1.20: Scheme comprising the superblock of Fig. 1.14.

Now, running the command "Simulate -> Run", will appear two screens of ScicosLab Graphic: one with a single graphic related to the block Cscope inside the superblock and one with two charts relative to the block CMSCOPE just entered.

As for the block Cscope, also for block CMSCOPE is possible to change the labels displayed on the chart axes through the settings in the menu "Edit -> Figures properties", with the difference that now the menu "Axes" present in the area "Objects browser "at the left are two.

Note that, apart from a scale factor related to the different settings of the maximum values shown on the axes, the graph produced from the block CSCOPE and the first of the block CMSCOPE are identical because they are related to the same signal.

When you have complex schemes is advisable to associate at the points of exit and entry of the superblocks some identification labels of the signals that have to be connected, to do this open the superblock and position on the I/O blocks IN_f and OUT_f, and by changing the parameter "block properties -> Identification" associate a label to the block input (or output) to the superblock. In our example we can associate the tag "in" to the input block and "out" at the output block, and the two patterns so modified finally appear as in Fig. 1.21 Fig. 1. 22.

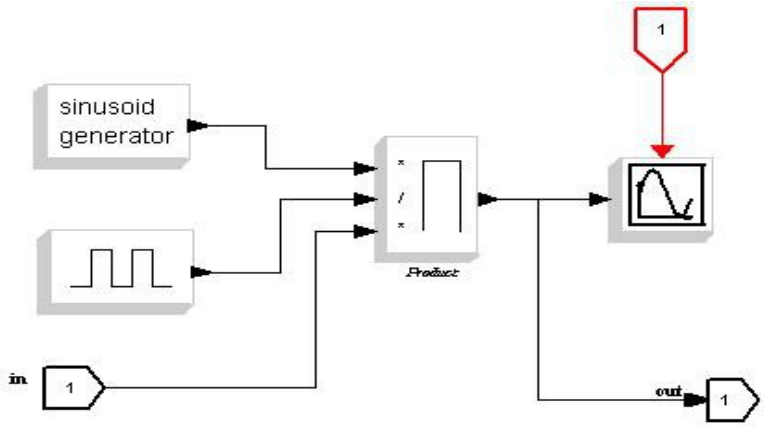

Fig. 1.21: Inner view of the Superclock created with the addiction of input and output label.

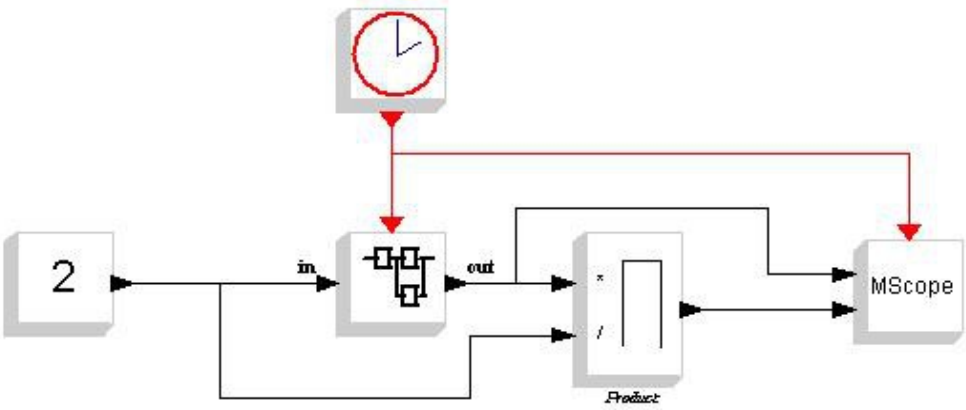

Fig. 1.22: Diagram of Fig. 1:20 after addition of labels of input and output from the superblock .

If desired, you can put all the new graphic in another superblock also contains the superblock previously created: the variables defined in the context of the overall scheme will remain valid at any level of superblock nestings.

When you are viewing the window that show the inner scheme of a superblock you can associate a name to it for not to be confused with other windows of other superblocks open: to make it go in the menu "Diagram -> Rename" and enter the name for the superblock.

As for the graphics is possible to save image related to Scicos scheme displayed on the Scicos window: to do that use the "file --> export" command that allow You to save the image as postscript file or by a "Graphic window" that open a new scheme that show the highlighted scheme resized to be filled in the display; once do that is possible to save it in various format (GIF, BMP ecc) by the command "File --> save" or "file --> export"

If we wanted to add more inputs or outputs to the superblock, we should add more blocks "IN_f" and "OUT_f" But these are always placed with the number "1" marked on the inside, and this will cause an error at the close of the superblock edited. The number enclosed in the block represents the position that the input (or output) have on the outside of the block, and as it is obvious is not possible to have two entries for the same position. To avoid this you should increasingly order the ID of the input and output blocks changing the parameter "Port number" in the settings window of the block IN_f or OUT_f, so as not to create conflicts.

1.2.1 Add new block to Scicos

Let's see how to add a new block to Scicos. Is necessary to make a short introduction: blocks Scicos can be constructed from only two code files: one that describes the action that the block has to make, called "Computational function", usually written in C language (but you can use other programming languages) and one describing the GUI of the block in the Scicos window, called "Interfacing function", written in Scilab.

The advice is to test first the computational function using a block made available by Scicos that lets you write custom code within a block with a standard interface, this block is to be found in the menu palette "Others" with the name "CBLOCK4"

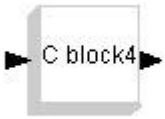

Fig. 1.23: CBLOCK4 block.

The block that will be realized will calculate the square root of its two inputs. This same feature may also be realized with the block "POWBLK_f" available in the menu " Non_linear "or by the block EXPRESSION in this same menu, but we are interested in a simple example to describe the creation of a new block, not something new.

Take, from the menu Sources two block CONST_m and the block CLOCK_c, then, from the menu Sink, for a change, the block AFFICH_m, then from the menu Linear, take the block SUMMATION and from the menu Others the cited block CBLOCK4.

Double click with the left mouse button on the blocks CONST_m and in the settings window that will open set the value of the field "Constant" respectively to ten and six.

Open the settings window of the block block SUMMATION and change the value of the "Number of input of sign vector (of 1, -1)" from "[1; -1]" to "[1; 1]" : this field indicates the operation that must be performed to the corresponding input: " 1" indicates that the input should be added to the other and "-1" which must be subtracted. Adding other "1" or "-1" you can add more inputs to the block. The order in which they will appear in the graphical interface of the block is the same, top to bottom, of how they are defined here from left to right.

If the number of inputs are considerable and the input arrows result too close, you can **Change the size of the block** (as of any other block) by clicking with the right mouse button on it and selecting "Block Properties - > Resize ".

Leaving aside momentarily blocking CBLOCK4, we conclude the description of the scheme saying that blocking AFFICH_m simply serves to display the numeric value of the input received: for proof that we are going to do the default settings are fine, however the advice is always to do several attempts with various options to see the effects and how they work independently: the practice teaches better than a thousand words. In place of this block we could use the block Cscope already seen, and as it is a static system we could have omitted the block CLOCK_c, however we move forward with what positioned and realize the scheme of Fig. 1.24.

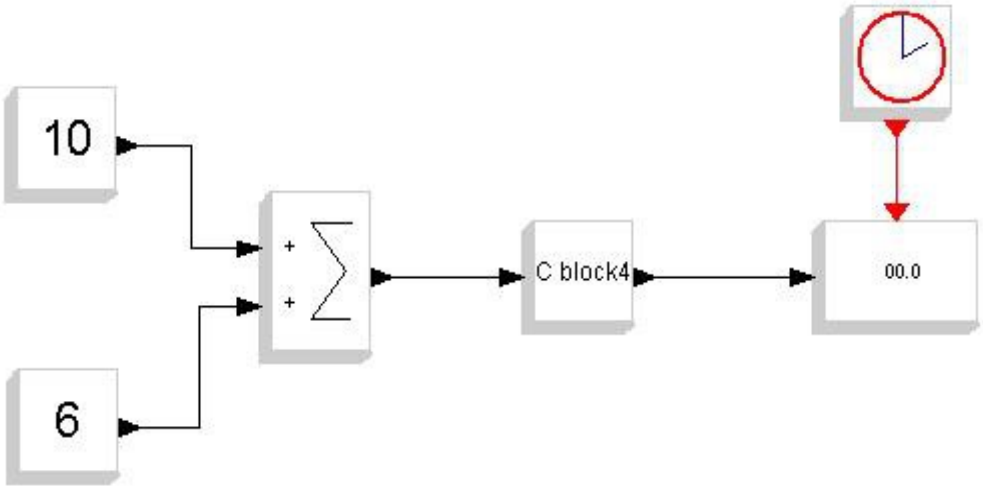

Fig. 1.24: Block CBLOCK4 evaluation scheme.

Now we have to enter the code in the block CBLOCK4: to make it open its settings window, which will appear as in Fig. 1.25

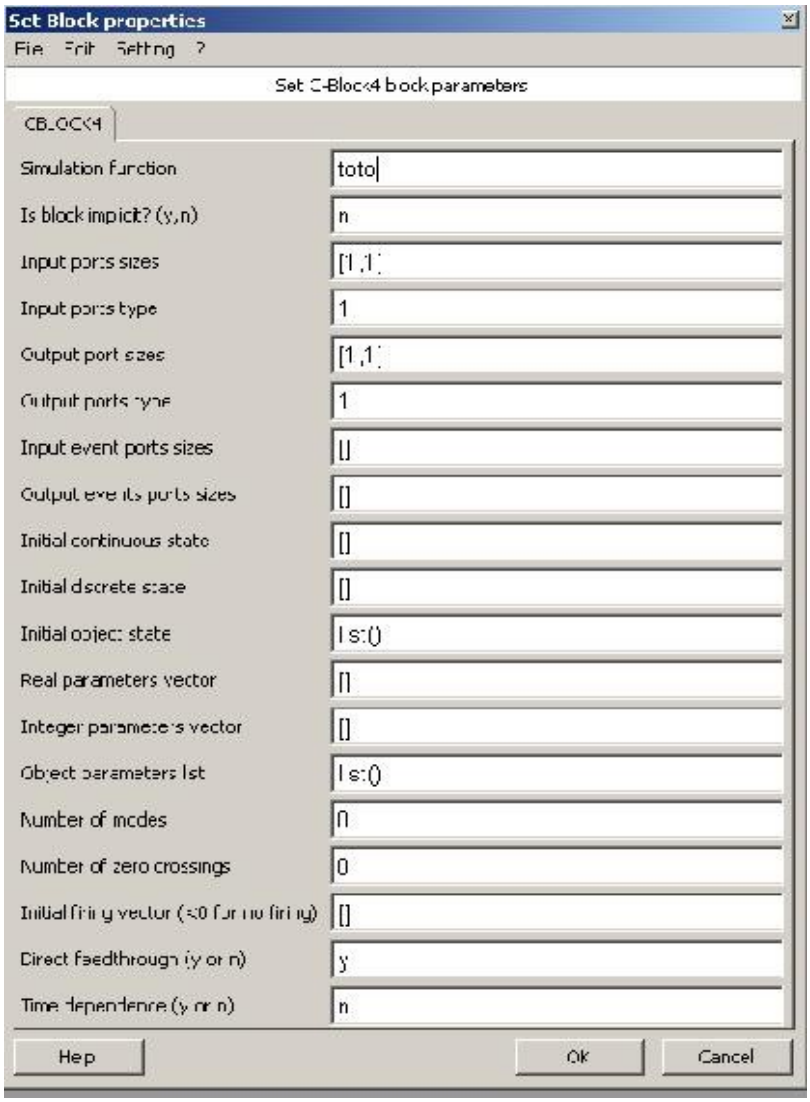

Fig. 1.25: Setting window of block CBLOCK4.

The "Simulation function" indicates the name of the function that the block will perform: the suggestion is to set the same name of the computational function choice, in this case, such as "Square".

The "Imput port sizes" indicates the number and the size of the block inputs: the message "1, 1" indicates that we have an input scalar type (1 * 1). For example, if we wanted two inputs of the same type we would have to change this voice by writing "[1, 1, 1, 1]"

The field "Imput port type" indicates the type of input to the block: the number 1 indicates, as already seen, a real input. We shall see later how the data types are associated with these numbers in Scicos, for now leave it so.

The items "Output port size" and "Output port type" indicate the same things respectively of the items "input port type" and "input port size", but related to the outputs.

The items "Input event port sizes" and "output event port sizes" indicate the number and type of inputs and outputs of events to the block (the "red arrows"). In our case we do not need them so we can leave these fields blank.

The item "Initial continuous state" indicates, for blocks that contain an internal state, the value that this must have at the beginning of the simulation.

The item refers to the case of a continuos time type function, for blocks containing discrete time type functions we will use the field "Initial discrete state".

The meaning of the rest of the items on the menu, along with all the others, is available by clicking on the "Help" button at the bottom left of the setting window. The "Help" menu is available for almost all the blocks in the same way: by selecting the Help button at the bottom of the setting window.

Continuing with our example, then, the only change that we make to the default settings is to set the title of the function in "Square". (Replace, in the field "Simulation function" of Fig. 24, the name "toto" with "Square"). Subsequently it will open a window titled "C function" containing the basic template of the computational function of Scicos, appearing as shown in Tab. 1.2.

```
#include <scicos/scicos_block4.h>

void Square(scicos_block *block,int flag)
{
  /* init */
  if (flag == 4) {

  /* output computation */
  } else if(flag == 1) {

  /* ending */
  } else  if (flag == 5) {

  }
}
```

Tab. 1.2: Basic structure of a Scicos block computational function in C.

The Scicos blocks defined with a computational function in C are based on the same basic structure through which you can access all the information of the block under examination: inputs, outputs, parameters and internal states.

This structure is defined in the file "scicos_block4.h" and this explain why we need to include this header in all of the computational function written in C.

During the simulation, the computational function is called with a flag which corresponds to the task to be executed. On the first execution the function is called with flag = 4 that performs the procedures of initialization of variables of the block (of the input, state and output) and then proceed with the flag = 1 which corresponds to the computation of outputs based on the values of inputs and of the state.

In case we have also state variables, subsequently to the flag 1, the computational function will activate the function corresponding to the flag = 2 in which are updated state variables defined within the block.

When the simulation ends, or in case of an error that requires abortion of the simulation, the last flag to be called is the 5 that addresses the procedures for releasing the internal memory for any variable declared in the initialization phase.

Is possible to activate also other flags related to other tasks for specific

operation, but for this example they does not interest us and since they are referred to advanced functionality their explanation is left to more specific tutorial.

To realize the function that calculate the square root of the imput the computational function must be modified as shown in Tab. 1.3.

```
#include <scicos/scicos_block4.h>
#include <math.h>

void Square(scicos_block *block, int flag)
{
  double *in = GetRealInPortPtrs(block, 1);
  double *out = GetRealOutPortPtrs(block, 1);

  /* init */
  if (flag == 4) {
     out[0] = 0;
     }

  /* output computation */
  else if(flag == 1) {
     out[0] = sqrt(in[0]);
     }

  /* ending */
  else  if (flag == 5) {
   }
}
```

Tab. 1.3: Computational function of the example of Fig. 1.23

Once the code has been changed as above press "ok" and ignore the next window in which we are asked whether to link external libraries, because we do not need them. Press "OK" in this window to return to the Scicos diagram. At this point, starting the simulation, the block AFFICH_m will display the value 4.0

Let's analyze line by line the code above: we already seen the directive "#include <Scicos / scicos_block4.h>" the next "#include <math.h>" is used to include the math library of C needed to be able to use the function "sqrt ()" that calculates the square root of the number entered as a parameter.

The line "void Square (scicos_block * block, int flags)," declares "Square" as a function that takes two parameters, a pointer to a structure of type "scicos_block" that represent the file itself, and an integer that will be used to select the task to be performed every time that the function will be called, as described previously.

The subsequent statements

double * in = GetRealInPortPtrs (block, 1);
double * out = GetRealOutPortPtrs (block, 1);

initialize two pointers to double type variables that represent the input and the output of our block, with which are associated by the macros "GetRealInPortPtrs (block, 1)" and "GetRealOutPortPtrs (block, 1)".

In Tab. 1.4 there are some of the most commonly used macros to define the inputs and outputs for the most common simple cases:

Macro	return	Description
GetInPortPtrs(block, x)	void*	Returns a pointer to the input port "x" specified.
GetRealInPortPtrs(block, x)	double*	Returns a pointer to the real part of the input port "x" specified
GetImagInPortPtrs(block, x)	double*	Returns a pointer to the imaginary part of the input port "x" specified
Getint8InPortPtrs(block, x)	char*	Returns a pointer to the character typed in the input port "x" specified
Getint16InPortPtrs(block, x)	short*	Returns a pointer to the data type short on the input port "x" specified
Getint32InPortPtrs(block, x)	long*	Returns a pointer to the data type long present on the input port "x" specified
Getuint8InPortPtrs(block, x)	unsigned char*	Returns a pointer to the data type unsigned char present on the input port "x" specified
Getuint16InPortPtrs(block, x)	unsigned short*	Returns a pointer to the data type unsigned short present on the input port "x" specified
Getuint32InPortPtrs(block, x)	unsigned long*	Returns a pointer to the data type unsigned long present on the input port "x" specified
GetOutPortPtrs(block, x)	void*	Returns a pointer to the output port "x" specified
GetRealOutPortPtrs(block, x)	double*	Returns a pointer to the real part of the output port "x" specified
GetImagOutPortPtrs(block, x)	double*	Returns a pointer to the imaginary part of the output port "x" specified
Getint8OutPortPtrs(block, x)	char*	Returns a pointer to the character present on the

		output port "x" specified
Getint16OutPortPtrs(block, x)	short*	Returns a pointer to the data type short present on the output port "x" specified
Getint32OutPortPtrs(block, x)	long*	Returns a pointer to the data type long present on the output port "x" specified
Getuint8OutPortPtrs(block, x)	unsigned char*	Returns a pointer to the data type unsigned char present on the output port "x" specified
Getuint16OutPortPtrs (block, x)	unsigned short*	Returns a pointer to the data type unsigned short present on the output port "x" specified
Getuint32OutPortPtrs (block, x)	unsigned long*	Returns a pointer to the data type unsigned long present on the output port "x" specified

Tab. 1.4: List of the most used macros to define inputs and outputs computational function written in C.

To clarify the relationship between the data type and the name of the macro, as well as the interpretation of the data types used in ScicosLab, we can refer to Tab. 1.5 that associates each data type expressed in the Scilab language the corresponding C data type as well as the number with which this is represented in the declaration of the inputs and outputs in the Scicos blocks.

Scilab	C	Scicos block
real	double	1
complex	double	2
int32	long	3
int16	short	4
int8	char	5
uint32	unsigned long	6
uint16	unsigned short	7
uint8	unsigned char	8

Tab. 1.5: Matches between variables in Scilab language and C language.

Now it should be clear why in the items "Imput port type" and "Output port type" in the settings window of the block CBLOCK4 we placed the number 1: (Fig.1.24) in this way the inputs to our block will be defined as "Real ", that correspond to the double type in C.

Continuing with the analysis of the code, in the body of the "Square" function we meets a flow control based on the switch statement: we have

already described the meaning of the activation flags in a computational function in Scicos, in these fields we are going to write the code which in each case will be gradually carried out by the function.

In our case the initialization just reset the output (flag = 4), while for the computation of the outputs (flag = 1) arises in the output value of the square root of the inputs calculated with the sqrt () function.

The use of a vector for to access at the output port (as for the input) in this case represents a refinement more conceptual than practical: having an output of scalar type we could also assign the output via the pointer to the name of the same:

*out = sqrt(*in);

In the generic case in which we have a vector type output, however, because the name of a vector represents a pointer to the first element of the vector itself, to access the various elements of the vector we should direct them in the same manner as we do with vectors, as done above.

Il flag = 5 risulta vuoto in quanto per la funzione Square non occorre eseguire alcuna operazione specifica al termine della simulazione.

The flag = 5 is blank because the function Square do not require any specific action at the end of the simulation. After checking the validity of the computational function by the usage of the block CBLOCK4, to add a new block to Scicos is necessary to write the interfacing function of the block.

The interfacing function, as mentioned, is written in Scilab language and unfortunately there aren't many tutorials on how to proceed: I do not deny that many of the statements herein have been identified by trial and therefore may be subject to error, in which case I ask forgiveness in advance, hoping that they are equally useful.

However, as for the computational function, also the interfacing function is based on a basic template structure to be edited from time to time depending on the block: for my experience, the skeleton of a basic interfacing function is reported in Tab. 1.6 (already adapted to our function Square)

```
function [x,y,typ]=Square(job,arg1,arg2)
x=[];y=[];typ=[]

select job
case 'plot' then
  standard_draw(arg1)
case 'getinputs' then
  [x,y,typ]=standard_inputs(arg1)
case 'getoutputs' then
  [x,y,typ]=standard_outputs(arg1)
case 'getorigin' then
  [x,y]=standard_origin(arg1)

case 'set' then
  x=arg1;

case 'define' then
```

```
  model = scicos_model()
  model.sim = list("Square",4)
  model.in  = [1]
  model.out = [1]
  model.blocktype='c'
  model.dep_ut = [%t %f]
  gr_i =
['txt=[''Square''];';'xstringb(orig(1),orig(2),txt,sz(1),
sz(2),''fill'');']
  exprs=list()
  x = standard_define([2 2],model,exprs,gr_i)
  end
endfunction
```

Tab. 1.6: Interfacing funcion of sunction "Square".

let's check the code line by line:
The first line,

function [x, y, typ] = Square (job, arg1, arg2)

is used by the editor of Scicos to have a reference to the block whenever it needs to access its functions. For simplicity I used the same name of computational function but this is not strictly necessary.

The next lines in which a specific "job" is selected represent the core of the *interfacing function*:in the first 9 rows (rows 4-12) the standard functions of Scilab are used to draw and place the block in the diagram and to assign the inputs and outputs to the block itself (The *interfacing function* is not completely disconnected from the *computational function*: besides the obvious fact for which the number of inputs and outputs defined in the computational function must be the same as stated in the *interfacing function*, also for other aspects apparently irrelevant from the point of view of visualization-As the presence of an internal state in the *computational function*- is necessary to provide precise definitions in the *interfacing function*).

The case 'set' is used for the description of the setting window of the operating parameters of the block (when we set the value of the constant internal to a block CONST_m, for example, the window that appears by the double click of the mouse on the block is defined in this "case". Of course also in this case the number of parameters to be set must be consistent with that defined in the computational function)

In our case we have no parameter to be set so just use the line

x = arg1;

That initializes the inputs.

The case 'define' contains the most critical link between computational function and interfacing function: the line

scicos_model model = ()

indicates that the *interfacing function* created is referred to a Scicos block whose model is defined by the function "scicos_model ()" (as it was necessary to insert the directive "`#include <Scicos / scicos_block4.h>`" at the beginning of the computational function)

The line

model.sim = list ("Square", 4)

indicates that the name of the *computational function* called by the block (SIMulation function) is "Square" and is written in C, then the block will be linked to a file "Square.c".

As mentioned, the *computational function* can be written in different languages: for example, if we had written a *computational function* in Scilab language we would have to write this line as "model.sim = list (" Square ", 5)"

The rows

model.in = [1]
model.out = [1]
model.blocktype = 'c'

indicate that our block will have only one input (If we had two we should have written "model.in = [1; 1]") and a single output (same as for the inputs) and that, again, the *computational function* is written in C.
The line

model.dep_ut = [% t% f]

serves to indicate to the *interfacing function* the behavior of computational function via two Boolean variables: dept_u (the first parameter) and dep_t.

If the variable dep_u (% t) is set to true the block is always active, ie the outputs depend also from the time (Which does not mean they necessarily have to change!) While the variable dep_t, if true, indicates that the block It has at least one output whose value depend only on the input values (without depending from internal states or parameters).

The line

gr_i = ['txt = [' 'Square' '];'; 'xstringb (orig (1), Original (2), txt, sz (1), sz (2),' 'fill' ');']

represent the graphical aspect of the block: the text and the position of the same, the size of the font etc ...

The line

exprs = list ()

contains an array of strings representing the default values displayed in the window setting of the block in the fields used for to set the parameters of the block: our block has no parameter to initialize so the list is empty.

The line

x = standard_define ([2 2], model, exprs, gr_i)

create through a standard function of Scilab the real block defining the size and linking the directives given in the various cases previously setted (the parameters in the field "model." defined in the case 'define', the window setting defined in case 'set', the graphical interface of the "gr_i" etc.) for the block to be fully functional.

Once our *computational function* has been tested and created its *interfacing function*, is necessary to save them for compiling and linking them in Scilab.

To do it open notepad and copy the code of the *computational function*, saving it as a C file in a default folder (or create a new dedicated folder) and do the same for the *interfacing function* saving it as a ".sci" file ("Square.sci"). Then open Scilab and set the work directory in the folder where you saved the file above (using the "Change directory" command using the button showing the yellow folder in the toolbar below the Scilab menu, between the words " Preferences "and" Control ", or by the command" cd ('path') ")

Once you positioned the working directory in the right folder you must compile and link the *computational function* by typing the following command:

libn=ilib_for_link("Square", "Square.o", [], "c", "Makelib");

This command require five parameters: the first, "Square", indicates the name of the function to be linked in Scilab, the second, "Square.o", the name of the object that will be created by the compiling process of our source code, the third is a list of external libraries needed to create the object (none in this case, that's why the list is empty), the fourth indicates that the code to be linked is written in C language and the fifth is the name of the default makefile.

Once the command is executed, in the default folder will be created seven new files, including a "loader.sce" file: this have to be executed in Scilab by typing

exec loader.sce;
and, after, compile the interfacing function by typing

exec Square.sci;

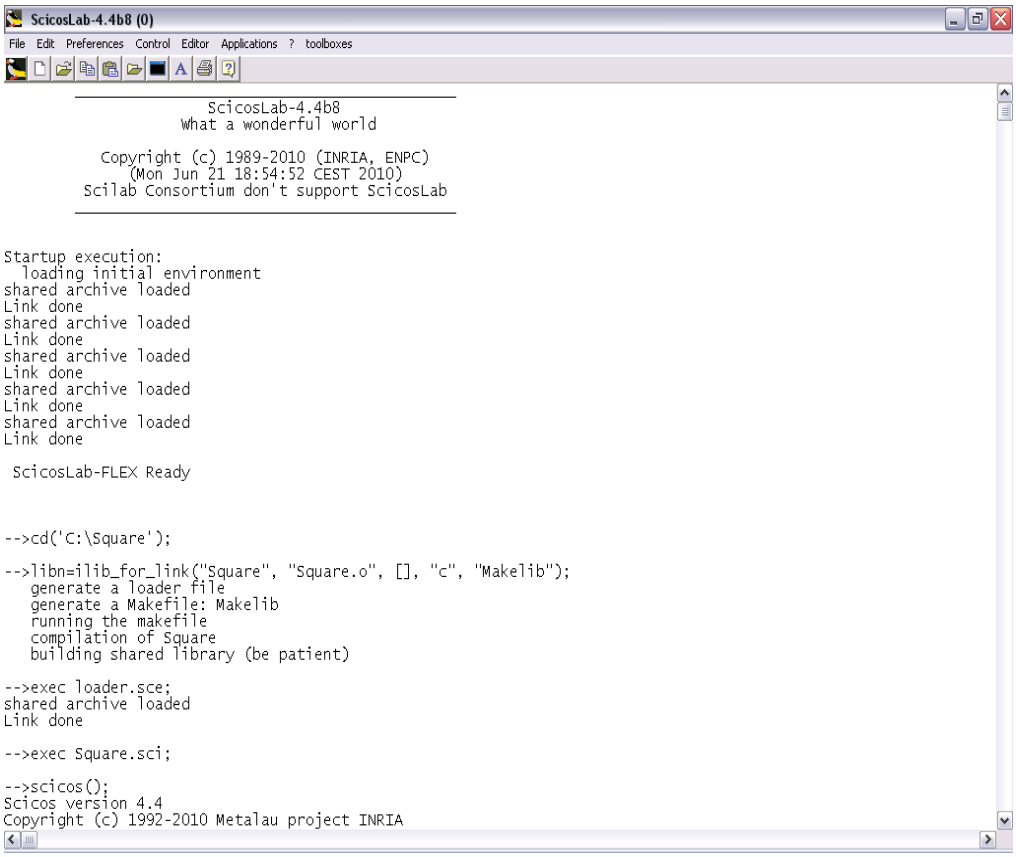

Fig. 1.26: Compilation and linkage of computational and interfacing function in ScicosLab / Scicos.

Now open Scicos and select the "Add new block" option available under the "Edit" menu as in Fig. 1.27.

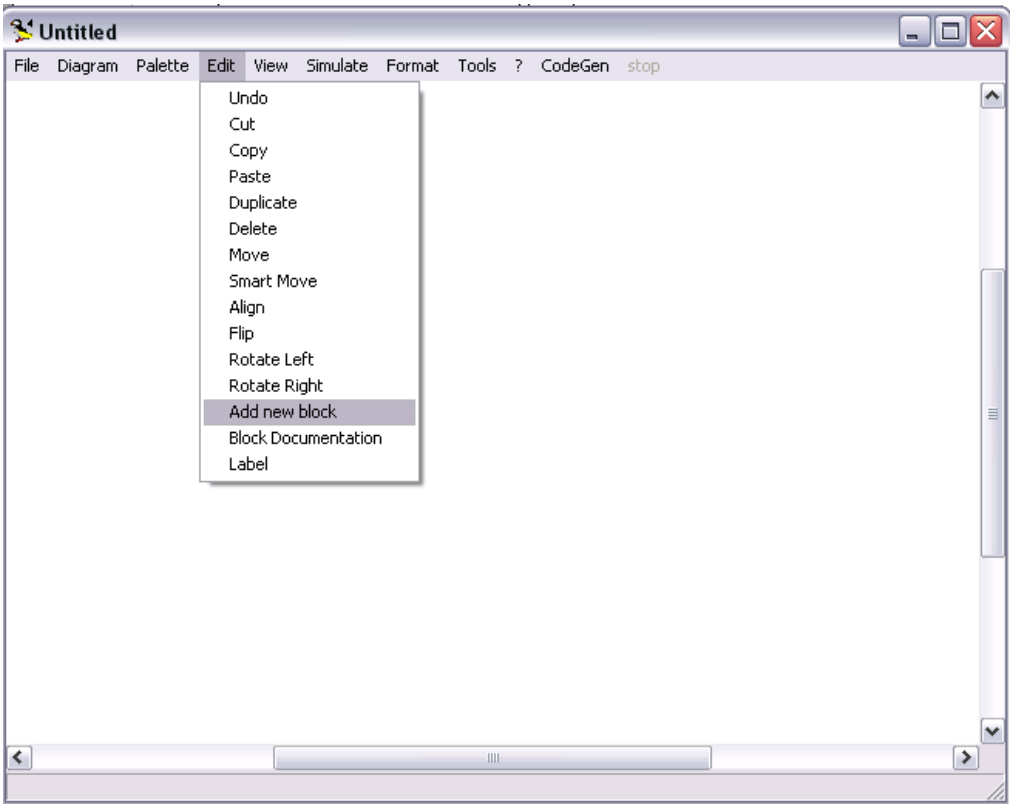

Fig. 1.27: Path to add a new block to Scicos.

In the window that will appear (see Fig. 1.28) insert "Square" and type "Ok"

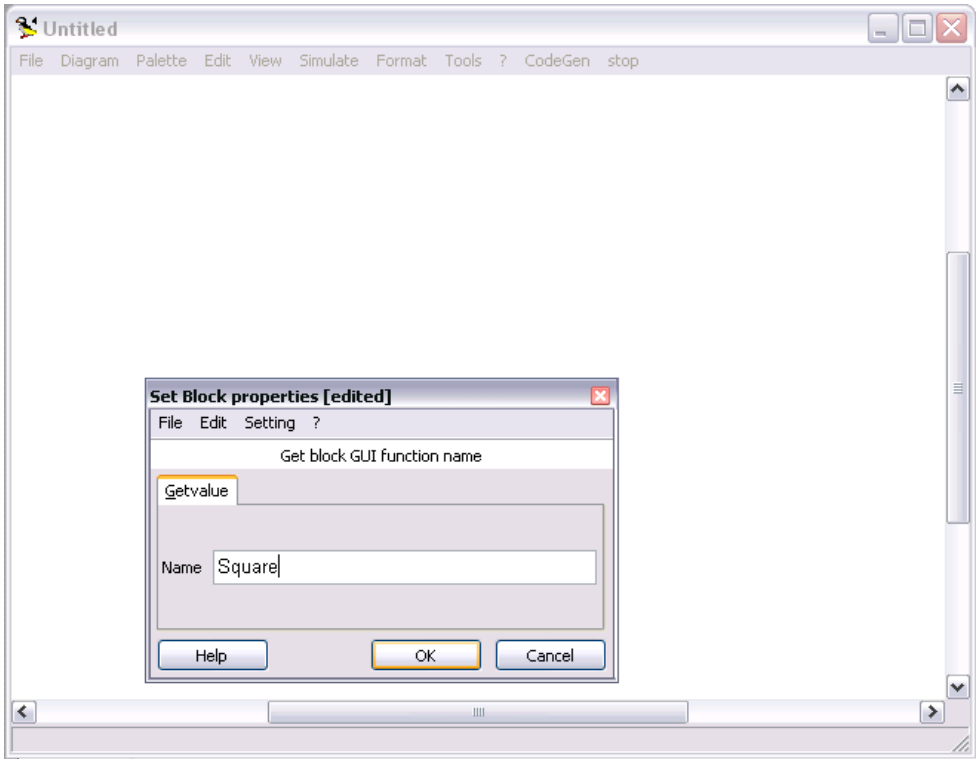

Fig. 1.28:Selection of the name of the new block that have to be added io the scheme

If everything is done correctly we will see in the scheme this our new block, which will appear as in Fig. 1.29.

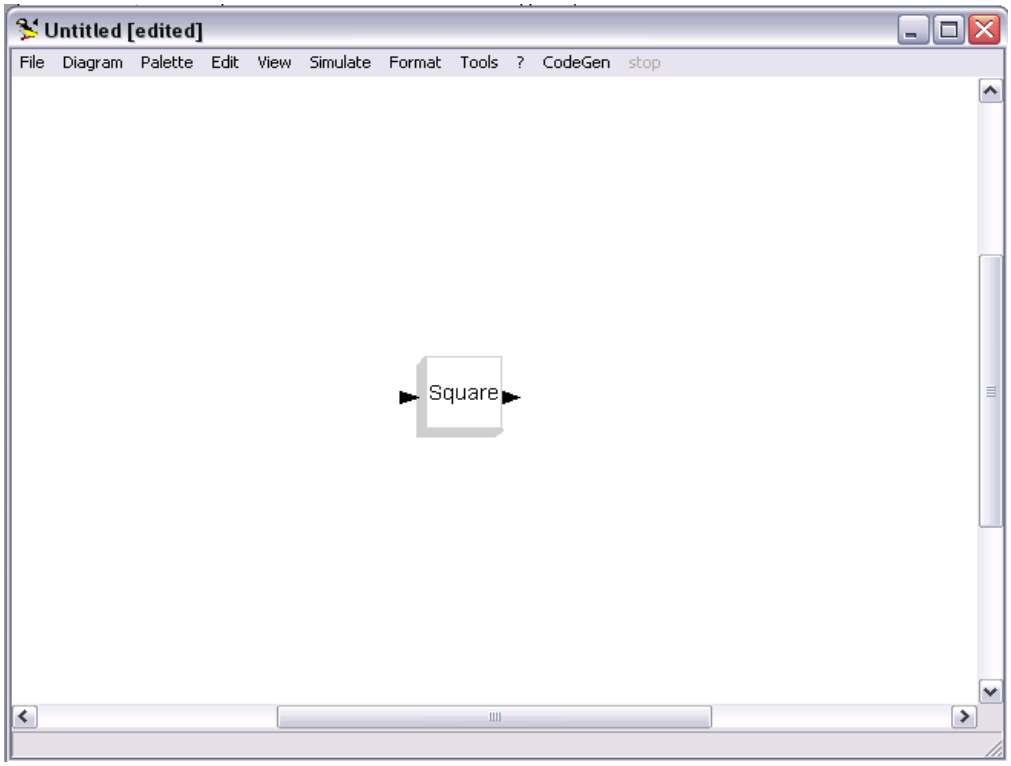

Fig. 1.29: The new block created placed in Scicos.

To test the new block created with the new interfacing function we just have to create a test scheme: the result should be identical to that obtained with the CBLOCK4 containing the *computational function* previously tested.

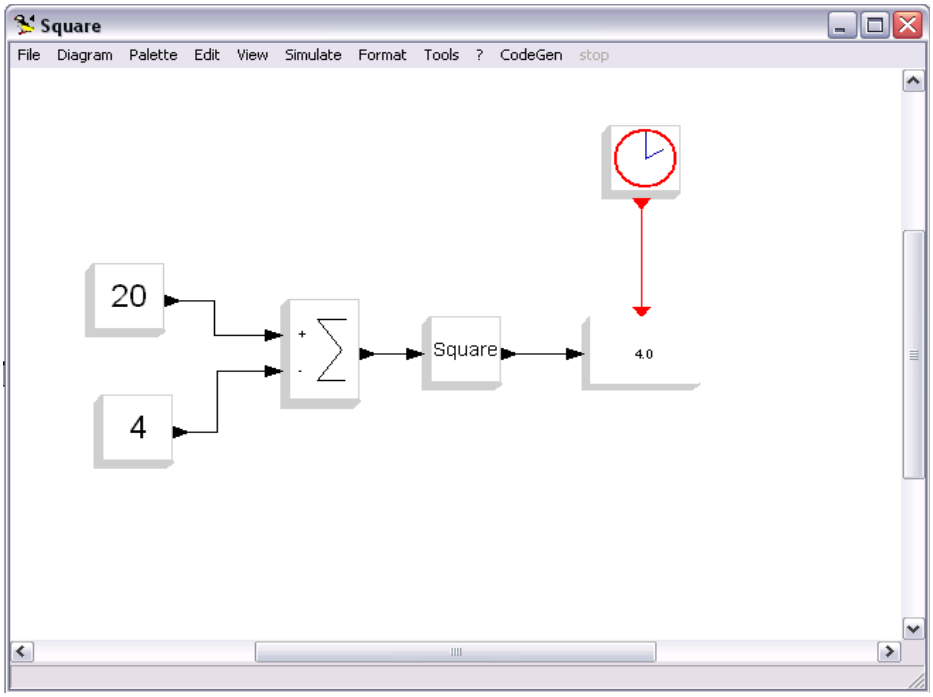

Fig. 1.30: Scicos scheme containing the new block created.

If during our work on Scicos, we needed to get back in ScicosLab to change the working directory, or perform calculations, we can do it without closing the Scicos window using the "Activate ScicosLab Window" command

located under the "Tools" menu; once you finished the work on ScicosLab you can return on Scicos typing "Scicos ();" or via the menu "Applications" of ScicosLab (or, more simply, clicking again on the window Scicos)

1.3 Bibliography

[1] THESIS, Ing Lorenzo Lombardi, *"Implementazione di algoritmi di controllo della traiettoria e di comunicazione per un robot a controllo remoto e relative verifiche sperimentali*", Ultrasound and NDT labs, university of Florence, Italy, 2011.

[2] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah *"Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4"*, second edition, Springer.

[3] Phil Schmidt, *"Creating a C Function Block in Scicos"*, tutorial online, http://www.scicos.org/ScicosCBlockTutorial.pdf

[4] Scicos Team, *"Constructing new blocks in Scicos"*, tutorial online, http://www.scicos.org/Formation_scicos_mars_2008.pdf