5.2 TUTORIAL EXAMPLE 2

The following assumes that the user has completed Tutorial Example 1.

This example is an altered version of Example 1. There are now ten adder nodes instead of two and each adder prints it's number and the sum it has calculated when it completes. Replication is used to produce the ten adders.

Contents of the Tutorial

I. Preliminaries
II. Entering Example 2
  Step 1. Draw all the nodes and arcs in a graph.
  Step 2. Use attribute forms to enter all information about the nodes, arcs, graphs and programs.
III. Executing Example 2 on the Sequent.
  Step 1. Save and compile the program.
  Step 2. Create the files which record the node function definitions.
  Step 3. Create the executable and run it.

I. Preliminaries

Get the X-Windows system running. Create a directory to hold the files associated with the program and move to that directory. Before running CODE, ensure that it is present on your UNIX path. Then enter

```
code2 rsum.grf
```

at the UNIX prompt. This will run CODE and open a program called "rsum". A single window will be opened and all activity will take place within it. Fig. 4.1 shows the CODE window.

II. Entering Example 2

Our goal is to draw the following graph:

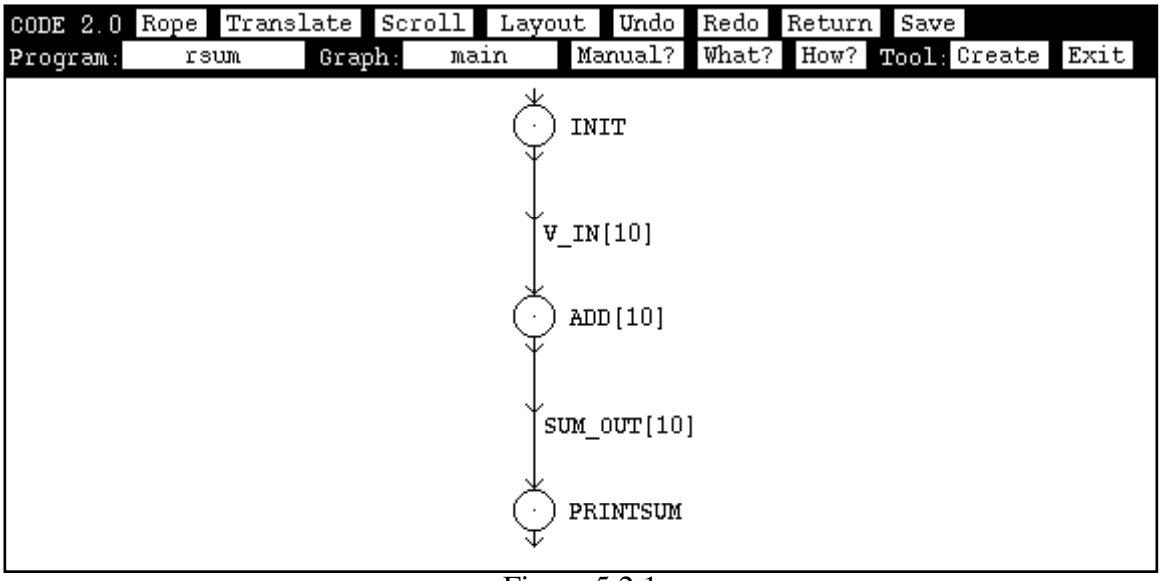

Figure 5.2.1

You may enter and annotate the nodes and arcs of your program in any order, but we will adopt the following sequence of operations:

Step 1. Draw all the nodes and arcs in a graph.
Step 2. Use attribute forms to enter all information about the nodes, arcs, graphs and programs.

STEP 1. Draw all the Nodes and Arcs in a Graph

A. Draw all the nodes.

1.Draw the UC node, INIT by

a. Position the create cursor in the CODE window, where the UC INIT is to be located.
b. Click with the left mouse button, to bring up the menu of node types.
c. Pick the Unit of Computation (UC) node by clicking on it.

2. Draw the UCs ADD[10] and PRINTSUM.

.

B. Draw all the arcs.

1. Draw the arc V_IN[10] by,

a. With the create cursor, click on the lower arrow of UC, INIT.
b. Also, click on the upper arrow of UC, ADD[10].

2. Draw the arc SUM_OUT[10].

At this point the graph looks like Fig. 5.2.2.

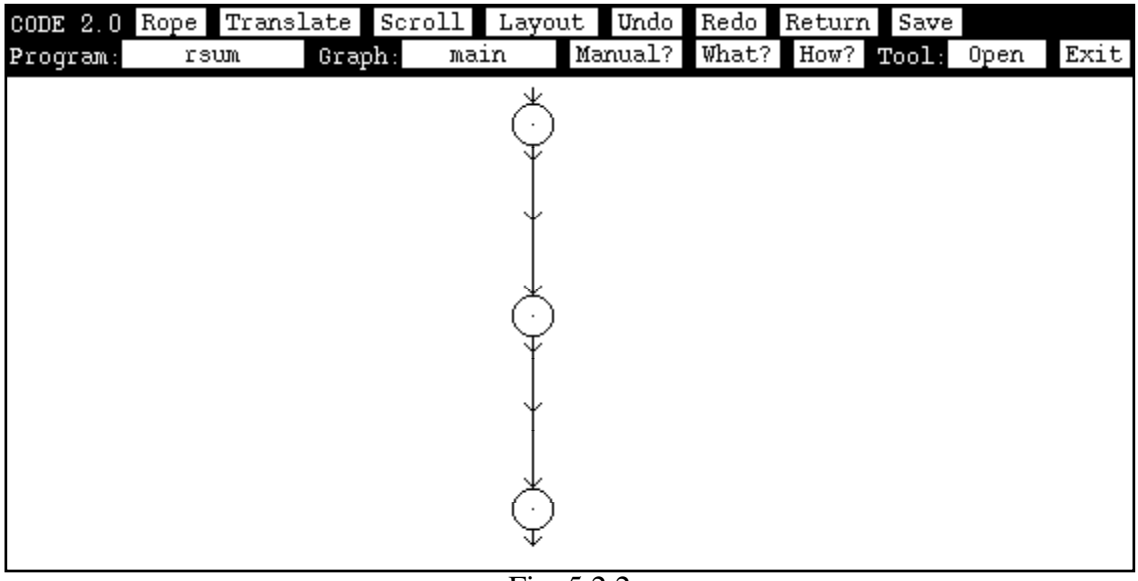

Fig. 5.2.2

STEP 2. Use Attribute Forms to enter all Information about Nodes, Arcs Graphs and Programs

A. Enter information about the nodes

1. Enter information about INIT

   a. Select the open cursor from the tool window.

   b. Click the open cursor on the UC INIT to open its attribute form. It looks like Fig.5.2.3

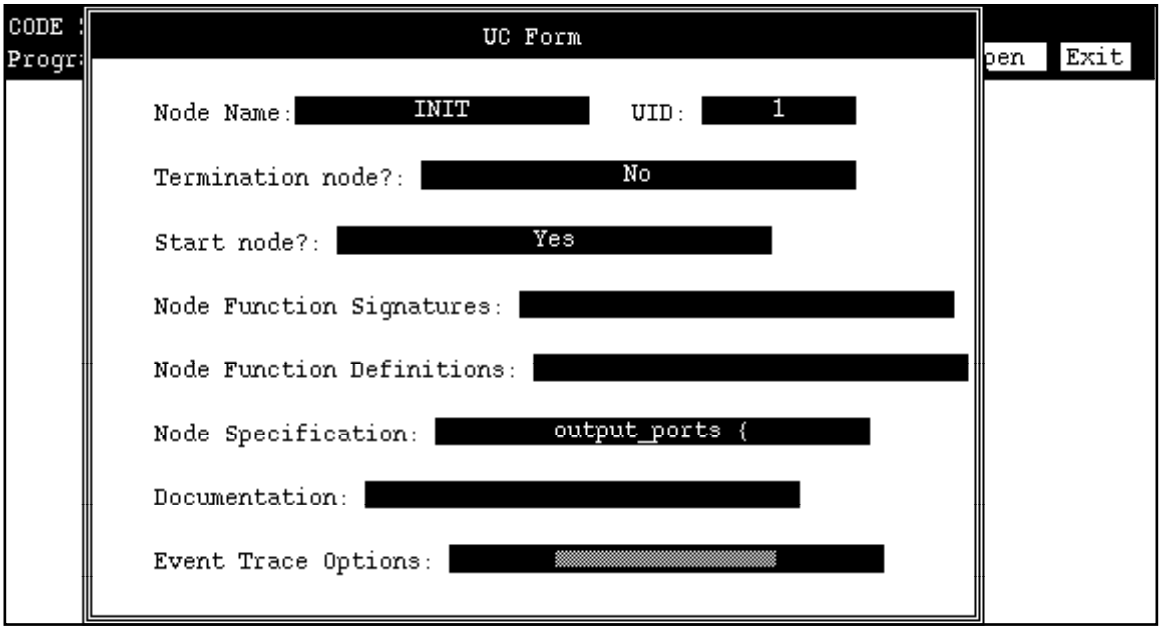

Fig. 5.2.3

c. Click on the node name field and a text box will appear. Enter the name 'INIT' in this box. You can make the text box go away by clicking on the white space in the UC form to return to it.

d. The value in the UID field is automatically generated by CODE. It is a debugging aid which you can ignore for now.

e. The termination node and start node fields will have a default value of 'No'. Since INIT is a start node, update the Start node field to 'Yes'.

f. The Node Function Signatures and Node Function Definitions fields are to be left blank.

g. Click on the Node Specification field. A text box labelled 'Text of UC Specs' will appear. The code to be entered in this text box is as shown below. This code can also be entered via a window running the editor defined by your EDITOR environment variable. To bring up this window press ESC-e (the escape key followed by e). After entering the code, you can exit from the window, the way you would normally exit from your editor.

```
output_ports {
 Vect OV;
}

vars {
 Mat V[10][10]; int i; int j; int k;
}

comp {
 k=0;
 j=0;
 while (k<10)
 {
 i=k*10;
 while (j<10)
 {
 V[k][j]=i;
 j=j+1;
 i=i+1;
 }
 k=k+1;
 j=0;
 }
}

routing_rules {
 TRUE => { OV[i] <- V[i]; : (i 10) };
}
```

h. The documentation field is to be left blank.

i. To exit the UC INIT attribute form, click on the white space outside the form in the CODE window.

The name 'INIT' will be displayed next to the node, but you can move it with the Move cursor.

2. Enter information about ADD[10] as in 1.

The one difference is that ADD[10] is not a Start node. The code to be entered in the Node Specification field of ADD[10] is as shown below,

```
input_ports {
 Vect IV;
}

output_ports {
 int SUMS;
}

vars {
 Vect C[10]; int sum; int i;
}

firing_rules {
 IV -> C =>
}

comp {
 i=0;
 sum=0;
 while (i<10)
 {
 sum=sum+C[i];
 i=i+1;
 }
}

routing_rules {
 TRUE => SUMS <- sum;
}
```

3. Enter information about PRINTSUM as in 1.

The one difference is that PRINTSUM must be designated as a termination node. The code to be entered in the Node Specification field of PRINTSUM is as shown below,

```
input_ports {
 int I_SUM;
}

vars {
 Vect p_sum[10]; int i; int t_sum;
}

firing_rules {
```

```
 { I_SUM[i] -> p_sum[i] : (i 10) } =>
}

comp {
 routine1(p_sum);
}
```

B. Enter information about the arcs

1. Enter information about V_IN[10]

   a. Click the open cursor on the arc V_IN[10] to open its attribute form. It looks like Fig. 5.2.4.

   b. Click on the name field and a text box will appear. Enter the name 'V_IN[10]' in this box. You can make the text box go away by clicking on the white space in the Value Passing Relation form to return to it

   c. The value in the UID field is automatically generated by CODE.

   d. Click on the Arc Topology Rule field and another text box will appear. The code to be entered in this text box is shown below,

```
.OV[i] => [i].IV
```

   e. Exit from both the text box and the form by clicking on the white space outside the two in the CODE window.

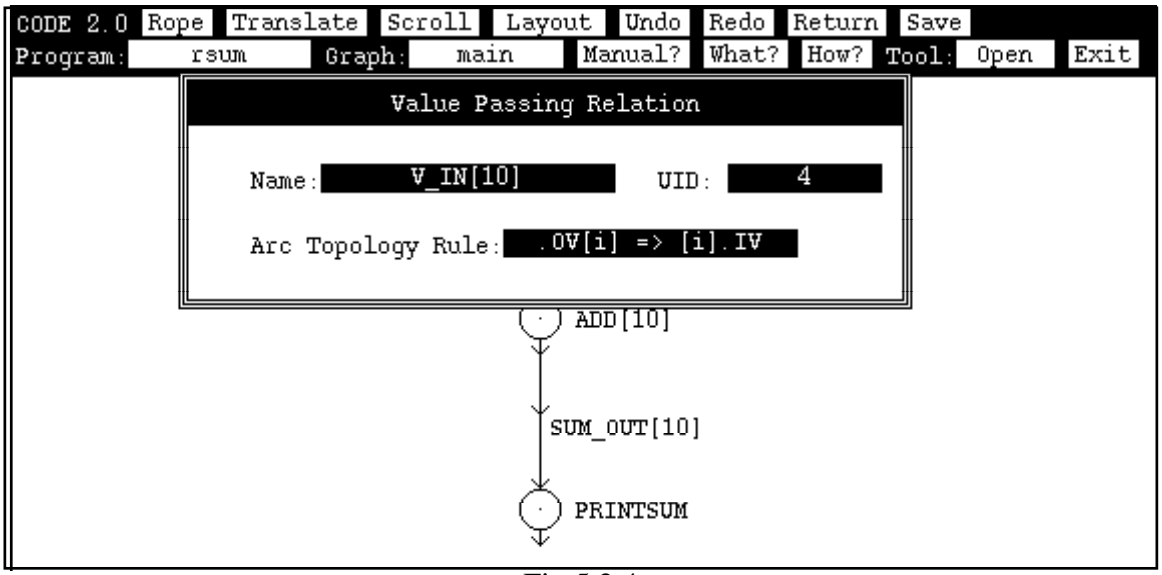

Fig.5.2.4

The name 'V_IN[10]' will be displayed next to the arc, but you can move it with the Move cursor.

2. Enter information about SUM_OUT[10] as in 1.

The Arc Topology Rule field of SUM_OUT[10] is,

```
[i].SUMS => .I_SUM[i]
```

C. Enter information about the program

a. Click the open cursor on the Program button in the CODE window, to open its attribute form. It looks like Fig. 5.2.5.

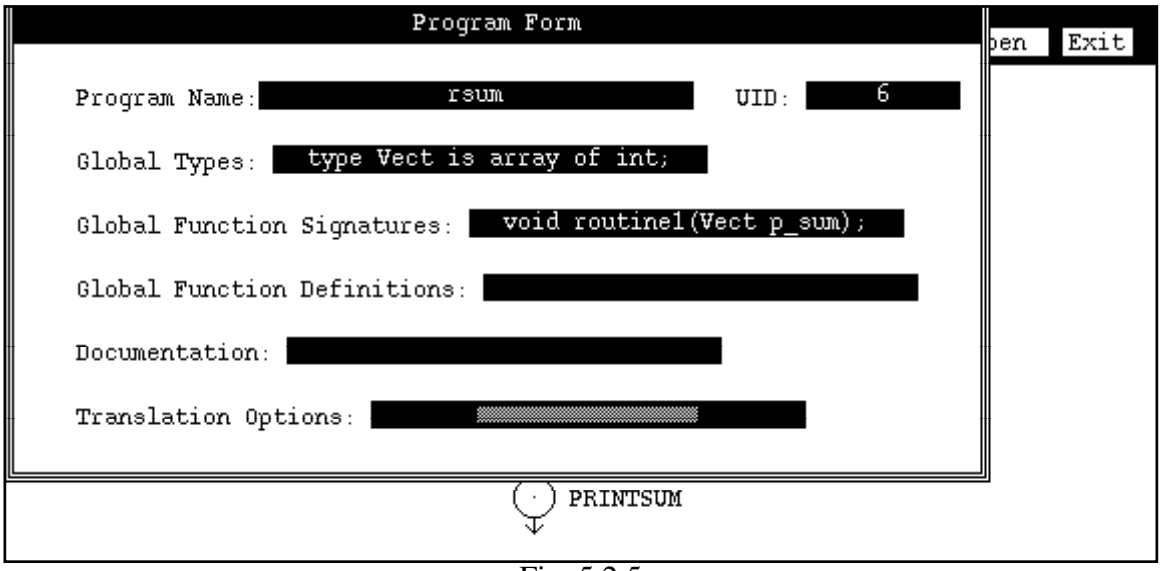

Fig. 5.2.5

b. The values in the program name and UID fields are automatically generated by CODE.

c. Click on the Global Types field and a text box will appear. The code to be entered in this text box is shown below,

```
type Vect is array of int;
type Mat is array of Vect;
```

d. Click on the Global Function Signatures field and another text box will appear. The code to be entered in this text box is shown below,

```
void routine1(Vect p_sum);
```

e. The Global Function Definitions and Documentation fields are to be left blank.

f. Click on the Translation Options field, to bring up it's attribute form. Now, click on the 'Object Files to Link' field. A textbox will appear. Enter the name of the file in which node function definitions will be specified, in this textbox.

```
filename.o
```

g. Exit from the program form by clicking on the white space outside the form in the CODE window.

III. Executing Example2 on the Sequent

We will now compile and run the program in the following order:

Step 1. Save and compile the program.
Step 2. Create the files which record the node function definitions.
Step 3. Create the executable and run it.

STEP 1. Save and Compile the Program

a. Click on the Save button.
b. Click the create cursor on the Translate button and select the 'Sequent' option.
c. If there are any errors, these are displayed in the window from where CODE was run.
d. Eliminate the errors, if any.
e. After the program has been successfully compiled and saved, exit the program by clicking on the Exit button. Select the 'save' option and quit.

At this stage, the following files and directories will be present under the directory example1.

rsum.grf	File
rsum.sequent	Directory

STEP 2. Create the Files which record the Node Function Definitions

$cd ./rsum.sequent — The following files will be present in this directory.
Makefile
c2_globtype.h
c2_main.h
c2_main.c
main.c

$vi filename.c — Record all node function definitions in this file. Give it the same name as the object file to be linked.The text to be entered in this file is as shown below. This file “routines.c” can also be picked up from the directory,
`examples/tutorial2/rsum.sequent.`
These directories will be present under the directory in which CODE2 has been installed.

```
routine1(p_sum)
int p_sum[];
{
 int i,t_sum;

 i=0;
 t_sum=0;
 while (i<10)
 {
```

```
 printf("Partial sum %d is %d\n",i,p_sum[i]);
 t_sum=t_sum+p_sum[i];
 i=i+1;
 }
 printf("Total sum is %d\n",t_sum);
}
```

STEP 3. Create the executable and run it

These operations will depend on how your workstation is connected to the Sequent, as in Tutorial 1.

$rlogin qt

After having logged in on the Sequent, give the following commands,

$cc -c filename.c	Compile the file specifying node function definitions.
$make	This will create the executable 'rsum'.
$./rsum	Run the executable. The output displayed should be that shown below,

```
Partial sum 0 is 45
Partial sum 1 is 145
Partial sum 2 is 245
Partial sum 3 is 345
Partial sum 4 is 445
Partial sum 5 is 545
Partial sum 6 is 645
Partial sum 7 is 745
Partial sum 8 is 845
Partial sum 9 is 945
Total sum is 4950
```