A Novel Visual Hardware Behavioral Language

Xueqin Li
Dept of Electrical Engineering
Utah State University
Logan UT 84322
Email:xueqin@slow.cs.usu.edu

H. D. Cheng
Dept of Computer Science
Utah State University
Logan UT 84322

Abstract - **Most hardware behavioral languages just use texts to describe the behavior of the desired hardware design, but it is inconvenient for VLSI designers who enjoy using schematic approach. The proposed visual hardware behavioral language has the ability to graphically express design information using visual parallel models (blocks), visual sequential models (processes) and visual data flow graph (which consists of primitive operational icons, control icons and Data and Synchro links).**

Thus the proposed visual hardware behavioral language not only can specify hardware concurrent and sequential functionality but also can visually expose parallelism, sequentiality and disjointness (mutually exclusive operations) for the hardware designers. That would make the hardware designers capture the design ideas easily and explicitly using this visual hardware behavioral language.

1 Introduction

The success of hardware design is heavily dependent on how effectively the input language captures the ideas of the designer in a simple and understandable way. A hardware description is behavioral when it is expressed in a manner which conveys only what the hardware module is supposed to do without committing to an implementation. The hardware behavior languages are basically the best way we know today to describe what a system looks like at a more abstract level. Many hardware behavioral languages have been proposed and used in both academical and industrial environment. For example, VHDL, Verilog, Esim, etc. Some HDLs provide a mechanism for attaching code written in Fortran, Lisp, Pascal or C in order to express complex behavior such as HardwareC[10]. But VLSI designers prefer drawing diagrams rather than writing codes in design hardware. If the hardware behavioral language can provide direct, manipulatable graphic models, which are consistent and complete from the hardware designer's point of view, then simulation and synthesis based on these visualized models will greatly improve user interface, and the users only need to understand the visual models. This requires that the hardware behavioral language has the ability to graphically express abstract design information. The proposed hardware behavioral language is such a visual hardware behavioral language that can not only specify hardware concurrent and sequential functionality but can also visually expose parallelism, sequentiality and disjointness (mutually exclusive operations).

There are two kinds of visual models in the proposed language: parallel model (block) and sequential model (process). Using blocks, we are describing the hardware modules as collections of interconnected objects which operate in parallel. Using processes, we are

Figure 1: The visual representations for the port

encapsulating certain aspects of the behavior which can be described in the form of a sequence of steps and expressing these steps in a visual pictorial dataflow style. Our visual behavioral HDL has its own hardware semantics and visual representations (such as process, block, signal link, port, wait operation icon, drive operation icon, etc) with adoption of some data flow syntax and visual representations from other visual language[4, 5, 6] to support the specification of the hardware design.

2 Definitions and Visual Representations

1. port: A connected point associated with objects which may appear in blocks and processes, and which represents information flow into or out of the objects. The attributes of a port are name, type, width, mode and sensitive token. The port name represents its own ID which may be a character or a letter and digit string. The type represents the type of information to be carried through the port which may be a type of bit, integer, float, character or string. The width indicates an array of units of information, and the mode represents the direction of data flow into or out of the object which may be IN, signifying information flow from the "outside" to the "inside", OUT, signifying flow from the "inside" to the "outside", or INOUT, signifying bidirectional information flow. The sensitive token has only two values 0 or 1 which indicates whether the port is a sensitive port to the attached process. The visual representation of the port is shown in Figure 1. During editing, a double-click on these port icons would enter the port editor as shown in Figure 2.

2. Signal link: A signal link connects two ports together. A signal link is represented as a line in the block editor. Signal link has the attributes of type and width which must be compatible with the ports to which they are connected.

3. Block (parallel model): A form of description managed by the block editor, which represents description as a collection of interconnected objects which are considered to operate in parallel. A block has an external port list and contains objects and signal links. An object may be: 1) an instance of another block; 2) a process. The ports of the block and ports on the contained objects may be interconnected with signal links. The visual representation for the block is an empty rectangle with some ports sticking with it which is shown in Figure 3.

4. Process (sequential model): A form of description expressed in a dataflow style. A

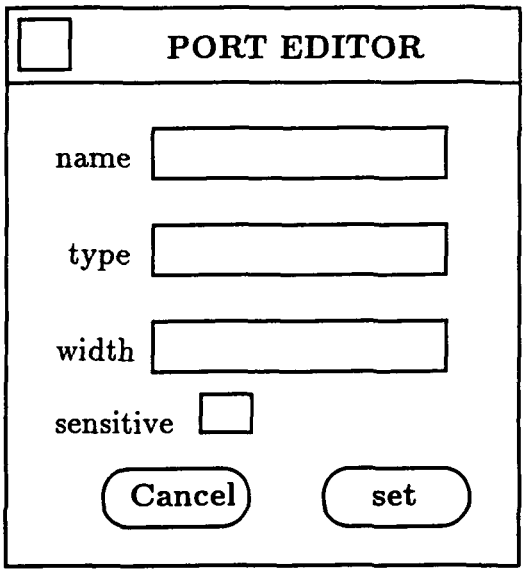

Figure 2: The port editor

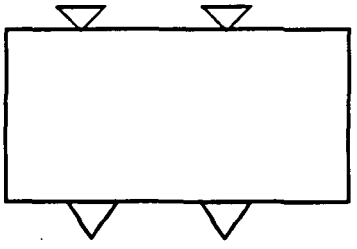

Figure 3: The visual representations for the block

Figure 4: The visual representation for the process

process has an external port list and contains data flow primitive operation icons which are linked by the data/synchro links to express certain behavior which is supposed to do within the process. A process has some sensitive ports which are indicated by tokens. A process is said to be sensitive to a particular set of its IN or INOUT ports at any time and such that execution will be resumed by the simulator whenever there is a state change at any currently sensitive ports of the process. A process which suspends when completes its execution (reaching the end of its dataflow graph) will be deemed at that time to be sensitive to all of its IN and INOUT ports and will be restarted at the beginning of its dataflow graph upon a state change on any of these ports. We assume that a process does not contain subprocesses. The visual representation for the process is a hexagon with some ports sticking with its edges as shown in the Figure 4.

5. Dataflow graph: Some ordered operational icons connected by data and synchro links represent certain algorithms. The visual data flow graph represents the algorithm as a set of operational icons connected by Data links and Synchro links. Consider a process which has the function: $F = X_1X_2 + X_3$ after 20ns. A process showing the dataflow model of this function is shown in Figure 5. The process will be activated whenever any one input variable of the input list: (X_1, X_2, X_3) changes. Once the process is activated, the "and" operation will be executed before the "or" operation because of its data-dependence. The "drive" operation is to model the time of function. That is, "F" will be updated after 20ns from the time the process is activated. The operational icons connected by Datalinks are guaranteed to execute in a serial order. The Synchro link allows the designer to specify control dependencies among data-independent icons. In Figure 6, the computation $A_1 + A_2$ must be carried out before the computation $A_3 + A_4$. The multiplication must be carried out after two additions because of its data-dependence.

   Following elements are needed to specify hardware functional behavior within the data flow graph.

   - Primitive icons: primitive icons visually represent arithmetic, relational or logical operations. These icons have at most two inputs (terminals) and one output (root) as shown in the Figure 7. This form corresponds closely to the operations that can be carried out directly in the hardware. Table 5 gives the definition of these primitive operations.

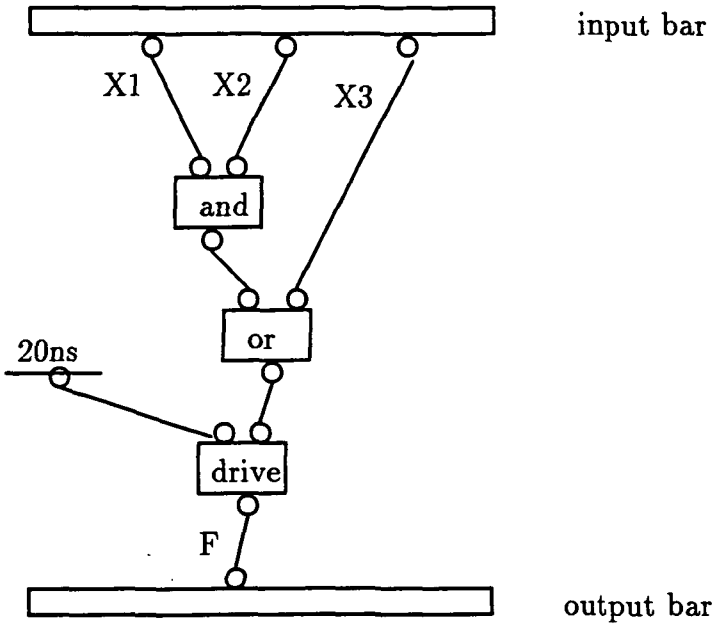

Figure 5: An example of dataflow graph to model a combinational logic

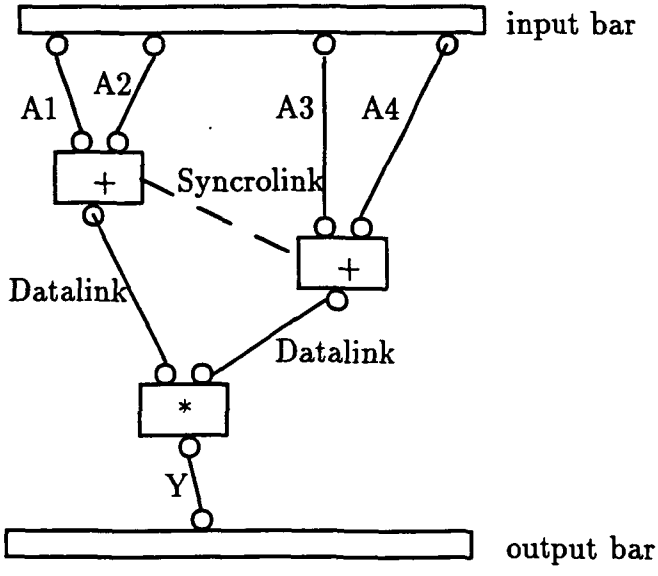

Figure 6: The visual representation for the Datalink and Synchro link

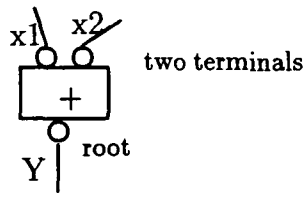

Figure 7: The example of primitive icon

Table 1: The definition of the primitive operations

Group	Symbol	Function
arithmetic	+	addition
(binary)	-	subtraction
	*	multiplication
	/	division
	**	exponentiation
	mod	modulus
	rr	rotate right
	rl	rotate left
	>>	shift right
	<<	shift left
	drive	signal assignment
arithmetic	+1	unary plus
(unary)	-1	unary minus
	abs	absolute value
relational	==	equal
	!=	nor equal
	<	less than
	>	greater than
	≤	less than or equal
	≥	greater than or equal
logical	and	logical and
	or	logical or
	nand	complement of and
	nor	complement of or
	xor	logical exclusive-or
	not	complement
input	ask	read data
output	show	display data

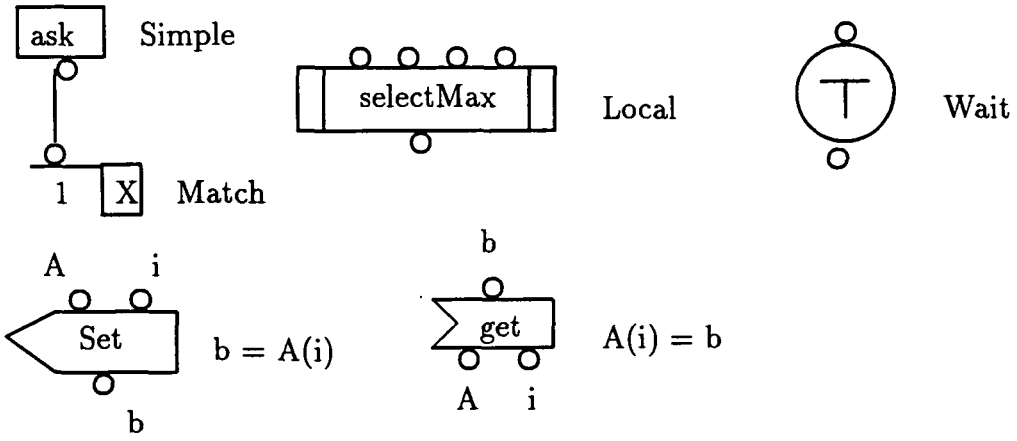

Figure 8: The Operation icons

- Operation icons: There are several operation icons: **Simple, Match, Wait, Constant, Set, Get, Local.** Figure 8 shows these operational icons. An operation when first created is Simple. After its name has been typed and the Return Key pressed, the interpreter parses the name and determines whether it refers to a primitive or a user-defined operation, altering the appearance of the operational icon accordingly. A constant icon has a value and a single root. When the operation executes, this value is made available on its root. When a match operation executes, its value is compared to the value flowing into its terminal. The match operation may be 1 or 0 with the success or failure of the comparison. Wait operation causes the process to suspend itself until the event being waited occurs or the time it is waiting for has elapsed. Wait operation provides strong synchronization across processes. Get and Set operation are used for the variable reference. Local operation is an encapsulation of a body of codes into a single icon. We can treat each local icon as a closed box that communicates with the rest of the program within the process only through its inputs and outputs. We can alter or rearrange the statements inside the box at will, provided the changes do not affect the local icon's input or output. In this way, we can define a hierarchy of procedure call within a process. Consider a simple example: $F = (a+b+c+d+x+y+z)*s = [(a+b+c+d)+x+y+z]*s$. In Figure 9, we make the sum of 4 operants into one local icon **sum4**.
- Control icons: Control icons are attached to the operation icons. Control icon is visually a small icon attached to the right side of an operation icon and is used for implementing the condition and loop functions. A control has two aspects: the action to be taken, and whether this action is taken on success or failure of the operation. Control icons depict these two aspects as follows:
  - A check mark (V) within the control icon indicates that it is activated on success of the operation.
  - A mark (X) within the control icon indicates that it is activated on failure of

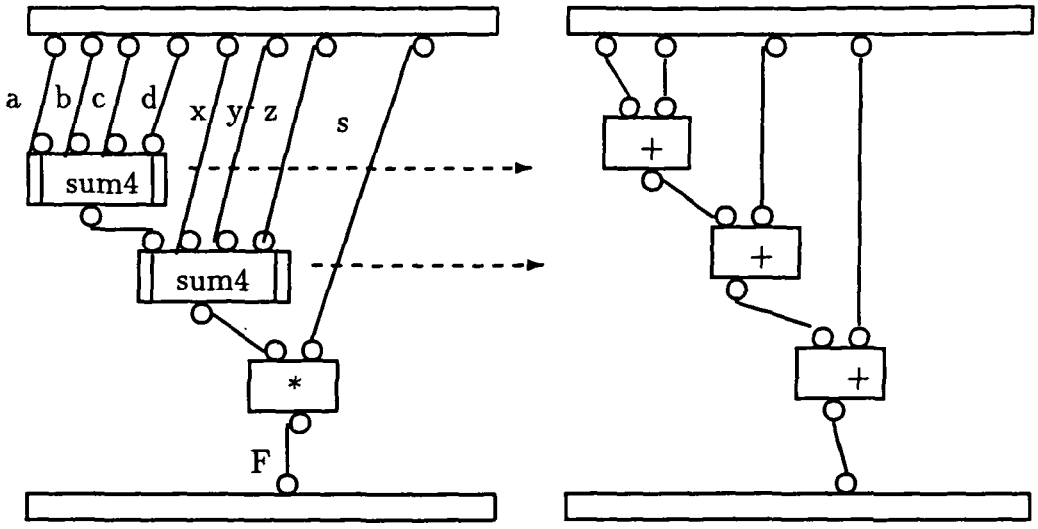

Figure 9: The example of the local icon

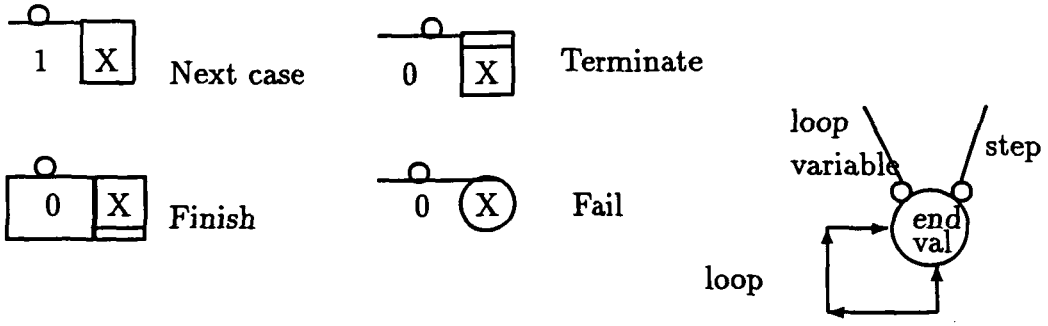

Figure 10: The control icons

the operation.

- The other graphics within the control icons indicate the action to be taken.

We use following types of controls: Next case, Terminate, Finish, Fail and loop as shown Figure 10.

Consider an example of a Mod-4 up counter. The counter counts up if the control input **enable** equals 1 at the rising edge of the clock, **clk**, signal. Figure 11 shows the behavioral representation of this counter. A method with two cases is required to express the counter. The first case is to be activated whenever **clk** and **enable** equal to 1. A primitive operation of the form **(1 + count) mod 4** is to be executed revealing the new value of count which will be scheduled after **clk-width**. The second case is activated if either **clk** or **enable** equal to 0. In this case, the counter will retain its current value.

6. Template library: An external collection of the blocks, which consist of some generic components such as: ALU, counters, comparators, registers and interconnect units

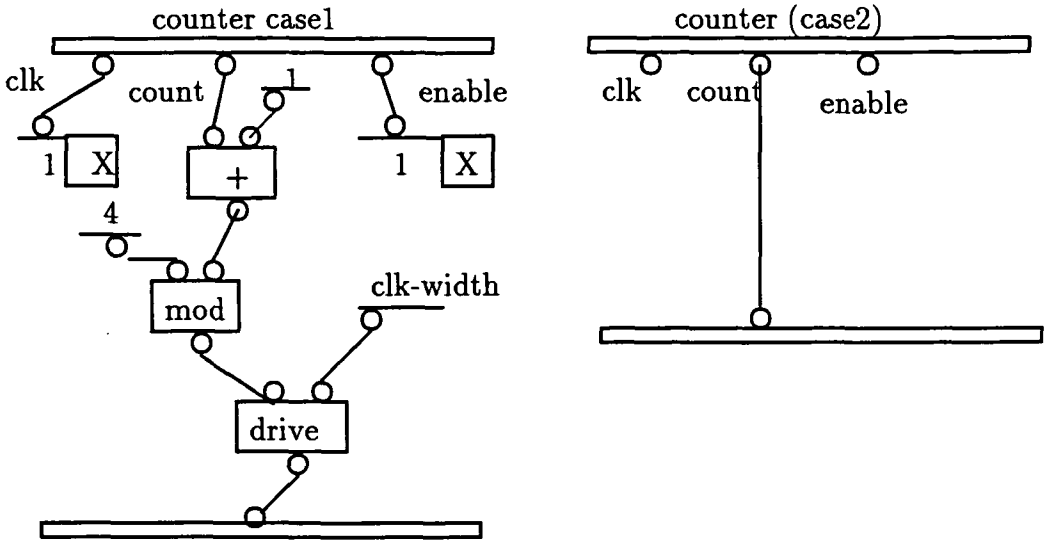

Figure 11: An example 4 mod up counter

(bus, multiplexors), etc. Each process in the template library has a certain function associated with it.

3 Visual Editors for the Visual Hardware Behavioral Language

In our proposed visual hardware behavioral language, the block is the most top model which may contain some processes and the instances of other blocks. When one is working in the block editor, a new model is required. The block editor is shown in Figure 12. In this editor, a mouse is used to direct a cursor about the screen, to control selections from drawing modes, editing modes and a set of pop-up menus, and to select and position design objects on the screen. Pop-up menus and overlapping windows are created on the fly as needed to select items for creation and to provide access to the file system. Drawing a rectangle or hexagon within the editing area creates an anonymous ("unbound") model instance, the model instance can be given a name, and ports can be added and annotated (given names, types, widths and modes). If we type the name inside the model instance area, a search can take place for a block or a process instance created before with the same name and the model instance is bounded if a match is found. Otherwise the model instance remains unbound. If the new model instance is a block and unbound, a double click on this instance can create a new block of the same name and enter the block editor. If the new model instance is a process and unbound, a double click on this unbounded process would enter the dataflow editor which is shown in Figure 13. A double click on a bounded model instance would enter the editor corresponding to the model that the instance is bounded to.

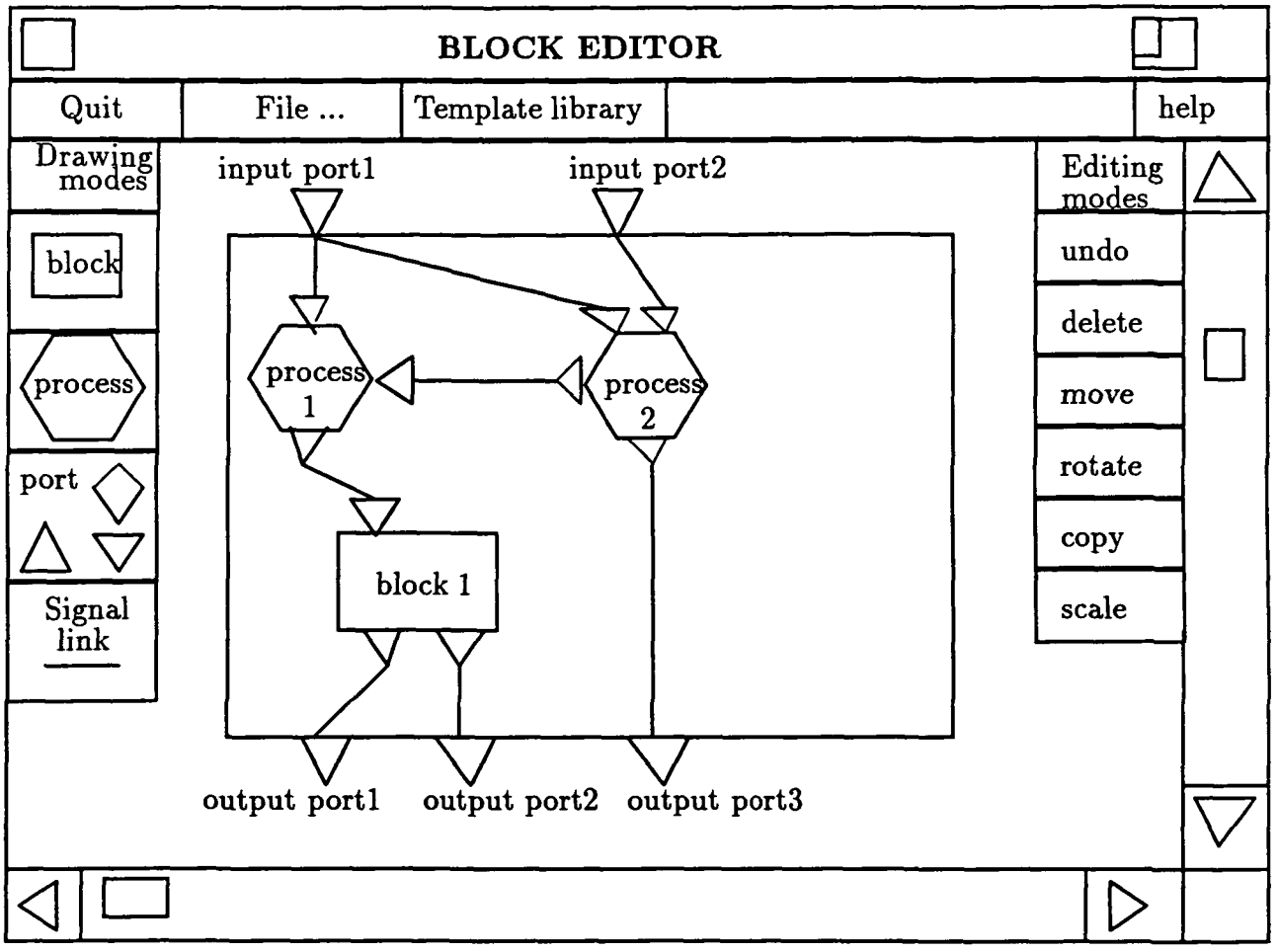

Figure 12: The Block Editor

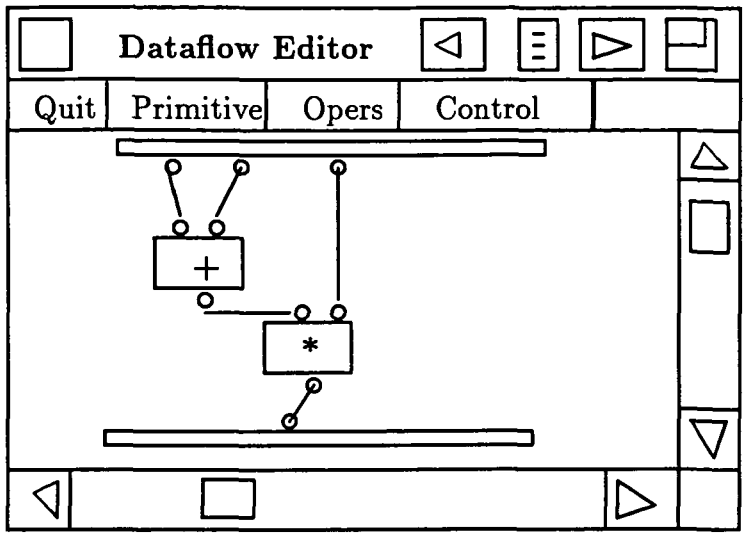

Figure 13: The data flow editor

4 Conclusions

The major features of the proposed visual hardware behavioral language are as follows:

- The ability to design abstract information graphically.

  It can graphically display parallelism, sequentiality and disjointness using parallel model (visual blocks), sequential model (visual processes) and data flow graph on the screen.

- Two level behavioral descriptions.

  At the architectural level, we use blocks and cooperating processes to describe the communication and connected behavior of the processes within the design. At the functional level, we use data flow graph to describe the algorithm accomplishing certain functions.

- Data flow style.

- Easy to learn and use.

  Because the proposed visual hardware behavioral language provides graphical information such as diagrams (rectangles for blocks and circles for processes) and operational icons in the actual process of hardware design, it is easy to learn and use for VLSI designers.

References

[1] A Rountable, "Behavioral description languages - PART I: Are designers Benefiting?". IEEE Design Test of Computers, February 1990, pp56 - 62.

[2] R. Camposano, L. F. Saunders and R. M. Tabet, "VHDL as Input for High - Level Synthesis" IEEE design Test of Computers, March 1991, pp43 - 49.

[3] D. Ku and G. De Micheli, "Hardware C: A Language for Hardware Design" tech, rpt. CSL-TR 90- 419, Computer System Lab., Stanford University, August 1990(version 2.0).

[4] "Prograph Tutorial" The Gunakara Sun Systems Limited, 1990.

[5] "Prograph Reference" The Gunakara Sun Systems Limited, 1990.

[6] Shi-kuo Chang, "Principles of Visual Programming Systems" Prentice-Hall. Inc, 1990.

[7] Alex Orailoglu and Daniel Gajski, "Flow Graph representation" IEEE 23rd Design Automation Conference, 1986, pp503 -509.

[8] Akira Sugimoto, "VEGA: A Visual Modeling Language for Digital Systems" IEEE Design Test of Computers, June,1991, pp38 - 45.

[9] Paul J. Dronggowski, Juahar R. Bammi and Tsu-Hua, Wang,"A graphical hardware Design Language", IEEE 25th Design Automotion Conference, 1988, pp108-144.