RAPID: A Tool for Hardware/Software Tradeoff Analysis

*Nicholas L. Rethman** and *Philip A. Wilsey*

Computer Architecture Design Laboratory
University of Cincinnati, Cincinnati, OH 45221–0030
phil.wilsey@uc.edu (513) 556–4779

Abstract

This paper presents RAPID, a design system that combines graphics and text to support the hardware/software design process. The graphical portion of RAPID allows the designer to represent system components as either rectangular objects or as graphical objects (constructed using **xfig**) implying some behavioral functionality. Each component can have up to three (equivalent) descriptions: a structural decomposition of its internal components, a hardware description in VHDL, or a program representation in Ada. The system modeler can configure the system with any set of hardware/software bindings. When the selection for the configuration is complete, RAPID drops the software components from the display the modeler then reconnects the remaining hardware components with an embedded processor. Finally, the Ada programs are compiled, RAPID instantiates the proper VHDL design entity/architecture pairs (which must be compiled) and the entire system can then be simulated.

1 Introduction

This paper will describe a design tool called RAPID that supports the designer in building and evaluating systems that contain both hardware and software components. RAPID is a single integrated system incorporating graphics and text for design representation. Each system component can have both a hardware and software definitions and the complete system can be configured, evaluated, and reconfigured in various hardware/software organizations. Such flexibility allows the designer to explore many design alternatives in order to achieve a more cost effective system.

RAPID is tightly integrated with the hardware description language VHDL [2] and loosely integrated with the programming language Ada [1]. Components are strongly typed and type conflict occurs through a type resolution file. Finally, the graphical display uses the X-windows libraries.

A system designed in RAPID is composed of *components*, *modules*, and *objects*. A component represents one element at one level of the hierarchical design and is graphical shown as a rectangle. Modules and objects are terms used to denote, respectively, representations of software components and hardware components. Initially components are drawn as rectangular boxes, however, RAPID allows the designer to draw an arbitrary symbol to represent a component. This symbol is created

*Nicholas Rethman is now with the SEG/CAD/CVO Group, Digital Equipment Corp., Hudson, MA 01749.

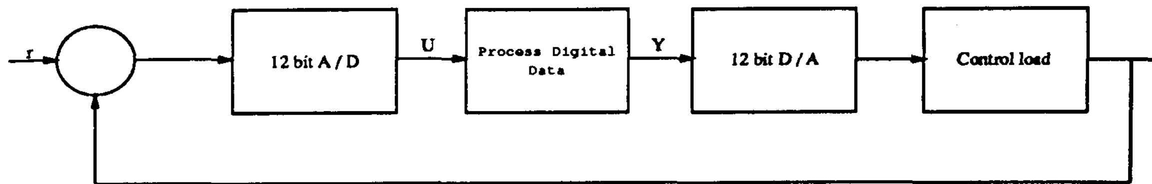

Figure 1: Example control system.

using the public domain program `xfig`. The user can then choose to display either the symbolic or rectangular form of the component.

Each component also has a set of *ports* that define windows for data transmission between its internal an external components. After defining the ports the user may choose to refine a component with a more detailed structural representation, by decomposing the component into smaller more manageable sub–components. Alternatively, the designer may choose to create links that connect the ports of one component to the ports of other components.

If the component is going to be described in hardware, RAPID will automatically generate the corresponding VHDL entity description (derived from the port specifications) and, optionally, create a skeleton behavior model for the corresponding VHDL architecture body. If the component is decomposed into a set of internal components, then RAPID will completely generate the full VHDL entity and architecture bodies to fully represent the component and its instantiated sub–components. If the component is not decomposed into sub–components, then the user must develop the appropriate VHDL behavioral code within the architecture body.

If the component is being designed as a software module then the user must generate an ASCII file that contains the source code for the module[1]. At this time, the user is responsible for ensuring that all of the ports on the component match up with variables in the assembly language source code.

At this point the source code and the VHDL code can be compiled by the user and then simulated to verify the design performs the desired function. If an error is detected the user must modify the design and re–attempt to compile and simulate the source programs.

2 Designing a digital control system with RAPID

A digital control system example was developed to demonstrate RAPID's use in designing and evaluating a hardware/software system. The digital control system designed in this example represents only a small portion of a larger control system. The complete control system can be seen in Figure 1. The only part of the control system from Figure 1 that will be implemented is the block labeled "Process Digital Data".

The "Process Digital Data" component is shown in Figure 2 as consisting of six sub–components. The "12 to 16" and "16 to 12" components are responsible for converting the 12 bit input data into a 16 bit output data and vice versa. This conversion is simply done by respectively multiplying or dividing the input by sixteen. The "Int to Float" and "Float to Int" components perform the task of converting an input data from a 16 bit integer to a 16 bit mantissa with a 4 bit exponent or vice

[1]Ideally we would restrict the description of the software modules to the Ada programming language. However, because of the high cost of Ada cross-compilers, we have chosen to use assembly languages for which simple assemblers can be quickly created.

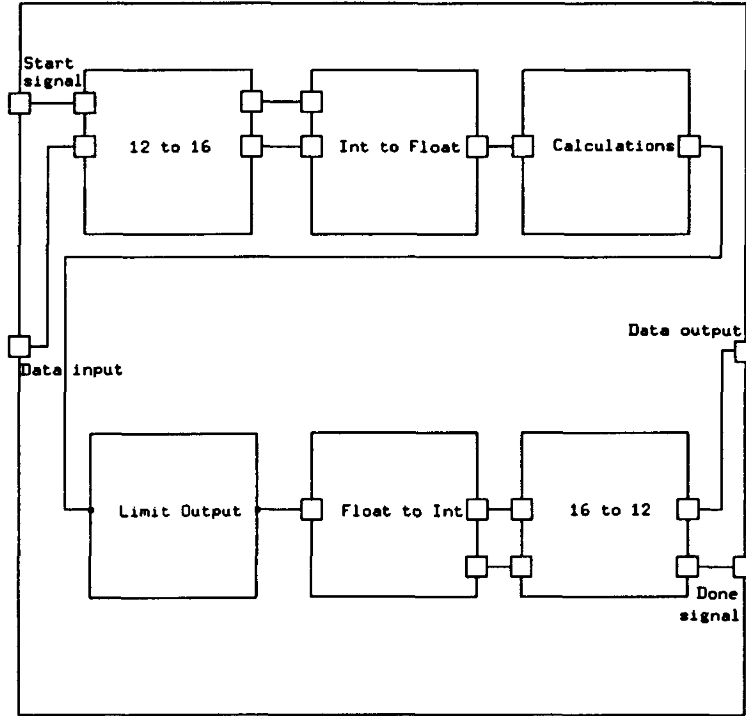

Figure 2: Top level design of "Process Digital Data".

versa. The "Calculation" component performs the following calculation:

$$y_k = \alpha_1 y_{k-1} + \alpha_2 y_{k-2} + \ldots + \alpha_5 y_{k-5} + u_k + \beta_1 u_{k-1} + \ldots + \beta_5 u_{k-5}$$

where α and β are constants, u is the input, y is the output, and k is a unit of time.

The "Limit Output" component is used to ensure that the input to the component falls between a high and low limit. If the input is outside one of these limits then that limit value will be used for the output, otherwise the input value is used. The high limit value used for this example was 1024 and the low limit value was -1024. These two values will limit the output of the system to be in the range of -64 and 64 because the "16 to 12" component will divide the results of the "Limit Output" component by sixteen.

The TMS32020 processor manufactured by Texas Instruments was used to execute all of the software modules in this design [3]. Since no Ada compiler was available that would produce executable code on the TMS32020 processor, the software modules had to be written in the assembly language for this processor. An incomplete VHDL model for the TMS32020 processor has been written, which incorporates all of the features of the processor required for this example.

The TMS32020 is an I/O mapped processor with three available interrupt lines. The type of interface that would be natural for this processor would be to use the interrupts to signal when the processor should start executing a certain function and to use the ports for inputting and outputting data. In this example only one interrupt line is needed along with three input ports and two output ports.

In order to demonstrate RAPID prototyping capabilities, we selected three different hardware/software configurations for implementing the control system. The first configuration binds all

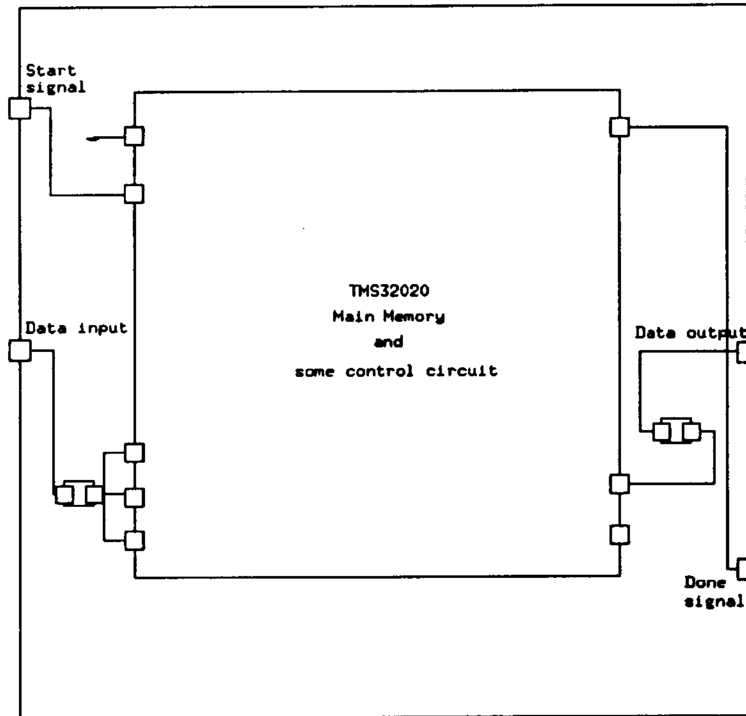

Figure 3: Design of control system with all six components in software.

of the components to software modules that are executed on a TMS32020 processor. In the second examples, the data conversion components are configured as hardware objects with the remaining components configured as software modules. Finally, in the third example, the limit component is configured as a hardware object.

2.1 All Components to Software Modules

In the first configuration all components are realized as software modules. The function of each components was written in the TMS32020 assembly language. After all six components were marked as software modules, the configure command was given to RAPID. The TMS32020 processor was then connected to the control system as shown in Figure 3.

In order to fully test this configuration, nine different simulations were run. The time required for each component and each interface between the components to complete is shown in Table 1. All of the times listed in Table 1 are measured in nano–seconds and were taken from the results of a VHDL simulator.

2.2 Configuring the Data Conversion Components as Hardware Objects

From the performance results of Table 1, we see that the software module for the "12 to 16" conversion function required 600ns to be executed on the processor. This software module is a good candidate to be designed in hardware because all the software module does is to shift its input by four places. The only thing that needs to be done to implement this component in hardware is to ground the lower four bits of its output and to shift all the inputs bit four places. The time required

	Inputs								
	65	64	63	1	0	-1	-63	-64	-65
Interface	1200	1200	1200	1200	1200	1200	1200	1200	1200
12 to 16	600	600	600	600	600	600	600	600	600
Interface	400	400	400	400	400	400	400	400	400
Int to Float	5600	5600	6000	8000	2200	8400	6000	6000	5600
Interface	400	400	400	400	400	400	400	400	400
Calculations	67400	67400	67800	69800	64000	70200	67800	67800	67400
Interface	400	400	400	400	400	400	400	400	400
Limit Output	5000	5000	8600	8600	8600	8600	8600	8600	8200
Interface	400	400	400	400	400	400	400	400	400
Float to Int	2000	2000	2000	2000	2000	2000	2000	2000	2000
Interface	400	400	400	400	400	400	400	400	400
16 to 12	1400	1400	1400	1400	1400	1400	1400	1400	1400
Interface	400	400	400	400	400	400	400	400	400
Total Times	105600	105600	110000	114000	102400	114800	110000	110000	108800

Table 1: Timing results for the first configuration. (Given in nano–seconds)

to execute this component in hardware is negligible because the input is connected directly to the output.

Similarly, the "12 to 16" component can be configured as a hardware module. In this case we will lose the four least significant bits of the inputs. Once these two components are configured as hardware, the system is viewed as shown in Figure 4.

For this configuration, the software for the TMS32020 processor had to be modified slightly in order to accommodate the two components being replaced with hardware. One change was the memory location at which the data input value was stored. Instead of this value being stored in a temporary memory location the value was stored where the "12 to 16" function would have stored its results. Also the output command was modified to output the results of the "Int to Float" component instead of the "16 to 12" component. The last change was to remove the calls to the two functions that had been replaced by hardware.

The same input values that were used to simulate the first configuration were again used to simulate the second configuration. The timing results from these simulations are shown in Table 2. The first and last interfaces are no longer shown in the table because the interface between the control system and the "12 to 16" component and the interface between the "16 to 12" component and the control system are no longer present. The time required for the "12 to 16" and "16 to 12" components would be the delay time that the wire would introduce and for this example we considered this time to be 0ns.

When comparing Table 1 and Table 2 the reader will notice that for each input of the control system, the second configuration operated 2,800ns faster than the first configuration. Since the only thing added to the second configuration was a few wires, it would be strongly recommended to use the second configuration over the first. But could we perhaps even do better in performance without adding much cost to the completed system? By examining the Tables we see that the "Limit Output" function took anywhere from 5,000ns to 8,600ns to execute in software. What if this function was in hardware? How much faster would the system be? Would interfacing the software to the hardware and then back to software outweigh the increased speed of the hardware designed "Limit Output" component?

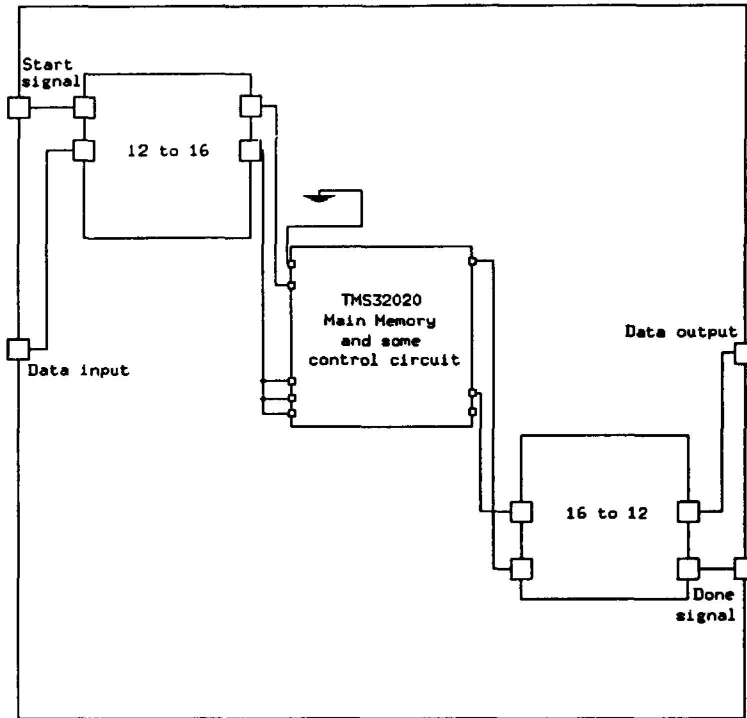

Figure 4: Finished design of the second configuration.

	Inputs								
	65	64	63	1	0	-1	-63	-64	-65
Interface	—	—	—	—	—	—	—	—	—
12 to 16	0	0	0	0	0	0	0	0	0
Interface	1200	1200	1200	1200	1200	1200	1200	1200	1200
Int to Float	5600	5600	6000	8000	2200	8400	6000	6000	5600
Interface	400	400	400	400	400	400	400	400	400
Calculations	67400	67400	67800	69800	64000	70200	67800	67800	67400
Interface	400	400	400	400	400	400	400	400	400
Limit Output	5000	5000	8600	8600	8600	8600	8600	8600	8200
Interface	400	400	400	400	400	400	400	400	400
Float to Int	2000	2000	2000	2000	2000	2000	2000	2000	2000
Interface	400	400	400	400	400	400	400	400	400
16 to 12	0	0	0	0	0	0	0	0	0
Interface	—	—	—	—	—	—	—	—	—
Total Times	102800	102800	107200	111200	99600	112000	107200	107200	106000

Table 2: Timing results for the second configuration. (Given in nano–seconds)

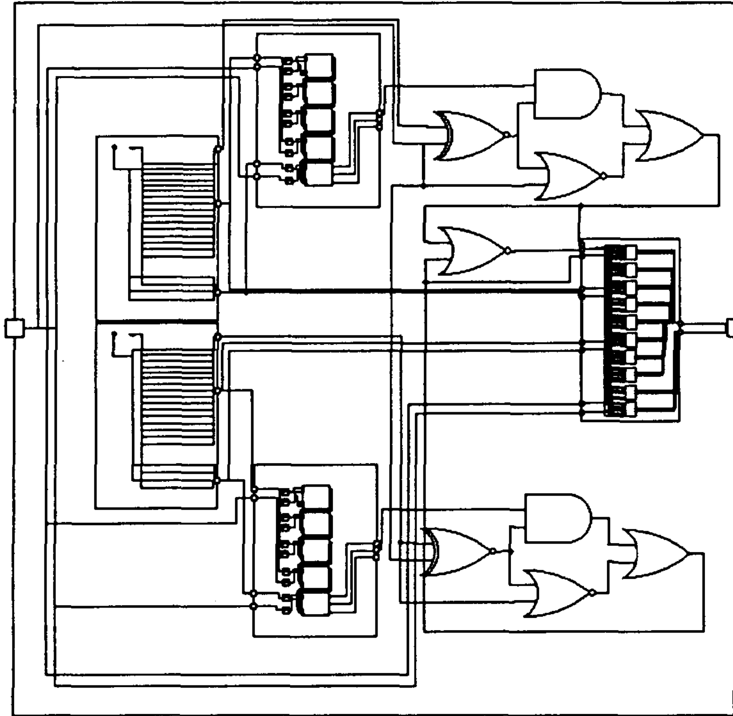

Figure 5: Completed hardware design of the "Limit Output" component.

2.3 Configuring the Limit function as a Hardware Object

To answer these questions the "Limit Output" function was designed using TTL components and can be seen in Figure 5. The hardware version of this component has its limit values specified by the setting of some dip switches, where as the software version of the component has its limit values specified in data memory. To change the limit values in software all one has to do is to change the value at a specified memory location. This operation could be done by another software module while the system was running. In order to change the dip switches in the hardware version, the machine will more than likely have to be shut down. This means that the hardware version of the "Limit Output" component will be less flexible than its software counterpart.

At this point the "Limit Output" component was marked as hardware and the system was reconfigured. The result can be seen in Figure 6. The software had to again be modified to accommodate this new configuration. Instead of calling the limit function right after the completion on the "Calculation" component as was previously done, the results of the "Calculation" component were sent to two different output ports. Since it was known that the hardware version of the limit function could in worst case produce a valid output after 88ns, one no–op instruction was used to allow enough time to pass before the results of the limit function were read back into the processor. The 200ns required for the no–op instruction would assure us that valid data would be available on the inputs once the instruction had completed its execution.

The same simulations were again run for this configuration and the timing results can be seen in Table 3. The timing results reported for the "Limit Output" component is the time that it took the hardware to have produce a valid output. The actual time required for the system to move beyond the "Limit Output" component was 200ns because of the no–op instruction in the software.

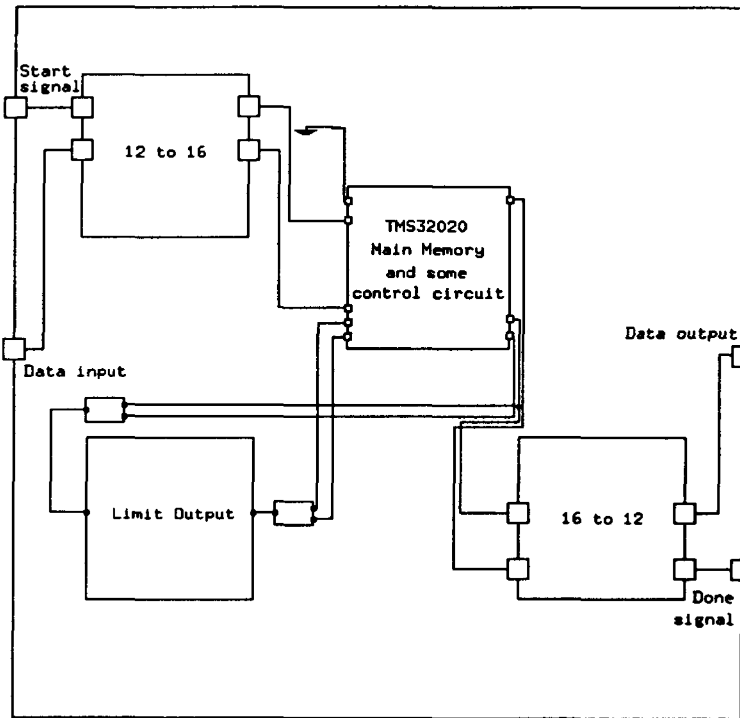

Figure 6: Finished design of the third configuration.

	Inputs								
	65	64	63	1	0	-1	-63	-64	-65
Interface	—	—	—	—	—	—	—	—	—
12 to 16	0	0	0	0	0	0	0	0	0
Interface	1200	1200	1200	1200	1200	1200	1200	1200	1200
Int to Float	5600	5600	6000	8000	2200	8400	6000	6000	5600
Interface	400	400	400	400	400	400	400	400	400
Calculations	67400	67400	67800	69800	64000	70200	67800	67800	67400
Interface	400	400	400	400	400	400	400	400	400
Limit Output	44	19	19	12	0	12	12	12	12
Interface	800	800	800	800	800	800	800	800	800
Float to Int	2000	2000	2000	2000	2000	2000	2000	2000	2000
Interface	400	400	400	400	400	400	400	400	400
16 to 12	0	0	0	0	0	0	0	0	0
Interface	—	—	—	—	—	—	—	—	—
Total Times	98400	98400	99200	103200	91600	104000	99200	99200	98400

Table 3: Timing results for the third configuration. (Given in nano-seconds)

It is interesting to notice that only 400ns were added to the time required for the interface that went from hardware back to software. One reason this time along with the time to interface the software to the hardware is so small is because no interrupt handling had to be done at these interfaces. This saved 800ns in the execution of the control system. The interrupt driven interface would have been necessary if the execution time for the "Limit Output" component could have been long or if it varied over a wide range of times. Since the execution time of the "Limit Output" component was small the no–op instruction could be used to wait for the results from the hardware component.

By comparing Table 2 and Table 3 it can be seen that the average decrease in time for the third configuration is 7155ns. Although unlike the decrease in time between the first and second configuration this decrease in time comes at some non–negligible expense. The cost for this configuration is the added expense for the hardware and the reduced flexibility of setting the limit values. To make a decision regarding which configuration is better we would require additional information regarding what this system is going to be used for and what type of performance requirements are placed on the system.

3 Summary

This example has demonstrated that RAPID can be used to develop systems that incorporate both software and hardware in the same design. This example has also demonstrated that both software and hardware can be simulated together by the same simulator to obtain timing results on how each component performs and on how the components interact with each other. This distinguishing characteristic of simulating combined software and hardware systems assures the designer that his system will function properly before the system is actually built. We have also demonstrated how RAPID can aid the designer in deciding on which components should be developed in software and which ones should be developed in hardware.

References

[1] ANSI. Reference manual for Ada programming language. Technical Report ANSI/MIL–STD–1815A–1983, American National Standards Institute, New York, 1983.

[2] *IEEE Standard VHDL Language Reference Manual.* New York, NY, 1988.

[3] Texas Instruments, Houston, Texas. *TMS32020 User's Guide*, 1986.