SLAC-PUB-5509
October 1991
(A/I)

FASTBUS Simulation Tools*

T. D. Dean
Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309
M. J. Haney
University of Illinois, 1110 W. Green Street, Urbana, IL 61801

Abstract

A generalized model of a FASTBUS master is presented. The model is used with simulation tools to aid in the specification, design, and production of FASTBUS slave modules. The model provides a mechanism to interact with the electrical schematics and software models to predict performance. The model is written in the IEEE std 1076–1987 hardware description language VHDL. A model of the ATC logic is also presented. VHDL was chosen to provide portability to various platforms and simulation tools. The models, in conjunction with most commercially available simulators, will perform all of the transactions specified in IEEE std 960–1989. The models may be used to study the behavior of electrical schematics and other software models and detect violations of the FASTBUS protocol. For example, a hardware design of a slave module could be studied, protocol violations detected and corrected before committing money to prototype development.

The master model accepts a stream of high level commands from an ASCII file to initiate FASTBUS transactions. The high level command language is based on the FASTBUS standard routines listed in IEEE std 1177–1989. Using this standard-based command language to direct the model of the master, hardware engineers can simulate FASTBUS transactions in the language used by physicists and programmers to operate FASTBUS systems.

I. Introduction

We have developed tools that allow the designer of a FASTBUS [1] module to simulate electronic design schematics with high level language commands of the type used by the programmers, engineers, and physicists to interact with real FASTBUS modules. These tools aid in determining that a design complies with the FASTBUS specification. Moreover, the code developed by the designer to exercise the design can be used as an effective starting point for developing the code that will be used to test and operate the fabricated design.

A. Conventional Tools

Modern CAE workstations have a variety of software packages to aid in the design and manufacture of FASTBUS modules and other electrical circuits. Many complex system designs begin at a high level of abstraction where the function of the system elements are fairly well known, but their implementation is not. High level modeling languages (e.g., Simula, Modula–3, Modsim) can be used to analyze the abstract function of a circuit board before physical design begins.

Thus the most difficult aspect of electrical system design today is the synthesis of a behavioral model into an electrical schematic. To address this problem, hardware description languages (most notably VHDL [2] and Verilog [3]) have become common approaches to behavioral modeling. Models written in a hardware description language may be as abstract as those written in a pure simulation language; they can also be as specific as an electrical schematic. This spectrum of application from behavioral to structural allows hardware description languages to describe, simulate, and represent designs continuously throughout the design process. This in turn eases the transition from abstract model to physical implementation.

To effectively simulate a behavioral model of a FASTBUS board being designed, it is necessary to simulate (or at least approximate) the environment that the board will be used in, i.e., a "virtual" FASTBUS crate. This is the motivation for the VHDL models developed by this work. By providing a simple FASTBUS master and slave, electrical schematics of modules under design can be tested against virtual modules in a virtual crate.

These same ideas were explored by Willwerth et al. [4,5] using the nonstandard DABL language available from Daisy Systems Corporation (now Dazix) [6]. The models described below, however, were developed in VHDL so as to be portable between CAE systems, and were constructed with the FASTBUS standard routines in mind as a language for control.

Traditional simulation packages require that the design testing be done in terms of specifying individual signals at specific times. The simulation proceeds in a fixed, predefined time sequence. Using the native language of most simulators limits the transaction to a fixed time step transaction. This does not allow for variable response of the addressed slave design. A CSR space read might look like the example in Fig. 1.

This is substantially different from the procedures used by programmers, engineers, and physicists who use the module. In actual use, it is more typical to employ high level languages such as FORTRAN or C, and direct the actions of the FASTBUS modules using standard routines [7]. In the FASTBUS environment, the timing of the actions is not fixed by an artificial simulation, but

* Work supported by Department of Energy contracts DE-AC03-76SF00515 (SLAC) and DE-AC02-76ER01195 (University of Illinois).

Presented at the IEEE Nuclear Science Symposium and Medical Imaging Conference, Santa Fe, NM, November 5–9, 1991.

```
@t=100ns    AD=<address>
@t=110ns    AS=1                (connect)
@t=200ns    MS=2                (set up NTA)
@t=200ns    RD=0
@t=200ns    AD=<sec.address>
@t=210ns    DS=1                (write NTA)
@t=320ns    DS=0
@t=400ns    RD=1
@t=420ns    DS=1                (read data)
@t=520ns    DS=0
@t=530ns    AS=0                (disconnect)
```

Fig. 1. Traditional simulation

are determined by the physical behavior of the modules. The FASTBUS models developed in this work follow this more natural pattern of operation; the designer specifies the FASTBUS cycles to be performed in a language based on the FASTBUS standard routines, and the timing of the simulation is dictated by the behavior of the models, not by an absolute specification of signal transition times.

B. VHDL

VHDL (VHSIC hardware description language), an IEEE standard [8], was originally developed on behalf of the Department of Defense to represent (document) electrical systems delivered to the government [9]. Currently a mandate for deliverables to the DOD, most CAE vendors are providing (or promising) VHDL support in their products. Thus VHDL is a suitable choice for developing models to be portable to various platforms.

Three VHDL models were developed to assist in the design of FASTBUS modules: a virtual master (VIM), a virtual slave (VIS), and the arbitration and broadcast timing controller (ATC) that services a virtual crate. The designer must provide a high level module that connects either a VIM or a VIS (or both) and the ATC to the module being designed. In many CAE systems, this would involve a single electrical schematic drawing to connect the models.

If the module being designed is a FASTBUS master, then the VIS model would be used as a basis for exercising the master. Causing the master being designed to perform the desired FASTBUS cycles is the responsibility of the designer.

If the module being designed is a FASTBUS slave, then the VIM model would be used to exercise the slave. In this case, the FASTBUS operations performed by the VIM on the slave are provided by an ASCII character control file (easily manipulated by any text editor) containing the FASTBUS cycles and arguments.

Finally, the VIM, VIS, and ATC may be used together as a teaching tool to aid in understanding FASTBUS transactions.

The VHDL models for the VIM, VIS, and ATC are discussed in the next section; the basic input language to the VIM is described in the section after. Following these, a highly portable language translation tool is described that transforms FASTBUS standard-like subroutine calls into the basic input for VIM.

II. VIM, VIS, AND THE ATC

The virtual master (VIM) is a VHDL model of a microsequencer state machine and a bus driver. The VIM accepts a stream of IEEE 1177-like low level commands (e.g., FB_CYCLE_PA_CSR to send a primary address cycle to connect to the CSR space of a module). These commands are read from a standard ASCII text file, one command (or data/argument) per line. This allows easy editing and manipulation of the command sequences using any text editor. The model is approximately 2100 lines of VHDL code, and is capable of performing all of the action routines described in Section 8 ("Primitive FASTBUS Action Routines") of IEEE std 1177–1989 [7]. From these, and a few "housekeeping" functions (e.g., TITLE and COMMENT), complex FASTBUS cycles can be composed. Although this command set is limited, the use of a high-level language translator (described below) can readily allow highly sophisticated control programs to direct the VIM.

The virtual slave (VIS) is a VHDL model of a simple memory-like slave device. The VIS can be addressed geographically only (logical addressing will be supported in a future version), in either data or CSR space. Attempts at broadcast addressing of the VIS result in messages printed during simulation indicating that broadcasts are not supported, and the VIS ignores the broadcast. Data space consists of 11 read-write locations (starting at 0), and 11 CSR registers (starting at 0) are provided. The CSR registers are simple read-write locations, and do not perform any special function (contrary to the FASTBUS specification for real modules). Only single-word transfers are supported; all other transactions result in simulation-time error messages. The VIS is approximately 350 lines of VHDL code.

The ancillary timing controller (ATC) is a VHDL model that provides the AK and DK handshake signals during a broadcast operation. Although the VIS does not support broadcast, the VIM requires these signals from the ATC when broadcast operations are performed. The ATC is approximately 150 lines of VHDL code.

These models can be connected together with a module under design by any of several approaches. Using textual VHDL, it is easy to define a virtual crate consisting of a VIM (or a VIS), an ATC, and the design module, and connect them all with FASTBUS backplane signals. Alternatively, using a CAE system such as Workview (by Viewlogic [10]), an electrical schematic of the crate can be used to connect the models. This drawing is translated to VHDL by the CAE system for simulation.

One possible view of the design process has the electrical design schematic on the top level of the design set,

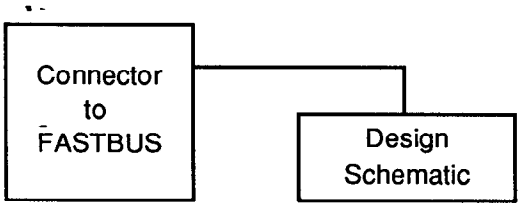

Fig. 2. Top level

as shown in Fig. 2. On most CAE systems, selecting the FASTBUS connector and pushing down one level reveals the environment of the design under simulation, as shown in Fig. 3.

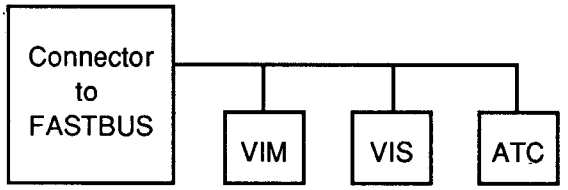

Fig. 3. Lower level

The lower level electrical schematic consists of the connectivity to the top level, and the VHDL models of the VIM, VIS, and ATC, as necessary. The geographical address of the module(s) is assigned on the lower level.

III. Base-level input to VIM

The VIM microsequencer state machine accepts a stream of low-level commands from an ASCII text file to specify the sequence of cycles to be performed. These commands are largely a subset of the FASTBUS standard subroutines defined by IEEE std 1177, with emphasis on the "Primitive Action Routines." These commands allow the VIM and the FASTBUS simulation to be directed in abstract terms, and allow for variable response from the addressed slave.

Due to the nature of the VHDL I/O routines, these commands must be presented one per line, and the arguments to these commands must also stand alone on separate lines. Thus, the previous example of a CSR space read would be coded as shown in Fig. 4.

Timing is not specified in this file. The VIM will initiate address or data cycles, but will proceed only when the addressed slave responds. Thus the slave's responses define the timing of the simulation. This reflects the behavior of a real FASTBUS system, where system timing is not specified by the designer, but arises from the interaction of devices. The designer is relieved of the burden of insuring that the timing of the individual signals is correct.

```
FB_CYCLE_PA_CSR
<address>

FB_CYCLE_WRITE_SA
<sec. address>

FB_CYCLE_READ_WORD

FB_CYCLE_REL_BUS
```

Fig. 4. Primitive commands

IV. High-level translator

The low level input to the VIM is simple, but not convenient. A higher level language was defined, also based on the IEEE std 1177, to allow commands and arguments to be listed in a more comfortable and familiar syntax, as demonstrated in Fig. 5.

```
FB_READ_CSR(     <address>,
                 <address>);
FB_WRITE_CSR(    <address>,
                 <address>
                 <data>        );
```

Fig. 5. High level commands

These examples use a predefined arbitration level and the numeric primary and secondary addresses specified.

```
ARB_LEVEL = 0x000000FE;
pad = 0x11;
sad = 0x1000 + 43;
data = 0xdeadbeef;

FB_WRITE_CSR(pad,sad,data);

FB_READ_CSR(pad,sad)
```

Fig. 6. Variables and expressions

Symbolic variable names and expressions can be used to simplify the coding of the commands. Reserved-name variables are used to control the environment. The example of Fig. 5 may be coded, as shown in Fig. 6.

Translation from the language of Fig. 6 to the low-level input needed by the VIM (Fig. 4) is performed through the use of a parser-compiler, available from the authors. This translator accepts free-form input containing the commands and arguments for the VIM, and restructures them into a line-at-a-time command file. Complex FASTBUS commands are decomposed into sequences of FASTBUS cycles used to direct the VIM for simulation. These can be readily adopted for use in a real system to communicate with the implemented design. Although these are microsequencing masters for FASTBUS, there is no physical

implementation of a "VIM"; thus master-specific changes to the high-level VIM commands may be needed to allow the control of actual FASTBUS masters.

It must be noted that the syntax of the high level language is not precisely that of the IEEE standard FASTBUS routines. Due to the relative simplicity of the VIM model (compared to a real FASTBUS master), many of the standard subroutine arguments are inappropriate. For example, the environment arguments of the standard routines are meaningless in most simulation environments. Thus, the argument lists contain only the arguments necessary to perform the requested simulation operations. With the above considerations about required master-specific changes, this discrepancy was not considered unacceptable. While the argument lists may vary, the FASTBUS subroutine names and functions have been maintained exactly.

V. Portability

A. VHDL

The original version of the VIM, VIS, and ATC models were developed using a VHDL implementation purchased from Viewlogics Systems, Incorporated [10] (a full-spectrum CAE vendor). The code was developed at SLAC according to the guidelines provided by Viewlogics for the creation of "portable" models.

The code was then "ported" to run under a VHDL environment purchased from Intermetrics, Incorporated [11]. Several "incompatibilities" were encountered that may be of concern to other users of this code, or others involved in the creation of "portable" VHDL models. It should be noted that Intermetrics has been involved in the development of the VHDL language since its inception; thus VHDL code that does not port easily to the Intermetrics environment probably will not port easily elsewhere.

The most immediate problem in porting the models was the choice of base logic signals, and the use of file I/O. As a language, VHDL supports user defined logic types which can have user defined logic values, typically "0" and "1" for true and false, "X" for unknown, etc. Simulators from different vendors are optimized for different definitions. Viewlogics has optimized their simulator to support a 4-level logic variable; Intermetrics has not. The controversy over the "best" multilevel logic package is still in progress. For more information on std_logic_1164, contact the VHDL consulting group [12].

Also, although the VHDL as a language supports a simple notion of file input and output, the specific implementations are vendor-specific. File I/O is extremely important to the VIM, both for reading the command stream, and for generating run-time messages for the user, describing VIM operation, error conditions, and protocol violations.

Other harmless, but annoying porting problems occurred because of semantic shortcuts offered by Viewlogic, but not supported by Intermetrics. For example, "prising(AS)" is a Viewlogic conditional to test if AS is changing from 0 to 1. Intermetrics does not support this shortcut; the more formal VHDL syntax is "(not AS'stable) and (AS='1')". For the sake of portability, these shortcuts had to be identified and eliminated.

B. Unix

The high level translator was developed using the standard *lex* parser generator and *yacc* compiler compiler [14,15] and the standard C compiler on several platforms. The translator code was compiled and tested on SUN UNIX, SCO XENIX system, and AIX systems at SLAC. The source code is available for compiling on the target system.

The high level translator was also tested using the Free Software Foundation [13] *flex* and *bison* on a Sun workstation. There were some differences between *lex* and *flex* and between *yacc* and *bison,* leading to some minor problems. The best results and the easiest porting were achieved using the UNIX or XENIX system supplied *lex* parser generator and *yacc* compiler compiler.

A UNIX *make* file was created to generate the parser and the compiler and to compile the resulting C code.

All of the above are available in source code form from the authors (see below). The *llib* and *ylib* libraries needed by *lex* and *yacc* are normally supplied with the UNIX system, and cannot be provided.

VII. Future work

Many of the features of the VIM, VIS, and ATC are inherently simple to make simulation debugging easier. However, theses models are highly modular in nature, and code modification is straightforward.

For the VIM, improved error handling and protocol violation checking/reporting are to be added. Also, as the FASTBUS standard evolves to incorporate more sophisticated broadcast and block-transfer operations, the VIM model will be updated to support these modes.

For the VIS, full functionality is planned (broadcast and logical addressing, block transfer, CSR registers that conform precisely with the FASTBUS standard, uniform SS codes, etc.). There are two reasons for developing a well organized, FASTBUS compliant slave as a reference model. First, masters can be tested in a fully consistent manner; the results of simulation can be expected to correspond very closely to a real FASTBUS environment. Second, and perhaps more important, the "technically perfect" slave could also serve as a reference model (starting point) for designing real slave modules using "cut-and-paste" design methods. This is one of the real values of using hardware description languages.

All of the code described above, as well as the corresponding documentation, is available via electronic mail and/or file transfer protocols. Information on the VHDL implementation and the translator may be obtained by electronic mail from tomdean@slacvm.slac.stanford.edu or from mjh@eng3.hep.uiuc.edu. The source code may be obtained by sending electronic mail to uihepa::mjh or mjh@eng3.hep.uiuc.edu.

VII. References

[1] IEEE Std 960–1989, *IEEE Standard FASTBUS Modular High-Speed Data Acquisition and Control System,* NY: IEEE, 1990.

[2] J. R. Armstrong, *Chip-Level Modeling with VHDL,* Prentice-Hall, 1989.

[3] E. Sternheim, et al., *Hardware Modeling with Verilog HDL,* Automata Publishing, 1990.

[4] T. W. Willwerth, "A Virtual FASTBUS System," Department of Electrical Engineering, University of Illinois, Urbana, IL, M.S. thesis, 1987.

[5] R. W. Downing, et al., "A Virtual FASTBUS Master Module," *IEEE Transactions on Nuclear Science,* vol. NS–34, no. 3, June 1987, pp. 692–694.

[6] DAZIX/Intergraph Corp., Huntsville, AL 35894–00001.

[7] IEEE Std 1177–1989, *IEEE FASTBUS Standard Routines,* NY: IEEE, 1990.

[8] IEEE Std 1076–1987, *IEEE Standard VHDL Language Reference Manual,* NY: IEEE, 1988.

[9] D. R. Coelho, *The VHDL Handbook,* Boston: Kluwer Academic Publishers, 1990.

[10] Viewlogic Systems, Incorporated, Marlboro, MA 01752.

[11] Intermetrics, Incorporated, Cambridge, MA 02138.

[12] William Billowitch, The VHDL Consulting Group, Allentown, PA 18103.

[13] Free Software Foundation, Incorporated, Cambridge, MA 02139.

[14] B. Kernighan and D. Brown, *The UNIX Programming Environment,* Prentice-Hall, 1984.

[15] T. Mason and D. Brown, *Lex and Yacc,* O'Reilly & Associates, 1990.