A Simulation of the FASTBUS Protocols

A W Booth

CERN

Geneva, Switzerland.

Abstract

FASTBUS is a standard bus system being developed for high speed data acquisition and processing in the next generation of large scale physics experiments. Prototypes are being built according to a draft specification. The FASTBUS protocols have been simulated using a powerful software tool which is a computer description language. This Instruction Set Processor Specification language, ISPS, has been used in the design and development of several microprocessor systems. It's applications are diverse, including automated design and the generation of machine relative software, as well as simulation. The results of the FASTBUS simulation are presented, with an overview of the ISPS hardware description language. An additional facility is discussed, which supplements the simulation by providing a visual presentation of the FASTBUS signals, that is, a timing-graph generator.

1. Introduction

FASTBUS, U.S.NIM Committee[1], is a high speed data acquisition and processing standard intended for use in large scale physics experiments. FASTBUS systems are built up from two kinds of segments, referred to as backplane segments and cable segments. Backplane segments are housed in crates which can be linked together by cable segments. Figure 1 shows an example of a simply-connected FASTBUS system. Communication can take place between any two points in the system, with the routing handled by look-up tables in Segment Interconnects which connect Cable Segments to Backplane Segments. Bus protocols, for communicating devices in the system, appear as flowcharts in the draft specification. To test the integrity of these flowcharts and provide the first step towards a large FASTBUS system simulation, a simple model of a FASTBUS Backplane Segment was developed using a hardware description language. In this model, communication between masters and slaves is simulated, as well as a priority arbitration mechanism which resolves contention if simultaneous requests for bus mastership occur.

The simulation is supplemented by computer generated timing diagrams, Booth[2], which provide a visual display of FASTBUS signals. This paper contains a brief description of the hardware description language, and a detailed account of its use for FASTBUS simulation. For an introduction to FASTBUS itself, see Rimmer[3].

2. The Hardware Description Language

The Instruction Set Processor Specification (ISPS) language, Barbacci [4], is a computer hardware description language which has been used to describe the behaviour of a number of microprocessor systems.

These include the Intel 8080, Motorola 6800 and Rockwell 6502. ISPS however is not confined to describing processor behaviour; it has been used as a vehicle for research in many areas, including automatic design of digital circuits, Hafer & Parker[5], compiler-compiler research, Cattel[6], verification of machine language programs, Crocker[7], and simulation of new architectures, Parker[8].

In the field of high energy physics, ISPS was successfuly used in the development of the MICcroprogrammed Engine (MICE), Halatsis[9], Van Dam[10], a fast microprogrammed processor (and PDP 11 emulator) for on-line data filtering in high energy physics experiments.

2.1. The ISPS Notation

Although ISPS can be viewed as a programming language, the aim of the notation is to describe computers and other digital systems as well as general computational algorithms.

The ISPS notation, Barbacci[11], describes the interface and behaviour of hardware units. The interface (i.e. external structure) describes the number and types of carriers used to store and transmit information between the units. In the simplest case, a unit is a carrier (e.g. a bus, a register, a memory, etc.), completely specified by its bit and word dimensions as shown in fig. 2. The description of the 8080 in fig. 2 begins by specifying the memory state, which is declared as an array of 64K words, each 8 bits wide. The memory has a name "M", and an alias "memory". In ISPS these aliases are a special form of a comment and are useful for indicating the meaning or usage of a register's name. Similarly, in the definition of the processor state, the PC which is declared as a single word having 16 bits, has the alias "program.counter". ISPS allows the user to select fields of a previously declared entity and treat them as individual entities. This can be seen in the definition of the instruction register in fig.2. By declaring a "GROUP" field, for example, a user can treat bits 6 and 7 of the instruction register as a separate entity. Each time the "GROUP" field is written into, it is automatically reflected in the instruction register.

The behavioural aspects of the units are described by procedures which specify the sequence of control and data operations. For example, Fig.3 shows a procedure which loads an Instruction Register with the contents of the memory location pointed at by the Program Counter, and then calls another procedure to execute the instruction. The next instruction is then loaded and so on.

A compiler and simulator exist for ISPS. These programs are written in BLISS-10 and run on the DEC PDP-10 computer. The compiler parses the users ISPS description and produces an output file which is eventually transformed into an executable module, see Booth[12]. The Simulator has a set of basic commands, for example, starting and stopping procedures, setting or interrogating registers and memories, tracing and monitoring variables,etc.

3. Wired-OR Connections

ISPS lends itself very well to describing bus systems, since a bus is simply a 'carrier'. However, bus lines which can be driven by many sources must be declared such that, in a wired-OR configuration, the line will take the logical value 1 if driven by any of the sources. There are various ways to achieve this in ISPS, but one which preserves a readable description is to describe the bus as a procedure, and to call for its update each time a source changes its value, as in fig.4.

In the actual simulation of a backplane segment, only those bus lines which can be driven simultaneously by several sources are described in this way, e.g. arbitration request (AR) and arbitration vector (AL) lines. Although, for example, the address synchronisation line (AS) can be driven by many sources, it should in principle be driven by only one source at any one time, i.e.the bus master. To monitor the error situation where more than one source attempts to drive AS, then its declaration must be changed to a procedure.

4. Signal Propagation

Another consideration is whether (i) a signal should be seen to propagate along the bus, or (ii) all devices on the bus should see the signal at the same time. The former produces a more realistic simulation of the bus, although the latter is simpler and maybe adequate . Once a Master-Slave connection has been made, there is little advantage to be gained by simulating signals propagating along the bus. However, before the Master-Slave connection is made, contending Masters must arbitrate for use of the bus. In this situation, a model of the bus where signals are seen to propagate, is better for modelling the FASTBUS arbitration mechanism[1].

In the present work a non-propagating scheme was used, even for arbitration, as the prime purpose of the model was to test the flowcharts. Although the model is not very realistic for demonstrating the FASTBUS arbitration mechanism, it is adequate to test whether or not the logic in the arbitration flowcharts is valid. A propagating bus scheme has since been used in the simulation of a FASTBUS to CAMAC interface, Asboe-Hansen[13].

5. Simulation of a Backplane Segment

The first part of the ISPS description is a declaration of the FASTBUS lines, wire-ORed where necessary, see Fig.5.

The behavioural aspects of the description consist of a procedure for each of the FASTBUS flowcharts. Some examples of flowchart names are as follows:-

Arbitration timing controller
Arbitration at master
Master address cycle
Slave data cycles, etc.

To test the protocol procedures, several bus devices, Masters and Slaves, are also described. Certain procedures run continuosly, e.g. Arbitration Timing controller and Slave Address Cycles, while others are invoked by a procedure 'call'. The following example illustrates a simulation session involving three Masters and a Slave. Using the basic commands of the simulator, the arbitration request line (AR) is set for all three masters. Each of the Masters is given a different arbitration vector, and the control parameters of the Master with the highest vector are set to address the Slave and perform a single word read cycle. At the start of the simulation or at any user defined break points, commands can be given to trace or monitor any signals of interest, or to continue with the simulation.

6. Computer Generated Timing Diagrams

The ISPS Simulator can write traced variables to an output file and/or the terminal. This feature is used by a program written to produce timing diagrams of ISPS simulations. The Timing Diagram Generator (TDG), Booth[2], is written in FORTRAN and run on the IBM 370 computer. It interprets the file of traced variables produced by the simulation. For one simulation run, a variety of diagrams can be obtained. They can be generated for both normal and abnormal working modes of the system, and are particularly useful for design and debugging in the prototyping stage of a project. The TDG interacts with the user to elicit variable names, time range, and title of the diagram. An example of a diagram taken from the FASTBUS simulation is shown in Fig.6.

7. Finding Errors

A combination of the simulation and the generated timing diagrams spotlighted certain errors in the flowcharts. These errors were corrected in the simulation before formal updates to the flowcharts were proposed. In some cases, these errors could have been found in a desk check. In cases where 2 or more procedures interact, errors have been much easier to locate under simulation conditions. An example of this is the situation where a master does not wait for a handshake after transmitting the final word of a data transfer.

8. Conclusions

The ISPS hardware description language has been used to simulate a FASTBUS backplane segment and test the FASTBUS protocols. Certain errors were found and corrected in the simulation. The first stage of a large FASTBUS system simulation is almost complete, the next stage is to simulate a Segment Interconnect and a cable segment so that two backplane segments may be joined together.

Acknowledgements

Of the people who have helped me in my work, I would like to thank especially the following:

E.M.Rimmer
P.Asboe-Hansen
J.Joosten
M.Letheren
R.Nierhaus

References

[1] [U.S.NIM Committee,1980], FASTBUS Modular High Speed Data Acquisition System for High Energy Physics and other Applications, Tentative Specification.

[2] [Booth,1981], Computer Generated Timing Diagrams to Supplement Simulation, Paper submitted to the International Conference on Computer Hardware Description Languages and their Applications, Kaiserslautern Univ., September,1981.

[3] [Rimmer,1980], An Introduction to the FASTBUS system, DD/80/27, CERN.

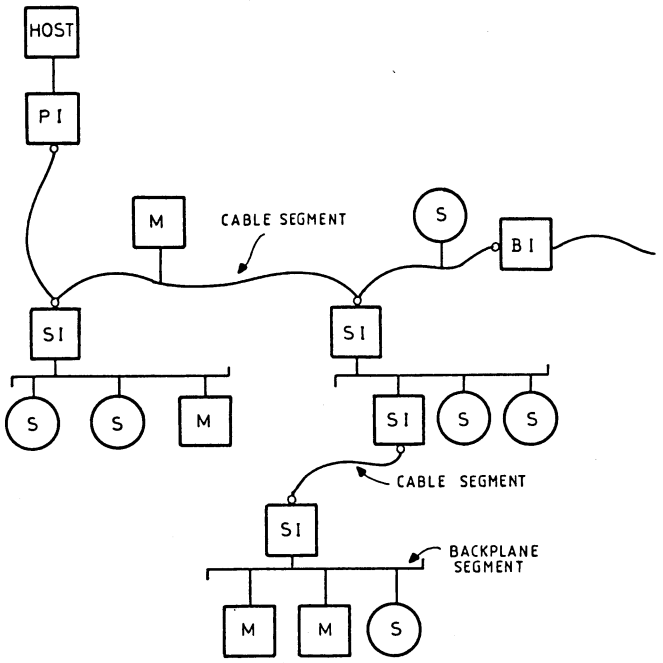
[4] [Barbacci et al,1977], The Symbolic Manipulation of Computer Descriptions: The ISPS Computer Description Language, Technical Report, Department of Computer Science, Carnegie-Mellon University.

[5] [Hafer & Parker,1978], A Register-transfer Level Digital Design Automation: The Allocation Process, Design Automation Conference Proceedings no. 15, ACM SIGDA, IEEE Comp. Soc. Tech. Com. on Design Automation, June 1978, pp. 213-219.

[6] [Cattel,1978], Formalisation and Automatic Derivation of Code Generators, Ph.D. Thesis, Department of Computer Science, Carnegie- Mellon University.
[7] [Crocker,1977], State Deltas:A Formalism for Representing Segments of Computation, Ph.D. Thesis, Computer Science Department,UCLA.
[8] [Parker,1978], Description and Simulation of Microcode Execution, Proceedings of the 5th Annual Computer Architecture Symposium, ACM SIGDA, IEEE Computer Society.
[9] [Halatsis,1980] Architectural Considerations for a Micro-programmable Emulating Engine Using Bit Slices, Proceedings of the 7th Annual symposium on Computer Architecture, IEEE-CS and ACM, La Baule, France.

[10] [Van Dam] Simulation of a Horizontal Bit Sliced Processor Using the ISPS Architecture Simulation Facility, DD/80/30, CERN. (To be published in IEEE Transactions on Computers: Microprogramming Tools and Techniques).
[11] [Barbacci,1981], Instruction Set Processor Specifications(ISPS): The Notation and its Applications, IEEE-CS Transactions on Computers, Vol. C-30, No.1, January 1981.
[12] [Booth,1980], Running ISPS on the PDP-10, CERN-FBDOC#n35, 1980
[13] [Asboe-Hansen,1981] To appear

A SIMPLY-CONNECTED FASTBUS SYSTEM



PI = Processor Interface
SI = Cable/Backplane Segment Interconnect
BI = Buffered Interconnect
M = FASTBUS Master Device
S = FASTBUS Slave Device

FIGURE 1

```
** MEMORY.STATE **

M\memory[0:63K]<7:0>,

** PROCESSOR.STATE **

PC\program.counter<15:0>,
DR\double.registers[0:3]<15:0>,
R\registers[0:7]<7:0>:=DR[0:3]<15:0>,
B<7:0>:=R[0]<7:0>,
C<7:0>:=R[1]<7:0>,
D<7:0>:=R[2]<7:0>,
E<7:0>:=R[3]<7:0>,
H<7:0>:=R[4]<7:0>,
L<7:0>:=R[5]<7:0>,
SP\stack.pointer<15:0>:=DR[3]<15:0>,
PSW\status.word<7:0>,

** INSTRUCTION.FORMAT **

IR\instruction.register<7:0>,
EBIT<>:=IR<3>,
GROUP<1:0>:=IR<7:6>,
DFIELD<2:0>:=IR<5:3>,
DRFIELD<1:0>:=IR<5:4>,
SFIELD<2:0>:=IR<2:0>,
```

Fig.2 An ISPS Description of the Memory State, Processor State and Instruction Format of the INTEL 8080 microprocessor.

```
LI\Load.instruction:=
      BEGIN
      IR = M[PC] next
      PC = PC+1  next
      EXEC()     next
      restart LI
      END,

EXEC\Execute.instruction:=
      BEGIN
      ! comment     This procedure executes the instruction
      END,
```

Fig.3 Example of ISPS procedures

```
BUS.LINE()<>:=
      BEGIN
      BUS.LINE=SOURCE.1 OR SOURCE.2 OR .......
      END,

  PROC():=
      BEGIN
      !
      ! IN THIS PROCEDURE, SOURCE.1 DRIVES THE BUS
      !
      SOURCE.1=1 NEXT
      BUS.LINE()
      END,
```

Fig.4 Updating a Bus Line

```
 AS\address.sync(),
 AK\address.ack(),

 DS\data.sync(),
 DK\data.ack(),

 CB\control.block(),
 NH\no.handshake(),
 EG\enable.geographic(),
 AD\address.data<31:0>,

 RD\read(),
 BK\busy.ack(),
 NK\negative.ack(),

 PE\parity.enable(),
 PA\parity(),

 !
 !   COMMENT:   The arbitration request line is a wired-OR of
 !              master requests
 !
 AR\arbitration.request()<>:= begin AR=REQUEST NEQ 0 end,
 !
 !   COMMENT:   The arbitration vector line are a wired-OR of
 !              master asserts
 !
 AL\arbitration.vector()<5:0>:= begin AL<0>= ASSERT[0] NEQ 0;
                                      AL<1>= ASSERT[1] NEQ 0;
                                      AL<2>= ASSERT[2] NEQ 0;
                                      AL<3>= ASSERT[3] NEQ 0;
                                      AL<4>= ASSERT[4] NEQ 0;
                                      AL<5>= ASSERT[5] NEQ 0
                                      end,
 AG\arbitration.grant(),
 GK\grant.acknowledge(),

 WT\wait(),
 SR\service.request(),
 RB\reset.bus(),

 SL\serial.line(),
 SLR\serial.line.return(),
```

Fig.5 ISPS description of FASTBUS lines

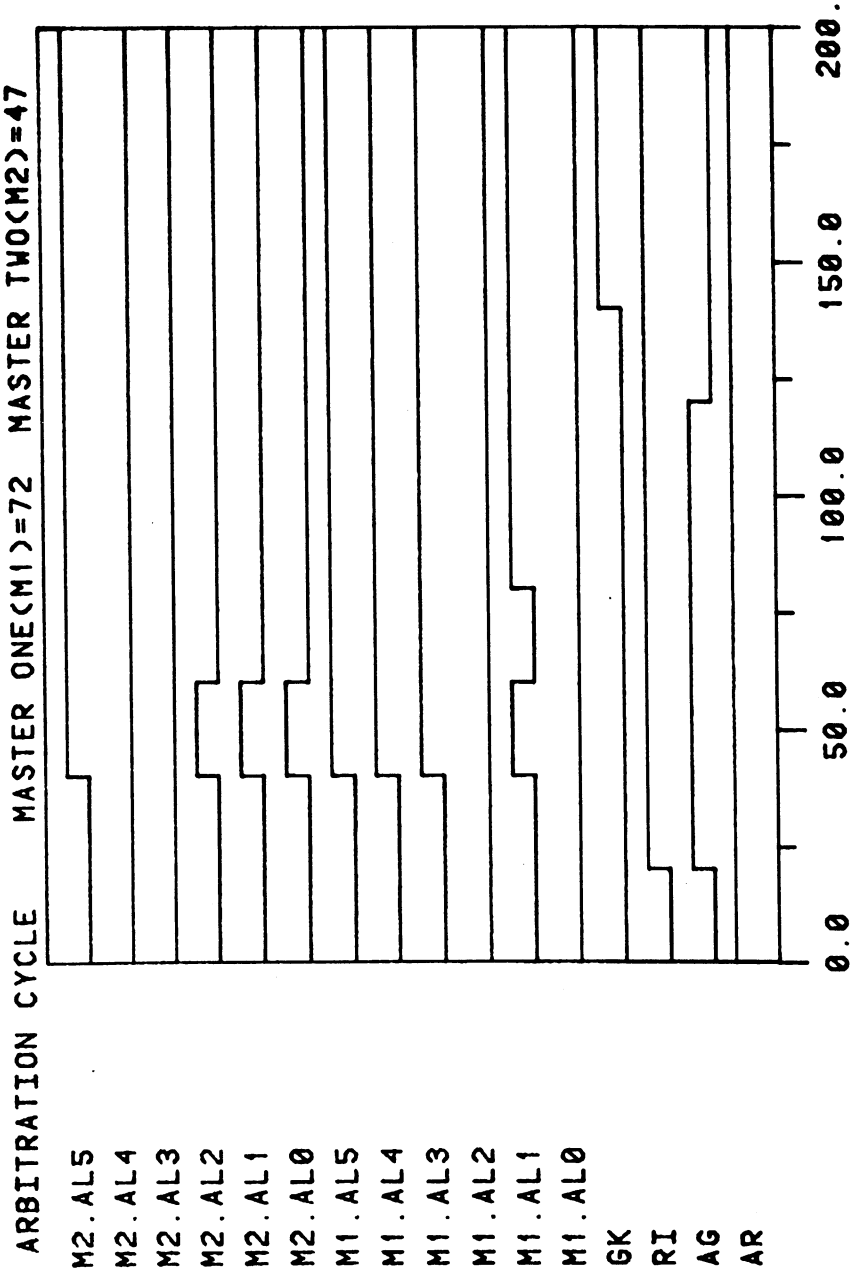


Fig.6 Timing Diagram taken from the FASTBUS Simulation