# COMPLEX MICROCIRCUIT SIMULATION AND TEST DEVELOPMENT USING BEhavioral STimulus TEST (BEST TEST) SOFTWARE

**Ronald D. Cox**
**Naval Surface Warfare Center (NSWC)**
**Crane, Indiana 47522 USA**
**Phone (812) 854-5251 Fax (812) 854-1916 E-Mail: ron@homer.nwscc.sea06.navy.mil**

ABSTRACT

**Developing comprehensive test stimulus for complex microcircuits, particularly microprocessor boards, can be a tedious and time consuming task of hunt and peck; racking up thousands of work hours of labor. This paper will describe a behavioral simulation method using software emulators for the creation and automatic timing generation of test stimulus. "BEST TEST(c)" software was used to develop tests for a 1750A based microprocessor module which also contains a MIL-STD-1553 data bus controller hybrid and other circuitry for which stimulus can be developed using behavioral methods. This paper will show how this technique saved many hours of tedious analysis and on tester debugging.**

## I. INTRODUCTION

The Naval Surface Warfare Center (NAVSURFWARCENDIV) Crane, Indiana is responsible for qualification test development for the 1750a microprocessor based, SEM format E module set. During the selection process for a method of testing the Embeddable Standard Avionics Processor (ESAP) module, Integrated Test Solutions (ITS) presented a pitch on using behavioral modeling techniques {BEST TEST(c)} for testing boards such as this one. After studying this method, it appeared to be superior to any other known method and therefore was utilized.

The power of the BEST TEST(c) tool then took shape as the development proceeded. This tool in combination with Teradyne's LASAR simulation environment and LABEL behavioral modeling language, dramatically lowered the test development effort, allowing for a Test Program Set (TPS) development that is affordable and competitive within the current TPS development market.

The BEST TEST(c) tool, which extracts nodal data from I/O pins that connect the Unit Under Test (UUT) to the Behavioral simulation models "Transactor's" that surround the UUT (as shown in figure 1.0), translates that data into TESTCOM compatible pattern files. This paper illustrates the use of this tool by outlining the TPS development of the Embeddable Standard Avionics Processor (ESAP) module developed by the Standard Hardware Acquisition and Reliability Program (SHARP).

## II. THE STANDARD PROBLEM

The conventional method of stimulus generation for the above module would be tedious and time consuming. The engineer would stimulate the inputs of the module using TESTCOM in the following fashion:

1.SEND OPCODE TO PROCESSOR
2.CLOCK FOR MANY PATTERNS
3.SIMULATE
4.PERUSE SIMULATION RESULTS TO DETERMINE AT WHICH PATTERN PROCESSOR RETURNS
5.CHANGE CLOCKS ABOVE TO REFLECT TIME OBSERVED IN STEP 3
6.RETURN TO STEP 1 AND REPEAT FOR ALL OPCODES SENT.....

LOOP UNTIL BUDGET EXPIRES

The difficulty in this approach is that the engineer does not know when particular devices on the module will be ready for stimulus until they simulate the module. The engineer would develop patterns based on previous simulation results so that the devices on the board are in the proper state as to receive the input stimulus. This can be a tedious and time consuming task of hunt and peck; racking up thousands of work hours of labor. The simulation would be developed in an iterative manner. You would first simulate to determine when and how to apply stimulus to the module. You would then add stimulus to your pattern file based on the previous simulation run. A process of simulation and pattern creation would continue until fault coverage goals are met. This process is inherently dangerous because minor pattern changes at the beginning of the simulation or minor changes to module design may have major effects throughout. Test engineers run the risk of losing major amounts of work due to small changes. A closer look at the ESAP module TPS development within this paper will reveal some of the

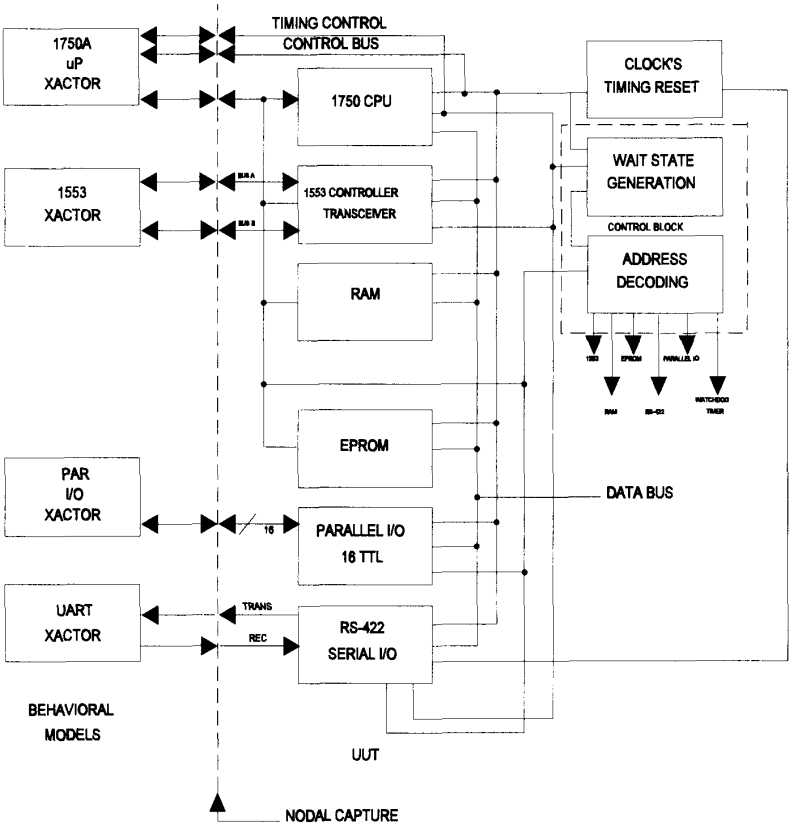


Figure 1. Simulation Model Block Diagram

difficulties encountered in the standard method of pattern generation and the solutions offered by the BEST TEST (c) simulation process.

## III. THE ECONOMIC PROBLEM

With today's economy, it is essential that developmental cost's for military electronics decrease. This must be done in association with increase, or at least no decrease, in reliability. For both cost and reliability scenarios to be satisfied the past developmental methods must be reevaluated. Not only does the standard development method take longer, it also causes higher support cost in the future. By using this development method and researching new method's, it will soon become economically feasible as well as absolutely necessary for the health and welfare of our nation to test military electronics thoroughly to prevent life threatening fleet failures and finding latent design defects.

This BEST TEST(c) developmental method will decrease the life cycle cost of the electronics. When design changes are made on an electronic module there will be much less effort in changing the TPS which tests this module. Re-simulation of the module could be as simple as changing the module net list, simulate and then postprocess to the tester. Whereas before, it was necessary to recreate test vectors in order to contain the correct timing changes that may be present within the new design. These benefits will become apparent after studying this paper. The TPS developed with this method will follow the module from birth to death without any proceeding lengthy developments reducing the life cycle costs.

## IV. WHAT IS BEST TEST(c)?

There are two primary components of the BEST TEST(c) technology:

1) TRANSACTOR'S : LASAR Behavioral Language (LABEL) Models.

2) STIMULUS EXTRACTION : Capturing TRANSACTOR contributions.

The TRANSACTOR'S emulate system interfaces such as the Rockwell R6551 serial interface, Intel 8255 parallel interface, MIL-STD-1553 serial interface and complex components such as the PACE P1750a Microprocessor. A P1750a TRANSACTOR is able to emulate the microprocessor and act as the host for a TRANSACTOR simulation. The MIL-STD-1553 interface TRANSACTOR is able to receive communications from the module and respond appropriately. With this approach LASAR simulation is able to emulate a high level, system like, functional test. The TRANSACTOR'S are added to the master LASAR netlist and a TRANSACTOR simulation is performed. When the desired response is achieved, the contributions from the TRANSACTOR'S at the module I/O (Figure 1) are captured by the BEST TEST(c) tool and the stimulus is converted into a traditional TESTCOM language pattern file.

The primary reasons for the improvements in TPS development are as follows:

1) Object oriented nature of BEST TEST technology.

2) High level approach to code development.

3) Automatic generation of response stimulus at system interfaces.

The object oriented nature of the technology allows TRANSACTOR models to be reused and modified, helping to eliminate the massive duplication of effort which might normally occur using manual TESTCOM to develop tests for a group of modules. The high level approach to code development allows the TPS development engineer to concentrate on the module functionality and allows the transactor's to handle the tedious details of the operation of the module. The use of a transactor at a system interface enables automatic generation of response stimulus. With manual TESTCOM an engineer would typically be required to simulate, evaluate the response and proceed with additional stimulus. With the BEST TEST(c) approach, the TRANSACTOR'S handle hand shake protocols and stimulus is applied when the interfaces are in the correct state.[1]

## V. PHASE I - TRANSACTOR CREATION

### *P1750A TRANSACTOR DEVELOPMENT*

The 1750a transactor (XACTOR) developed by Integrated Test Solutions (ITS), P1750A_XT, supports basic memory and I/O read and write cycles. In addition to these basic functions there are three additional operations: The first instruction in the OPCODE.DAT control file indicates whether XACTOR microprocessor emulation will be used, no operation cycle to insert wait states, and wait for interrupt 0 (INT0) to occur. On the embeddable Standard Avionics Processor (ESAP) card INT0 is used to indicate 1553 bus controller interrupts.

### *INITIALIZE*

The P1750A_XT XACTOR consists of a synchronizing clock input, control inputs and outputs, interrupts, and a 16 bit information bus (IB[0:15]) which carries both address and data with bit 0 being the Most Significant Bit (MSB). The clock input is used to synchronize all activity created by the XACTOR. The clock input should be specified in the XACTOR simulation timing file as an input with a phase assertion time different from the CPU clock assertion time. Preferably an assertion time early in the pattern at the same time as the other inputs (RESET, etc.) in the XACTOR simulation will be used to minimize the number of phases created during BEST Test(c) Stimulus capture. The XACTOR input clock will be stripped during Capture using the DELETE_NODES command.

Upon reset the P1750A_XT tri-states the information bus as well as processor output control signals M_IO, BSLOCK_, BUSY_, STRBD_, STRBA and R_W. All error and interrupt signals are set to the inactive state. The XACTOR also reads in the OPCODE.DAT control file. The OPCODE.DAT file contains information on whether or not the XACTOR will be used and if so emulation will be used. The following section shows how the OPCODE.DAT control file is formatted and explains the operation of the file.

### *OPCODE.DAT Control File Format*

The OPCODE.DAT Control File is read by the microprocessor at the beginning of simulations and is used to control the operation of the microprocessor emulator. The format of the OPCODE.DAT file is illustrated in the following table.

**TABLE I**
OPCODE.DAT Control File Format

| Address | Data | Operation |
|---|---|---|
| 0011 | 5555 | 0 |

The first line of the file and every odd line after that are interpreted as comments by the P1750A_XT XACTOR to enable the user to comment the OPCODE.DAT source file. As shown in the table above, the first field of an operation is the 16-bit address which will be issued on the Information Bus. The second field of an operation is the 16-bit data which will also be issued on the Information Bus during write cycles. The third field of an operation is the operation indicator. There are 7 operations which are supported by the P1750A_XT XACTOR as shown in the following table.

**Table II**
P1750A_XT XACTOR Supported Operations

| Operation | Description |
|---|---|
| 0 | I/O Write Cycle (M_IO Low). |
| 1 | I/O Read Cycle (M_IO Low). |
| 2 | Lockout bus, select microprocessor emulation. |
| 3 | No operation, create wait states. |
| 4 | Memory Write Cycle (M_IO High). |
| 5 | Memory Read Cycle (M_IO High). |
| 6 | Wait for 1553 Interrupt (INT0). |

The first instruction in the OPCODE.DAT control file is always interpreted as the command selecting P1750A_XT Microprocessor Emulation. The follow table indicates the two possible selections for the first instruction.

**Table III**
OPCODE.DAT Control File First Operation

| Address | Data | Operation | Description |
|---|---|---|---|
| 0 | 0 | 2 | P1750A_XT Microprocessor Emulation Selected, lock out bus |
| 0 | 0 | 0 | P1750A_XT Microprocessor Emulation not selected. |

When P1750A_XT Microprocessor Emulation is selected the OPCODE.DAT control file operations are executed to stimulate the module. When Emulation is not selected the on board processor is granted the bus (BUSGNT_ Low) and enabled to run self test and on board ROM code.

***Operation***

Table II is referenced throughout this section as the Operation Codes are explained in more detail. The first instruction of the OPCODE.DAT file indicates whether the XACTOR will be used to emulate the 1750A Microprocessor. If Emulation is selected with Operation Code 2, then the Bus Lock signal BSLOCK_ is asserted to prevent the on board processor from gaining access to the bus.

Operation Code 3 is used to create wait states while another module operation such as a serial data transfer are taking place. This operation will create a waiting period of 8 XACTOR clock periods.

Operation Code 6 will cause the XACTOR to wait for the INT0 signal to be asserted (high).

Operation Codes 0 and 4 are I/O and Memory Write Cycles respectively. These operation cycles emulate the basic write cycles of the PACE 1750A Microprocessor. The steps of the Sate Machine used to control this cycle are as follows:

1. Assert R_W cycle indicator low indicating a write cycle. Assert M_IO low for an I/O write and high for a memory write. Assert BUSY_ active low indicating the bus is currently being used. Assert BSGNT_ low to indicate that the bus has been granted to the P1750A_XT XACTOR. Drive the address on the information bus. Drive STRBA high enabling address to be driven onto address bus.

2. Drive STRBA low latching the address on the module

address bus.

3. Drive the data on the information bus.

4. Drive STRBD_ low latching the data bus information. XACTOR issues message containing the address being written to, the data, and a simulation time stamp. This information will appear on screen during an interactive simulation and in the command listing file for a BATCH simulation.

5. Wait for signal RDYD high indicating that the data cycle has completed.

6. Deassert STRBD_ data strobe, BUSY_ and BSGNT_.

Operation Codes 1 and 5 are I/O and Memory Read Cycles respectively. These operation cycles emulate the basic read cycles of the PACE 1750A Microprocessor. The steps of the State Machine used to control this cycle are as follows:

1. Assert R_W cycle indicator high indicating a read cycle. Assert M_IO low for an I/O read and high for memory read. Assert BUSY_ active low indicating the bus is currently being used. Assert BSGNT_ low to indicate that the bus has been granted to the P1750A_XT XACTOR. Drive the address on the information bus. Drive STRBA high enabling address to be driven onto address bus.

2. Drive STRBA low latching the address on the module address bus.

3. Assert STRBD_ low and tristate information bus enabling data to be driven on the bus by the peripheral device being read from.

4. Wait for RDYD active high indicating that the data cycle has completed.

5. XACTOR issues message containing the address being read from, the data received, and a simulation time stamp.

6. Deassert STRBD_ data strobe, BUSY_ and BSGNT_.

***Interface Pinout***

The following table describes each pin on the P1750A_XT transactor. Type indicates if the pin is an input, output or bidirectional.

**Table VI**
Pin Description

| Type | Pin | Description |
|---|---|---|
| I | CLK | Input clock. Drive XACTOR state machine, Synchronize activity. |
| I | RDYD | Data Ready. Active high indicator that data cycle completed. |
| I | RDYA | Address Ready. Active high indicator that address cycle completed. |
| I | TRGRST_ | Trigger Reset. Pulses during initialization. |
| I | RESET_ | Master Reset input, initialize XACTOR. |
| I | BSREQ_ | Bus request, on board processor requesting bus. |
| I | CS232_ | Serial interface selector, special handling required. |
| I | X2 | Serial interface clock. |
| I | RW | Serial interface read/write indicator. |
| I | INT0 | 1553 interrupt signal. |
| B | BSLOCK_ | Bus lock, lock out bus when emulation selected. |
| B | BUSY_ | Bus busy, read/write cycle boundary indicator. |
| B | IB(0:15) | Information bus, address and data, MSB is bit 0. |
| O | M_IO | Memory I/O selector, high for memory, low for I/O. |
| O | CONREQ_ | Console request, initiate console operations. |
| O | BSGNT_ | Bus Grant, grant bus to on board processor or XACTOR. |
| O | IOINT1 | I/O level interrupt 1. |
| O | IOINT2 | I/O level interrupt 2. |
| O | INT2 | User interrupt 2. |
| O | INT3 | User interrupt 3. |
| O | INT4 | User interrupt 4. |
| O | INT5 | User interrupt 5. |
| O | TMRCLK | Timer clock. |
| O | STRBA | Address strobe. |

**Table VI (CONT)**
Pin Description

| Type | Pin | Description |
|---|---|---|
| O | STRBD_ | Data strobe. |
| O | R_W | Read/Write data flow control. |
| O | MPTER_ | Memory protect error. |
| O | MPAER_ | Memory parity error. |
| O | XADER_ | External address error. |
| O | PDINT | Power Down Interrupt. |
| O | SFLT0 | System fault 0. |
| O | SFLT1 | System fault 1. |

## VI. SAMPLE SIMULATION

This section will show an example OPCODE.DAT simulation control file and an example simulation listing file. The simulation listing file is an example of the messages the XACTOR will issue to indicate the status of communications interactively during simulation.

### *OPCODE.DAT File*

Table VII contains a listing of an OPCODE.DAT file used in stimulating the parallel interface. The P1750A_XT XACTOR will read and execute the OPCODE.DAT control file. The P1750A_XT XACTOR is programmed to recognize that every other line of the OPCODE.DAT file is a comment.

**Table VII**
Example OPCODE.DAT File

```
!!OPCODE.DAT!!
!Select XACTOR emulation, lock out bus
0 0 2
!Read from U2 port A, 8255 parallel interface
0010 0 1
!Read from U2 port A, 8255 parallel interface
0010 0 1
!Read from U2 port B, 8255 parallel interface
0011 0 1
!Read from U2 port B, 8255 parallel interface
0011 0 1
!Write to U2 control register, all ports output ports
0013 80 0
!Write to U2 port A, 8255 parallel interface
0010 55 0
!Write to U2 port A, 8255 parallel interface
0010 AA 0
!Write to U2 port B, 8255 parallel interface
0011 55 0
!Write to U2 port B, 8255 parallel interface
0011 AA 0
!Write to U2 port C, 8255 parallel interface
0012 55 0
!Write to U2 port C, 8255 parallel interface
0012 AA 0
```

### *Simulation Listing File*

Table VIII illustrates the messages which the P1750A_XT XACTOR will issue to the current output devices interactively during a simulation. If the simulation is being run interactively by the user the messages will appear on screen. If the simulation is being run in batch mode the messages will appear in the batch listing file.

**Table VIII**
Simulation List file (i.e. SIMRUN.LIS)

```
LASAR> simul/ran=1:700/tim=typ
Requested range = 1 : 700
Actual range    = 0 : 700
Finished Load of Opcode Data File!!!!!! Type name BUS61553: hardware
output change assigned default functional delay trigger.

Type name p1750: hardware output change assigned default functional
                                                    delay trigger.
Read Add:0010 Dat:XX55 at User Pattern: 68  Time: 50.000 NS
Pattern 69/69. 0 Unsolved races. 0 Unsolved conflicts. 14 Seconds.
Read Add:0010 Dat:XXAA at User Pattern: 76  Time: 50.000 NS
Read Add:0011 Dat:XX55 at User Pattern: 84  Time: 50.000 NS
Read Add:0011 Dat:XXAA at User Pattern: 92  Time: 50.000 NS
Write Add:0013 Dat:0080 at User Pattern: 97  Time: 50.000 NS
Write Add:0010 Dat:0055 at User Pattern: 104  Time: 50.000 NS
Write Add:0010 Dat:00AA at User Pattern: 111  Time: 50.000 NS
Write Add:0011 Dat:0055 at User Pattern: 118  Time: 50.000 NS
Write Add:0011 Dat:00AA at User Pattern: 125  Time: 50.000 NS
Write Add:0012 Dat:0055 at User Pattern: 132  Time: 50.000 NS
Write Add:0012 Dat:00AA at User Pattern: 139  Time: 50.000 NS
```

CREATING TESTCOM WITH BEST TEST(C)

This section will show Best Test(c) being used to extract transactor output and create low level TESTCOM patterns.

### *Running BEST Test(c)*

BEST Test(c) uses a control file like the one shown in Table IX. This file sets up the parameters that determine how stimulus is created. In the RT simulation BUSHOST and both P1750A_XT transactors have stimulus created from their

outputs. To create the stimulus from the system prompt:

```
$ bt :== $btdir:cap1
$ bt cap_control.dat
```

**Table IX**
Capture Control File

```
!CAP CONTROL FOR TESTING CAPTURE ENVIRONMENT
CAPTURE_MODE = REPLACE;  !use replace mode
REPLACE_PIS;             !include all pi activity in new pattern file
DEVICE_NAME E2,E3,E4;    !use these transactor outputs to create
          !new stimulus
PHASE_ERROR = 1500;      !any edge within 1.5ns of each other can
          !be combined into the same phase
FILTER_WIDTH = 1000;
DELETE_NODES = UNNCLK;
```

### *BEST Test(c) Output*

BT produces a low level pattern file and a new netlist. The netlist will have the transactors removed, and any pins formerly driven by a transactor will now be declared as bidirectional nodes that are driven by the pattern file. Table X contains a condensed version of the BT timing and pattern files from the RT simulation. Table XI shows the new netlist created by BEST Test(c).

**Table X**
BEST Test(c) Timing File

Capture.tim File

```
Set Tset 1 clock = 400ns
Phase 1 assert = 5 NS return = 65 NS
Phase 2 assert = 30 NS return = 85 NS
Phase 3 assert = 10 NS return = 399 NS
Window 1 open = 360 NS close = 380 NS
;

Set Phase 1 trigger = $t0pat;
Set Phase 2 trigger = $t0pat;
Set Phase 3 trigger = $t0pat;
Set Phase 4 trigger = $t0pat;

SET DIGITAL ( RESET_ STRBA STRBD )
window = 1,
phase = 1,
Format = $nret;

SET DIGITAL ( IB7 IB6 IB5 IB4 IB3 IB2 IB1 IB0 )

window = 1,
phase = 2,
Format = $nret;

SET DIGITAL ( CPUCLK )
window = 1,
phase = 2,
Format = $nret;

SET DIGITAL ( CPUCLK )
window = 1 phase = 3,
Format = $rzero;
```

Capture.pat File

```
Include 'capture.tim'
Highspeed
Use Tset 1
Cpp = 1

Hi          RD_ WR_ BSGNT_ INT2 IOINT1

  IB7 IB6 IB5 IB4 IB3 IB2
  IB1 IB0 RESET_
    ;!pattern 1
Lo RESET_
;!Pattern 2
   .
   .
   .
!!!!Remainder of Simulation not shown here!!!!
   .
   .
   .
```

**Table XI**
BEST Test(c) Netlist and CAPTURE.E File

Capture.e File

```
Node Section;

bi IB7,@IB7;
bi IB6,@IB6;
bi IB5,@IB5;
bi IB4,@IB4;
bi IB3,@IB3;
bi IB2,@IB2;
bi IB1,@IB1;
bi IB0,@IB0;

End_Node;
```

Capture.net File

```
Board = 'CAPTURE_module';

Include 'CAPTURE.e';

Declaration Section;

Ignore Component Section;

U1=P1750A;
E1=P1750A_XT;
End_component;

Node Section;

@IB7,E1-9,U1-44;
@IB6,E1-8,U1-4;
```

```
@IB5,E1-7,U1-43;
@IB4,E1-6,U1-3;
@IB3,E1-5,U1-42;
@IB2,E1-4,U1-2;
@IB1,E1-3,U1-41;
@IB0,E1-2,U1-1;

End_node;
End_board;
```

---

A new simulation can now be run with the BT output files. This simulation can be checked for timing hazards and accuracy. By adjusting BT parameters, such as PHASE_ERROR, the user can modify how output timing is created. The number of phases and tsets, including assert and return times, can be controlled to meet tester and simulation requirements.

BT output files will have the default name of "capture". This can be changed with an OUTPUT_FILES=FILENAME; command in the capture control file.

## VI. COST DRIVERS

When developing digital Test Program Set (TPS) software using traditional methods, the largest cost driver is stimulus development. When using Best Test(c) software the cost drivers, although minimal compared to traditional stimulus development, exist in transactor modeling. After all modeling is complete, using the Best Test(c) method, the only stimulus development necessary is initialization (if necessary) and clocks. Any additional stimulus development would be minimal. This section will explore these two methods of TPS development and trade-offs associated with both.

### *Traditional TPS Development*

When using the traditional method of TPS development on a microprocessor module, the stimulus development flow progresses as follows:

1.SEND OPCODE TO PROCESSOR
2.CLOCK FOR MANY PATTERNS
3.SIMULATE
4.PERUSE SIMULATION RESULTS TO DETERMINE AT WHICH PATTERN PROCESSOR RETURNS
5.CHANGE CLOCKS ABOVE TO REFLECT TIME OBSERVED IN STEP 3
6.RETURN TO STEP 1 AND REPEAT FOR ALL OPCODES SENT.....
----- LOOP -----

This method is a manual method, machine code oriented,that requires many work/hours of labor and massive CPU time of the simulators host processor. Modeling is confined to on board devices. Considering the stimulus development time and availability of component simulation models (Structural,Behavioral and Hardware), modeling should be a minimal percentage of development time .

### *Best Test(c) TPS Development*

When using the Best Test(c) method of TPS development on a microprocessor module, the stimulus development flow progresses as follows:

1.DEVELOP AND/OR PROCURE TRANSACTOR SIMULATION MODELS
2.WRITE HIGH LEVEL CODE TO FUNCTIONAL EXERCISE CPU
3.COMPILE HIGH LEVEL CODE INTO A ROMCODE.DAT FILE
4.CLOCK FOR MANY PATTERNS (DATA FILE IS PULLED IN AND EXECUTED USING TRANSACTOR)
5.POSTPROCESS USING LSRTAP
6.CAPTURE NODAL DATA USING BEST TEST(c) SOFTWARE
7.SIMULATE TEST PATTERNS GENERATED BY BEST TEST(c)

This method is a high level approach to stimulus development that immensely reduces the stimulus development time. The 1750A assembler code was compiled to a romcode.dat file which is pulled in by the ROM transactor which in turn is accessed by the 1750A CPU for instructions. At this point the stimulus generation becomes semi-automatic. Sending clocks to the CPU under simulation creates stimulus by causing nodal activity between the module I/O and the transactor models which can be captured and converted into stimulus. This Best Test(c) conversion process creates the following files:

- CAPTURE.NET (NEW NETLIST EXCLUDING TRANSACTORS)
- CAPTURE.TIM (TIMING FILE CREATED BY BEST TEST(c))
- CAPTURE.PAT (PATTERN FILE CREATED BY BEST TEST(c))
- LSRTAP.LIS (FILE CONTAINING CAPTURE WARNINGS AND ERRORS)

The major cost drivers using the Best Test(c) method is the transactor development/procurement. The transactor development time is minimal compared to developing stimulus the traditional way for a CPU module.[2]

## VIII. COST SUMMARY

The cost saving accrued by using Best Test(c) as opposed to manual Testcom is dependent upon the complexity of the electronics. The higher the complexity of the circuitry the higher the cost savings. The following table shows a summary of cost savings based on the 1750A CPU module explained earlier. The UCP module listed in the table below is a completed TPS development with a 68000 CPU. The UCP was developed using Traditional stimulus generation methods and does not contain a complex MIL-STD-1553 hybrid bus controller.[1]

| Module | Development Time (Months) | Test Vector Count | BEST Test | Fault Dictionary |
|---|---|---|---|---|
| UCP | 18 | 52,540 | NO | NO |
| ESAP | 6 | 11,739 | YES | YES |
| ESAP(1) | ~24 | ~60,000 | NO | NO |

**NOTE (1) : Estamated manual Testcom Stimulus Development**

## IX. CONCLUSION

I. Dramatic Improvement in TPS Development Time Achieved

II. Primary advantages of BEST Test(c) Methodology

a) Object Oriented Technology: Transactors are reusable and modifiable.

b) High level approach: Complex timing, hand shaking, and other "state sensitive" functions embedded in transactors.

c) System Interface Communications: Automatic generation of response stimulus by transactor.

III. In the competitive market of TPS development it's no longer practical to use manual Testcom in the development of medium to highly complex TPS's.

## ACKNOWLEDGMENT

NSWC Crane thanks Jeff Stearns of Integrated Test Solutions (ITS) for his help in the development of the Test Program Set (TPS) which made this publication possible.

## REFERENCES

[1] Joe Paliotta and Jeff Stearns, "Behavioral Stimulus Test (BEST) Test Development," Teradyne Users Group (TUG) 1993.

[2] John Amaral and Frank Meunier, "BEST TEST BEHAVIORAL STIMULUS FOR TEST DEVELOPMENT," TUTORIAL.