Q'n Apps #P4

**Q:** **How can I test memories with Presto?**

Testing a Memory Under Presto

The Memory Test feature of the ETS is possible because of the tremendous versatility of the HiLevel proprietary gate arrays that control the ETS pin electronics. The powerful Presto software redefines the functions of these chips so that very deep memory devices can be tested using very few vectors. When using the ETS for Memory Test the software takes control of the vectors and the pattern generator program, so it is important that you not modify these resources manually.

It is very important to plan ahead for Memory Test, particularly when building up your DUT board. This is due to the way in which the pin electronics boards are assigned for specific purposes. PE board #1 (pins 1-32) is used for the memory address pins of your device. Presto will control these pins like a counter, sequencing through memory addresses as part of a pattern generator loop. PE board #2 (pins 33-64) will function as the data I/O pins to the memory device under test. PE board #3 (pins 65-96) provides control pin functions for your memory device, such as output enables, chip enables and read/write pins.

You can use the PinList import feature to assign your pins and names, or do it manually from the Main Test setup window. Just be sure to assign pins according to their types as illustrated in the above paragraph, and in accordance with your DUT board wiring. Here are the main steps in preparation. You can also use the Presto User Manual and the HATGOL manual for more assistance.

Begin the setup by checking the Memory Test box on the "Modes" area of the Main Setup window:
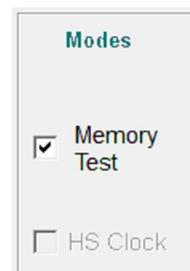
Now the Memory Setup window appears in the lower right corner of the Main Setup window. This is where you will define the basic characteristics of your device and also access other Memory test features.

Select your memory type; SRAM, FLASH or DRAM. "Width" means "number of data pins".

Define the columns of your memory's configuration. Rows will be set automatically.

Set the control pin types for your device.

Memory Type	Width	Depth	Rows	Cols
SRAM	8	32K	512	64

Control Pins
/CS, /OE, /WR

Algorithm
CheckerBoard

Choose the algorithm for the first test pattern.

Create

PinList   SET file   HATGOL

After completing your configuration, PinList will generate the file to use for your DUT board wiring.

The Set file button will use the PinList file to set up the system. Be sure to save the Set file afterwards.

This button will create the HATGOL program for generating the memory test vectors.

There are three suggested Control Pin configurations. Don't worry if none match your configuration exactly; you can edit the PinList file to match your needs.

Control Pins
/CS, /OE, /WR

/CS, /OE, /WR
/CS1, CS2, /OE, /WR
/CS, /OE, /CLK, /WR
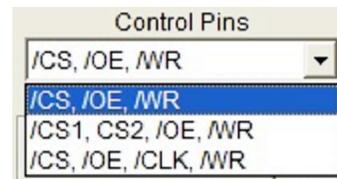
The Presto software supports these standard algorithms. Just use the scroll buttons to select one for your first test. Later, you can select different ones to create more vector sets.

Algorithm
CheckerBoard

WalkingOnes
WalkingZeros
MarchingOnes
MarchingZeros
Sequential Address
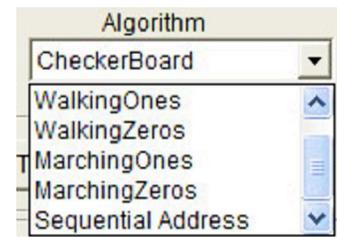
The PinList shown on the next page is the result of clicking the PinList button after your configuration has been defined. Save this file after you have typed in your DUT pin numbers and made any other changes. Then press the Set file button to import the PinList.

```
; SysCh   Pin#    Name    Grp    Type
16        ?       ADR15   1      SMA        ; DISCONNECTED
15        ?       ADR14   1      SMA
14        ?       ADR13   1      SMA
13        ?       ADR12   1      SMA
12        ?       ADR11   1      SMA
11        ?       ADR10   1      SMA
10        ?       ADR9    1      SMA
9         ?       ADR8    1      SMA
8         ?       ADR7    1      SMA
7         ?       ADR6    1      SMA
6         ?       ADR5    1      SMA
5         ?       ADR4    1      SMA
4         ?       ADR3    1      SMA
3         ?       ADR2    1      SMA
2         ?       ADR1    1      SMA
1         ?       ADR0    1      SMA

40        ?       DATA7   2      SMD
39        ?       DATA6   2      SMD
38        ?       DATA5   2      SMD
37        ?       DATA4   2      SMD
36        ?       DATA3   2      SMD
35        ?       DATA2   2      SMD
34        ?       DATA1   2      SMD
33        ?       DATA0   2      SMD

65        ?       /WRITE  3      I          ; make this R1 signal
66        ?       /OE     4      I
67        ?       /CS     5      I

$DISCONNECT PIN 16
```

In our example, channel 16 is disconnected because Address bit 15 is not needed for a 32K depth.

You will set leading and trailing edges for the write pulse. If your write pin is active hi, then use RZ format.

Example from a PinList file for **Memory Test**

The pin TYPEs SMA and SMD in the PinList file mean "Split Memory Address" and "Split Memory Data", as can be seen as the Pin Direction in the Pin Setup area of the Main Setup window. Do not change these settings.

Pin Direction	Split Memadr ▼
Stimulus Format	I/O
	Split I/O
	Split Memadr
Pins...	16, 15 Split Memdata ▼
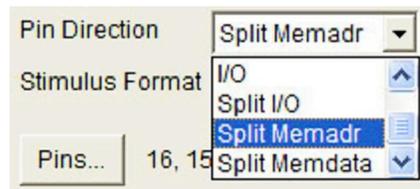
You will also notice that for the Address and Data groups (groups 1 and 2) that the timing section of the Main Setup window has changed. The "Trailing Edge" field is now called "Memory Edge".

Timing

Strobes	Dly	Value	Unit
Leading	5	4	% ▼
Memory Edge	7	75	% ▼
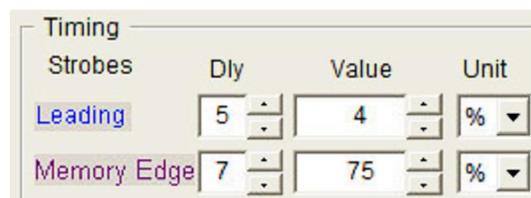
Now you can define your DUT power supplies, logic levels, and test rate. You will see that the sequencing mode is set to Memory Test. Do not change this setting. Save your Set file for future loading.

Run Setup

Test Rate
| Freq (MHz) | 10.000 |
| Period (ns) | 100.000 |

Test ExtClock
| Clock Source | Internal ▼ |
| Sequencing | MemoryTest ▼ |

Select a Test Pattern from the Algorithm box in the MemoryTest Setup section. For your first pattern, it is common practice to use CheckerBoard.

If you open the Vectors window, you can see your assigned pin groups displayed. You'll also see random, unmeaningful data as the vectors (see below). Press the HATGOL button on the Memory Test section of the Main Setup window. This action will cause the Comment field of the Vectors window to be filled with the HATGOL statements necessary to translate vectors for the selected test pattern, as can be seen in example on the next page. Refer to the HATGOL manual for details about these instructions and commands in the Comment field.

Vector Address (Hex)	Program	Address	Data	/WRITE	/OE	/CS	Comments
000000	NOOP	?Z?? FFFX	?? XF	1	Z	1	
000001	NOOP	?Z?? FFXX	?? FX	Z	Z	1	
000002	NOOP	?Z?? XFFF	?? FF	Z	Z	0	
000003	NOOP	?Z?Z FXXF	?? FF	Z	Z	Z	
000004	NOOP	??Z? XFFF	?Z FF	Z	Z	Z	
000005	NOOP	???Z FFFX	?? XF	Z	Z	Z	
000006	NOOP	ZZ?Z FXFX	?F XF	1	Z	Z	
000007	NOOP	?AZZ FFFF	?? FF	Z	1	Z	
000008	NOOP	??ZZ FFXX	?? FF	Z	Z	Z	
000009	NOOP	?Z?? FXXF	?? FF	Z	Z	Z	
00000A	NOOP	???? FFFX	?? FF	Z	Z	1	
00000B	NOOP	ZZ?? FXFF	ZZ FF	Z	1	Z	
00000C	NOOP	ZZ?? FFXF	?? FF	1	0	1	
00000D	NOOP	ZZ?Z FFXX	?? FF	Z	0	1	
00000E	NOOP	?Z?? FFXX	Z? FX	Z	1	Z	
00000F	NOOP	???? FFFX	?? FF	Z	1	Z	
000010	NOOP	???? XFFX	?? FF	Z	Z	Z	

Vector Window Before HATGOL Button is Pressed

In the vectors Comment field, some definitions may need to be modified prior to translating, i.e. 'WRITE', 'READ', and 'default'. Each of these is preceded with the '#define' statement. Initially these will be all ones. The order of bits for these binary values is determined by the 'consign' statement, which is used to assign the appropriate control pins to the 'Control' group referenced by the HATGOL 'control' statement. For example, if groups /WRITE, /OE, and /CS (all active low) are groups 3, 4, and 5 respectively, and are 'consigned' as 3 4 5, then modify the 'WRITE' definition from 0b_111 to 0b_010 and

modify the 'READ' definition fomr 0b_111 to 0b_100 ('0b_' denotes binary). Reference the HATGOL Manual for more detailed information.

```
                                        /
                                        W
Vector                                  R
Address        Program                  I  /  /
 (Hex)                                   T  O  C
                         Address   Data  E  E  S      Comments

000000  NOOP    ?Z?? FFFX  ?? XF  1 Z 1     HATGOL MemoryTest Checkerboard
000001  NOOP    ?Z?? FFXX  ?? FX  Z Z 1
000002  NOOP    ?Z?? XFFF  ?? FF  Z Z 0     jmp $Start
000003  NOOP    ?Z?Z FXXF  ?? FF  Z Z Z     consign 3 4 5
000004  NOOP    ??Z? XFFF  ?Z FF  Z Z Z     #conf width 8
000005  NOOP    ???Z FFFX  ?? XF  Z Z Z     #define depth 32K-1
000006  NOOP    ZZ?Z FXFX  ?F XF  1 Z Z     #define PageSize 64-1
000007  NOOP    ?AZZ FFFF  ?? FF  Z 1 Z     #define repeats 512/2-1
000008  NOOP    ??ZZ FFXX  ?? FF  Z Z Z     #define rd_count 64/2-1
000009  NOOP    ?Z?? FXXF  ?? FF  Z Z Z     #define WRITE 0b_010
00000A  NOOP    ???? FFFX  ?? FF  Z Z 1     #define READ 0b_100
00000B  NOOP    ZZ?? FXFF  ZZ FF  Z 1 Z     #define default 0b_111
00000C  NOOP    ZZ?? FFXF  ?? FF  1 0 1     #define InitData zeroes
00000D  NOOP    ZZ?Z FFXX  ?? FF  Z 0 1     #define Complement ones
00000E  NOOP    ?Z?? FFXX  Z? FX  Z 1 Z
00000F  NOOP    ???? FFFX  ?? FF  Z 1 Z     control default all
000010  NOOP    ???? XFFX  ?? FF  Z Z Z
000011  NOOP    ZZ?Z FXFX  ?Z XX  1 Z Z
000012  NOOP    ???? FFFF  ?? XF  1 Z Z
000013  NOOP    ??Z? XXFX  ?? FX  Z 1 Z
000014  NOOP    ?ZZ? FFFF  ZZ FF  Z Z Z
000015  NOOP    ??ZZ FFXF  ?? FX  Z 0 Z
000016  NOOP    ??ZZ FFXX  ?Z XF  Z Z Z
000017  NOOP    ZZ?Z FXXX  ?? FF  Z Z Z
000018  NOOP    Z??Z FFXX  ?? XF  Z Z Z
000019  NOOP    ?Z?Z FFFX  ?? FF  Z Z Z
00001A  NOOP    ?ZZZ FFXX  ?? FF  Z Z Z
00001B  NOOP    ZZ?? FFXF  ?? XF  0 Z 1
00001C  NOOP    ???Z FFFX  ZZ FX  1 1 Z
00001D  NOOP    ?Z?Z XXFX  ?? XX  Z Z Z
00001E  NOOP    Z??? FFFX  ?? FF  Z Z 1
00001F  NOOP    ???? XFFF  ?? XF  Z Z Z
000020  NOOP    ZZ?? FFFF  ?? FX  Z 1 Z     :Write_2_Pages
000021  NOOP    ?ZB? FXXF  ?? FX  0 1 1     loop PageSize
000022  NOOP    ??Z? FFFF  ?? FX  0 1 Z
000023  NOOP    ?ZZ? FFFF  Z? XX  1 Z Z
000024  NOOP    Z??Z FFXX  ?Z XX  1 1 Z     control WRITE, addr = @ + 1, DATA = ~@
000025  NOOP    ?Z?Z FXFX  ?Z XX  Z Z Z     loop PageSize
000026  NOOP    ???? FFXF  ?Z XF  Z Z Z
000027  NOOP    Z??? XXXX  ?? FF  Z Z 1
000028  NOOP    ???? FFXF  ?? FF  1 Z Z     control WRITE, addr = @ + 1, DATA = ~@
000029  NOOP    ???Z FFXX  Z? FX  Z Z Z     return
00002A  NOOP    ?Z?? FXXF  ?? FF  1 1 0
00002B  NOOP    Z?ZZ FFFX  ?Z FF  Z Z Z     :Read_2_Pages
00002C  NOOP    ZZ?Z FFFF  ?? FX  Z Z 1     loop rd_count jmp
00002D  NOOP    ?Z?? XXXX  ?? FX  Z 1 Z
00002E  NOOP    ???Z XFXX  Z? FF  Z Z Z     control READ, compare InitData, addr = @ + 1
00002F  NOOP    ZZ?? XFXX  ZZ XF  Z Z Z     control READ, compare Complement, addr = @ + 1, jmp -1
```

Vector Window After HATGOL Button is Pressed

After making any modifications, merely press the 'Translate' button labeled [HT] on the main toolbar (top of window). This action will cause the HATGOL to be translated into vectors, as seen on the next page. The HATGOL program can be edited at any time for changes, but you must hit the "HT" button again to translate your changes into vectors.

```
                                        /
                                        W
                                        R
 Vector                                 I  / /
 Address      Program                   T  O C
 (Hex)                   Address Data    E  E S    Comments

 000000  NOOP            0000 XXXX 00 XX 1 1 1     HATGOL MemoryTest Checkerboard
 000001  NOOP            0000 XXXX 00 XX 1 1 1
 000002  JMP     0100    0000 XXXX 00 XX 1 1 1     jmp $Start
 000003  NOOP            0000 XXXX 00 XX 1 1 1     consign 3 4 5
 000004  NOOP            0000 XXXX 00 XX 1 1 1     #conf width 8
 000005  NOOP            0000 XXXX 00 XX 1 1 1     #define depth 32K-1
 000006  NOOP            0000 XXXX 00 XX 1 1 1     #define PageSize 64-1
 000007  NOOP            0000 XXXX 00 XX 1 1 1     #define repeats 512/2-1
 000008  NOOP            0000 XXXX 00 XX 1 1 1     #define rd_count 64/2-1
 000009  NOOP            0000 XXXX 00 XX 1 1 1     #define WRITE 0b_010
 00000A  NOOP            0000 XXXX 00 XX 1 1 1     #define READ 0b_100
 00000B  NOOP            0000 XXXX 00 XX 1 1 1     #define default 0b_111
 00000C  NOOP            0000 XXXX 00 XX 1 1 1     #define InitData 0
 00000D  NOOP            0000 XXXX 00 XX 1 1 1     #define Complement 0xff
 00000E  NOOP            0000 XXXX 00 XX 1 1 1
 00000F  NOOP            0000 XXXX 00 XX 1 1 1     control default all
 000010  NOOP            0000 XXXX 00 XX 1 1 1
 000011  NOOP            0000 XXXX 00 XX 1 1 1
 000012  NOOP            0000 XXXX 00 XX 1 1 1
 000013  NOOP            0000 XXXX 00 XX 1 1 1
 000014  NOOP            0000 XXXX 00 XX 1 1 1
 000015  NOOP            0000 XXXX 00 XX 1 1 1
 000016  NOOP            0000 XXXX 00 XX 1 1 1
 000017  NOOP            0000 XXXX 00 XX 1 1 1
 000018  NOOP            0000 XXXX 00 XX 1 1 1
 000019  NOOP            0000 XXXX 00 XX 1 1 1
 00001A  NOOP            0000 XXXX 00 XX 1 1 1
 00001B  NOOP            0000 XXXX 00 XX 1 1 1
 00001C  NOOP            0000 XXXX 00 XX 1 1 1
 00001D  NOOP            0000 XXXX 00 XX 1 1 1
 00001E  NOOP            0000 XXXX 00 XX 1 1 1
 00001F  NOOP            0000 XXXX 00 XX 1 1 1
 000020  NOOP            0000 XXXX 00 XX 1 1 1     :Write_2_Pages
 000021  LOADX     00    0000 XXXX 00 XX 1 1 1     loop PageSize
 000022  LOAD    003F    0000 XXXX 00 XX 1 1 1
 000023  DEC             0000 XXXX 00 XX 1 1 1
 000024  LOOP@           A1A1 XXXX B0 XX 0 1 0     control WRITE, addr = @ + 1, DATA = ~@
 000025  LOADX     00    0000 XXXX 00 XX 1 1 1     loop PageSize
 000026  LOAD    003F    0000 XXXX 00 XX 1 1 1
 000027  DEC             0000 XXXX 00 XX 1 1 1
 000028  LOOP@           A1A1 XXXX B0 XX 0 1 0     control WRITE, addr = @ + 1, DATA = ~@
 000029  RET             0000 XXXX 00 XX 1 1 1     return
 00002A  NOOP            0000 XXXX 00 XX 1 1 1
 00002B  NOOP            0000 XXXX 00 XX 1 1 1     :Read_2_Pages
 00002C  LOADX     00    0000 XXXX 00 XX 1 1 1     loop rd_count jmp
 00002D  LOAD    001F    0000 XXXX 00 XX 1 1 1
 00002E  DEC             A1A1 XXXX ZZ 00 1 0 0     control READ, compare InitData, addr = @ + 1
 00002F  CJMP    002E    A1A1 XXXX ZZ FF 1 0 0     control READ, compare Complement, addr = @ + 1, jmp -1
```

Vector Window After HT Translate Button is Pressed

Memory Test vectors are actually more like instructions for the HiLevel pin electronics, so at first they do not look like what you may expect. When you click your RUN button, the actual (and more meaningful) vectors can be seen in the Analysis window, where you'll see actual memory addresses and data. By using this Algorithmic Instruction method, very large memories can be tested with just a few hundred vectors. After creating these vectors, be sure to upload the TRN and PRG files along with the Set file.

Also See:
QnApp #P1: **PinList Function**
QnApp #P36: **Memory Test Functional Description**
QnApp #P37: **Memory Test: Flash**
QnApp #P43: **Memory BitMap**