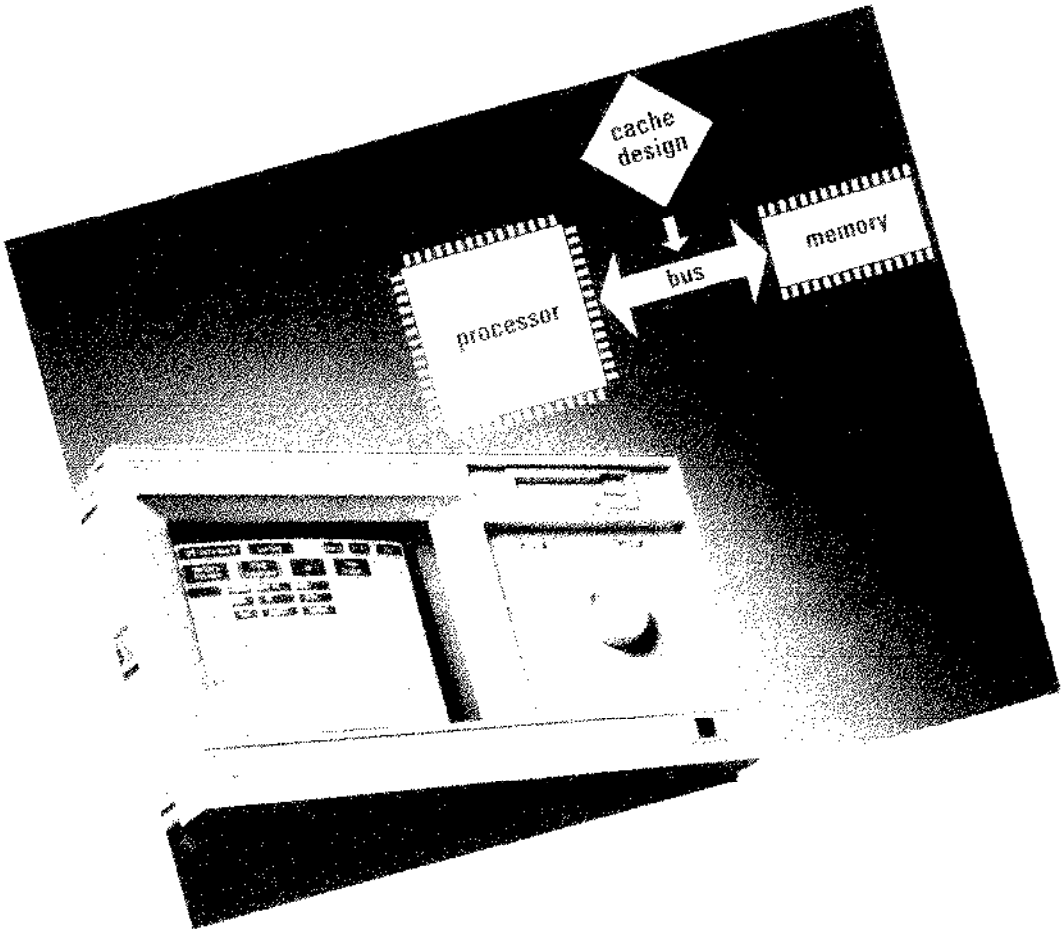
HEWLETT PACKARD

Cache Hit or Miss Analysis with the HP 16542A



Application Note 1225-2

Cache Hit or Miss Analysis with the HP 16542A

A cache can speed up memory-intensive systems significantly, however, an efficient cache that enhances system performance is an outcome of a good design model. This application note discusses some of the problems encountered when capturing system data needed to determine the size and makeup of the cache appropriate to your system. It also explores how to capture contiguous 1 Meg blocks of data using two HP 16542A data acquisition cards and one HP 16520A pattern generator card.

The pattern generator is used to count 1 Meg samples captured by the acquisition system. Once 1 Meg of address references have been sampled, the pattern generator immediately sends out a halt vector to the Motorola 68020 $\overline{\text{HALT}}$ line, which puts the processor in a halt state. The first 1 Meg block is then uploaded to a computer. Capturing an unlimited number of 1 Meg address references is accomplished by repeating the process until enough data is available for adequate analysis by a cache simulation program.

Advantages of Increasing Cache-Hit Efficiency

While it may seem obvious that a properly designed cache can significantly increase system performance, it's helpful to see some indications of how significant these effects are. The following simple formula of an average READ-cycle time gives an objective idea.

$$t_r = t_c + (1\text{-}h)\, t_m$$

where: t_r = read cycle time[1]
t_c = cache cycle time
t_m = main memory cycle time
h = probability of address reference being in cache (cache hit)
(1-h) = probability of address reference not in cache (cache miss)

If the average main memory cycle time is 3 times the cache cycle time, a decrease of 2% in the cache miss probability (1-h) will result in a 6% decrease in system cycle time (t_{cycle}). When evaluating a cache design, the rewards of increasing cache efficiency can be great.

[1] This formula assumes an external cache, and that the cache controller only initiates an external (main memory) cycle when a cache miss occurs.

Cache Performance: Size vs. Misses

In general, the performance of a cache follows the exponential curve shown in figure 1. However, due to the system's workload and the architecture of the cache, the knee of the curve can vary along the *cache size* axis making it difficult to put quantities on either axis. Part of a cache designer's task is to add quantities to the axes relating to their particular system.

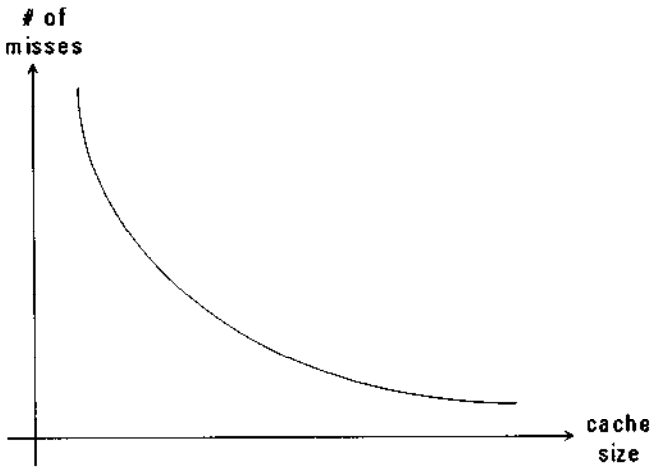


Figure 1. Number of Misses Relative to Cache Size

Capturing Data for a Cache Simulation Program

A cache's efficiency in any system relies on the following factors:

- Absolute size
- Set associativity
- Number of lines/set
- Number of bytes/line

A cache simulation program can analyze system data (that is, address and type of microprocessor cycle) and help to determine the best cache architecture for any system. The results of the cache simulation are closely tied to the ability to operate on data that accurately represents actual system workloads.

Two methods for generating data for a cache simulation program are:

- Artificially-generated data from a statistical model
- Data captured from an actual system

Artificially-generated data may help to model some simple systems, but these data are only approximations of actual operations. While considerable research has been done on modeling a cache, factors such as workload are difficult to quantify and cannot be approximated with any accuracy. Artificially-generated data is used when actual data from the system is not available.

If the actual system (that is, one to which you can make physical connections) data is available, the next problem is the measurement system. Two major constraints of a measurement system are:

1) Capture speed
2) Capture depth (how much data can the measurement system store?)

It's fairly easy to determine necessary capture speed by using the bus-cycle speed of the target system. But the amount of data needed is more difficult to determine. The following equations, courtesy of Dr. Harold S. Stone[2], give a starting point. For example, a 32 Kbyte cache has the following parameters:

cache size = 32 Kbytes
set associativity = 4 way
line width = 16 bytes
number of sets = 512
miss ratio = 3 %

If you assume you must capture enough address references to get 100 misses per set, the minimum number of address references is:

$$\frac{\text{100 misses/set}}{\text{0.03 miss ratio}} = \text{3333 hits/set}$$

3333 hits/set × 512 sets = 1.7M address references[3]

If you increase the cache size to 128 Kbytes, increase the number of sets to 2048, and assume that your miss ratio will drop by a factor of 2 for the larger cache, your estimate becomes:

$$\frac{\text{100 misses/set}}{\text{0.015 miss ratio}} = \text{6667 hits/set}$$

6666 hits/set × 2048 sets = 13.7 M address references[3]

---

[2]Dr. Harold S. Stone, High-Performance Computer Architecture, Addison Wesley, copyright 1990, pages 47 - 48.

[3]Each address reference contains multiple bytes dependent on the number of bits of address and function code captured. If each address reference consists of 4 bytes of address and 2 bytes of status information, each address reference in this equation should be multiplied by 6 to get the total amount of storage space needed by the host computer.

The problem of capture depth may be further compounded by the need to capture 32 or more channels of address references, depending on the requirements of the cache simulation program. For each of the address references, address and function code information needs to be captured. This clearly requires a measurement system with deep memory.

Methods of Capturing Data

When capturing address references, there are two possible approaches. The first, illustrated in figure 2, is a sampling method. The data acquisition system takes some number of samples and uploads the data to a host computer. When the upload operation is complete, the next data acquisition takes place. With this sampling method you must take enough samples to be statistically valid. Because states executed during the upload operation are never captured, the statistical sampling method runs the risk of missing infrequent events and may give a less-than-ideal representation of actual system operations.

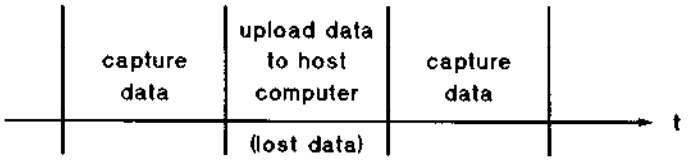


Figure 2. Sampling Method of Capturing System Data

The second alternative, illustrated in figure 3 and used in this application note, involves HALTing or holding off operation of the microprocessor while the measurement system transfers the captured data to the host computer. When the upload is complete, the microprocessor is released and resumes operation from the last executed cycle. The advantage of this method is that data is captured in a contiguous fashion, not in a sampled manner. This means you can capture as much data as necessary, limited only by the amount of storage space available to the host computer. This method also ensures the capture of infrequent events.

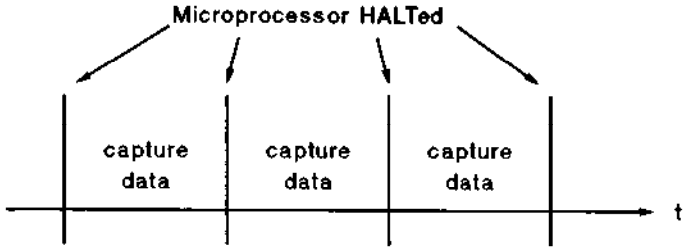


Figure 3. Contiguous Method of Capturing System Data

Measurement Description

For the measurements made in this application note the Motorola 68020 microprocessor was used because it is HALTable. Most Intel microprocessors cannot be operated in a halt mode in spite of the fact that they have a HOLD line. When the HOLD line on an Intel microprocessor is asserted, another device must take over as bus master

within one clock cycle. If a master does not take control of the address bus within this clock cycle, the address bus is tri-stated and any device on the bus (including the microprocessor) is placed in an indeterminate state. Once the HOLD line is deasserted, the startup vector may be lost, resulting in lost information.[4]

The process described in the remainder of this application note is based on capturing contiguous 1 Meg x 32 (channel) blocks, using two HP 16542A, 2-Mbyte data acquisition cards. An HP 16520A pattern generator card is used to assert the $\overline{\text{HALT}}$ line on the 68020 and to count states for the acquisition cards, as shown in figure 4.

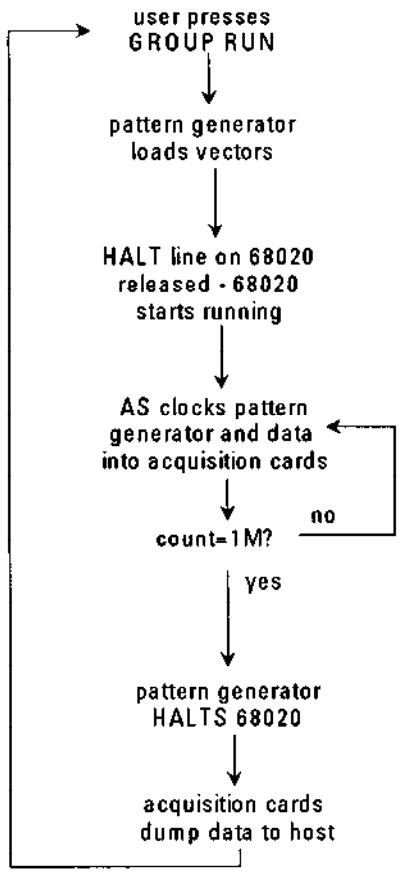


Figure 4. Measurement Overview

[4]An alternative for Intel processors is to use a non-maskable interrupt that forces a jump to a routine of NOPs.

For each 1 Meg of address references, you need only press the GROUP RUN field on the HP 16500A. Note that each 1 Meg trace from the acquisition cards sent to the computer is 4 Mbytes long because each sample is 4 bytes wide.

Equipment Used

Figure 5 is a diagram of the measurement equipment used for this measurement.

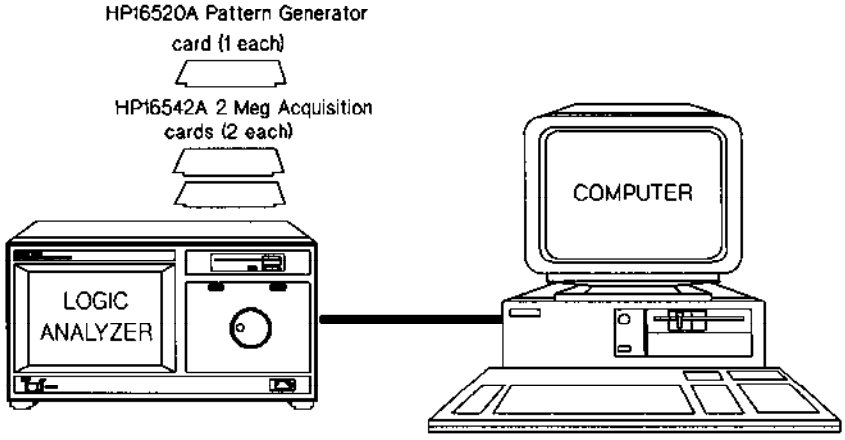


Figure 5. Necessary Measurement Equipment

Connection and Set Up Specifics

Figure 6 shows the necessary connections from the measurement equipment to the target system. The pattern generator is used to control whether the target system is running or HALTed. The pattern generator also counts the number of output states because the acquisition cards can't count the states captured.

Notice that the single line from the pattern generator to the 68020 $\overline{\text{HALT}}$ line has a delay circuit. This is necessary because of the way the pattern generator loads and then outputs its vectors. The pattern generator loads its first vector and outputs this vector while it is loading the rest of its memory. It takes the pattern generator about 800 ms to load the 4k vectors necessary for this measurement. The delay circuit gives the pattern generator time to load its vectors before releasing the 68020 $\overline{\text{HALT}}$ line.[5]

[5] If the measurement were made without the delay circuit the pattern generator would output its first vector (HALT deasserted), which would immediately put the 68020 in a RUN state. This, in turn, would result in data being acquired by the HP 16542A before the pattern generator is ready to begin counting states for the HP 16542A. The end result would be noncontiguous blocks of data acquired by the HP 16542A.

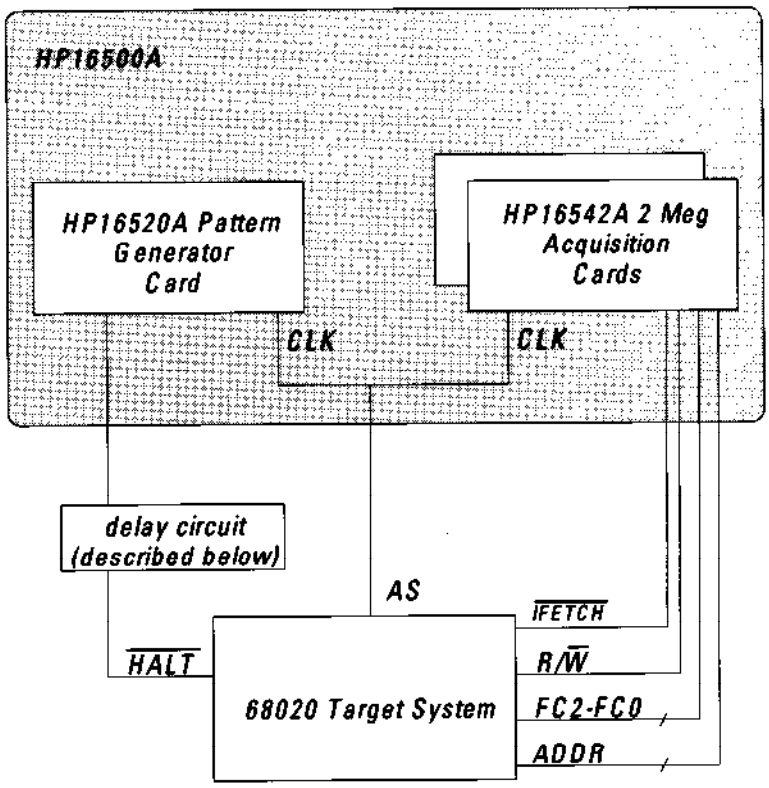


Figure 6. Connections to Target System

A schematic of the delay circuit is shown in figure 7. The circuit delays the RUN ($\overline{\text{HALT}}$ deasserted) signal from the pattern generator by more than 1 second to allow the pattern generator to load all of its vectors before it runs. The inverters are used to 'square up' the signal after each RC integrator and to prevent glitches from being propagated through the circuit.[6]

[6]Any circuit with a delay of 1 second or more will work as long as it does not produce glitches.

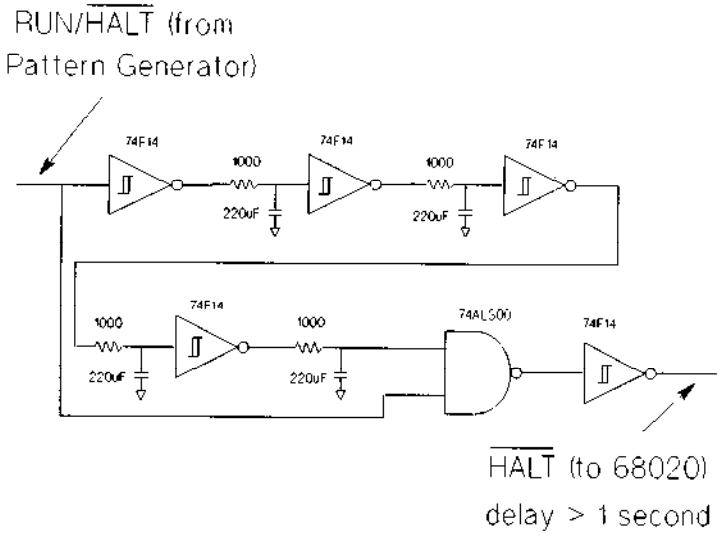


Figure 7. Delay Circuit for HP 16520A RUN/HALT Vector

Figure 8 is a timing diagram showing the RUN line from the pattern generator and the affects on the HALT line to the 68020 and the AS line. The AS line is used to clock the pattern generator and the acquisition cards.

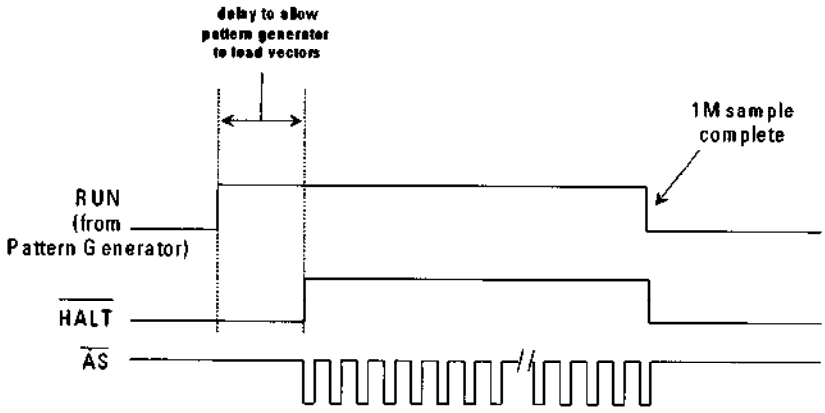


Figure 8. Timing Diagram of Input and Output of Delay Circuit

The physical connections to the HP 16500A analyzer cards are as follows:

HP 16520A Pattern Generator Card

input of delay circuit	⇒	Pod 3 channel 0
AS from 68020	⇒	EXT CLK of input pod

HP 16542A Data Acquisition Card

68020 address lines	⇒	Pod D channels 0 - 15 and Pod C channels 0 - 7
FC2 - FC0	⇒	Pod C channels 8 - 10
R/W	⇒	Pod C channel 11

The menu setups for the data acquisition and pattern generator cards are shown in figure 9 through figure 11:

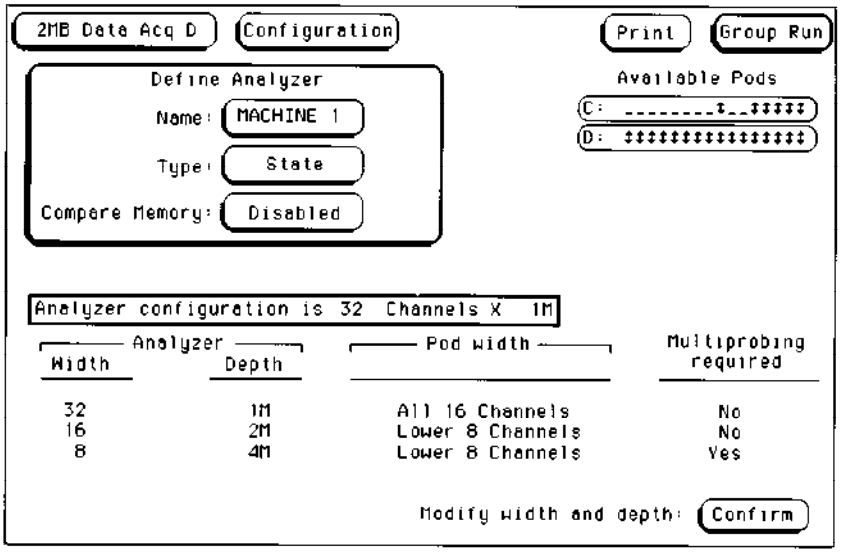


Figure 9. Acquisition Card Configuration Menu Set Up

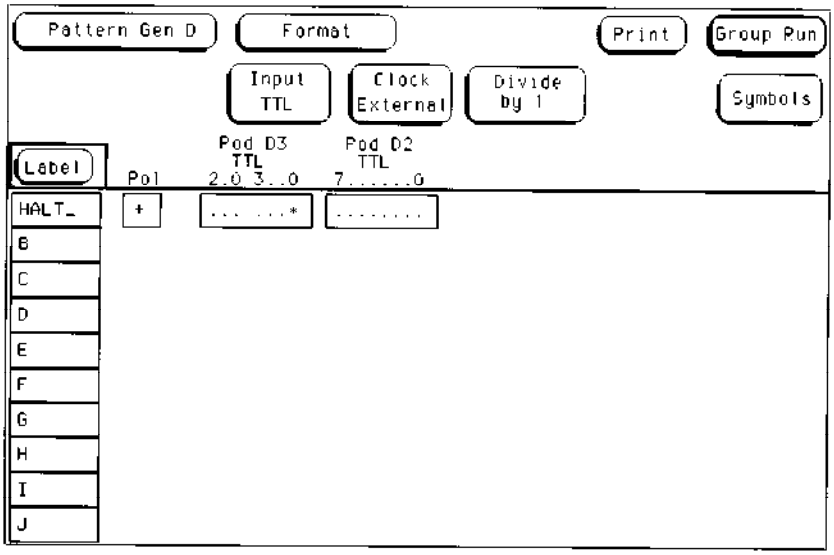


Figure 10. Pattern Generator Format Menu

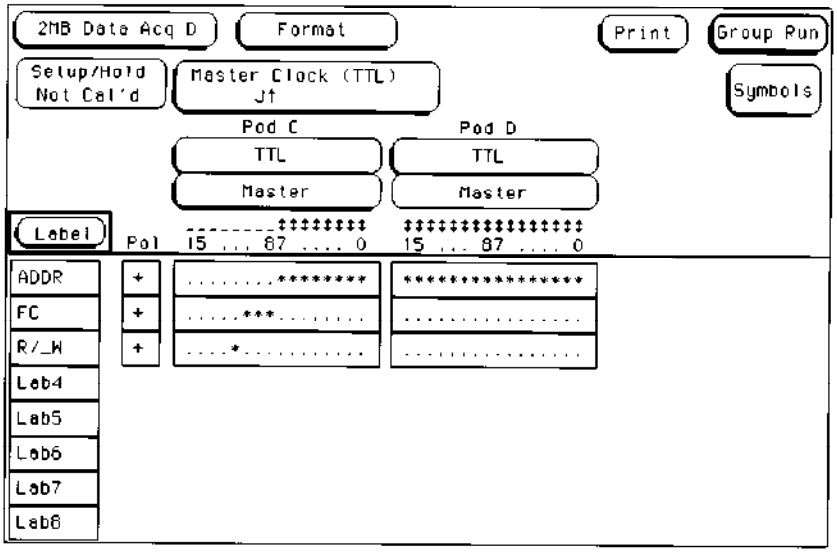


Figure 11. Acquisition Card Format Menu

Once the physical connections are made and the channels assigned for the acquisition and pattern generator cards, the pattern generator must be programmed to output the vectors to control the $\overline{\text{HALT}}$ line of the 68020. The measurement requires 1 M vectors for each run. The start of program and end of program listing for the pattern generator is shown in figure 12 and figure 13.

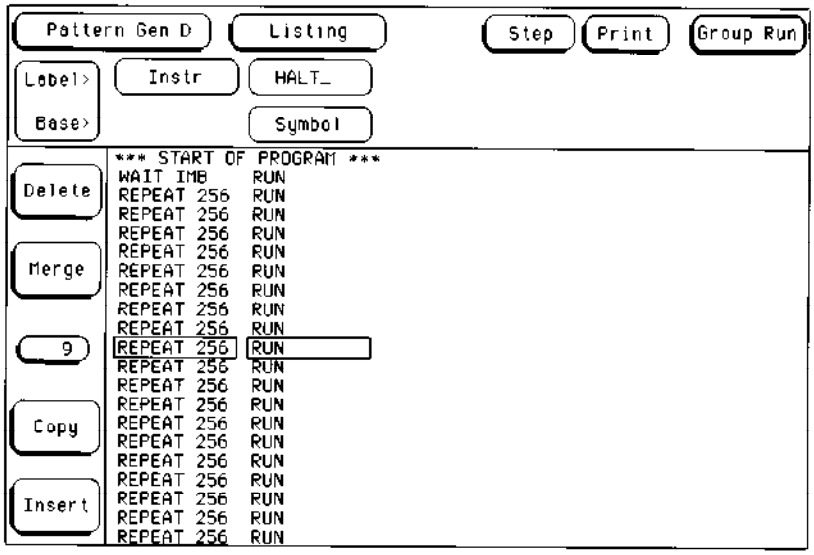


Figure 12. Start of Pattern Generator Listing

The first line in the pattern generator listing is WAIT IMB (Wait Intermodule Bus). For this measurement, the WAIT IMB line tells the pattern generator to wait for the arm from the acquisition card before stepping through the remainder of the vector listing. This arm will occur when the trigger condition is satisfied in the HP 16542A data acquisition card. In this example, the trigger condition is the first state seen by the data acquisition card.

Note that each step in the pattern generator listing, except for the first and last steps, is a REPEAT statement. Each REPEAT 256 tells the pattern generator card to keep the microprocessor in the RUN state for the duration of 256 AS External Clock cycles. The end result of these REPEAT statements is that the microprocessor is allowed to RUN for 1047546 Read/Write cycles (REPEAT 256 * 4091 + REPEAT 250). The number of cycles counted by the pattern generator REPEAT statements is approximately equal to the number of states acquired in the data acquisition card.

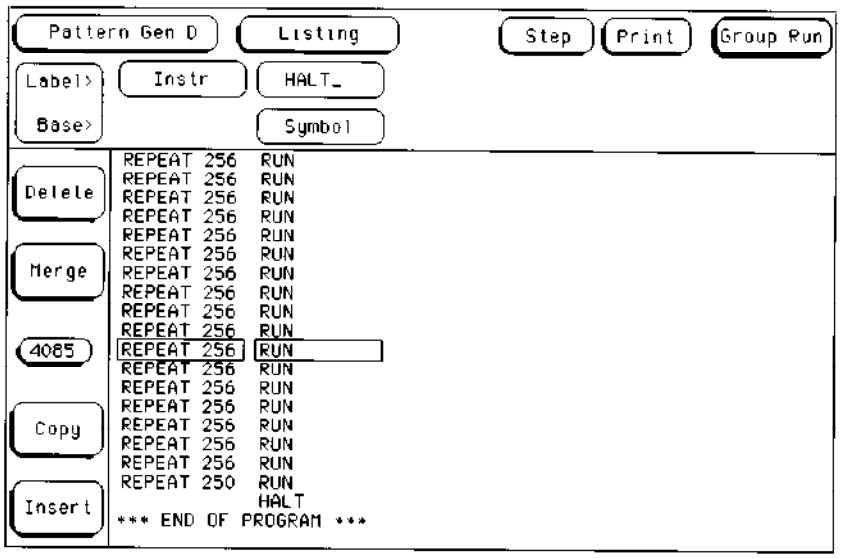


Figure 13. End of Pattern Generator Program Listing

The last line of the pattern generator listing, as seen in figure 13, is HALT. When this vector is sent out, the 68020 $\overline{\text{HALT}}$ line is asserted and all microprocessor activity is halted. This in turn halts all bus activity between main memory and the microprocessor. When the $\overline{\text{HALT}}$ occurs, the HP 16542A will have completed an acquisition cycle.

Once the pattern generator listing is entered, the next step is to set up the trigger conditions for the data acquisition cards. Because you are interested in all states generated by the 68020, the trigger condition will be DON'T CAREs, and the data captured will be any state as shown in figure 14.

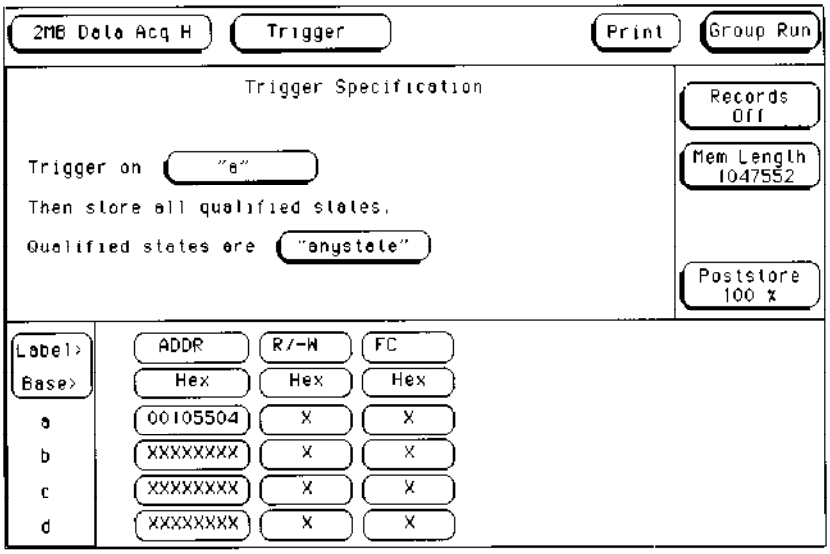


Figure 14. Acquisition Card Trigger Menu

The final step is to set up an intermodule sequence that allows the pattern generator and acquisition cards to work together in a single acquisition run. Figure 15 shows the Intermodule menu which is set up to have the HP 16542A 2-Mbyte data acquisition card arm the HP 16520A pattern generator card when the trigger condition is found.

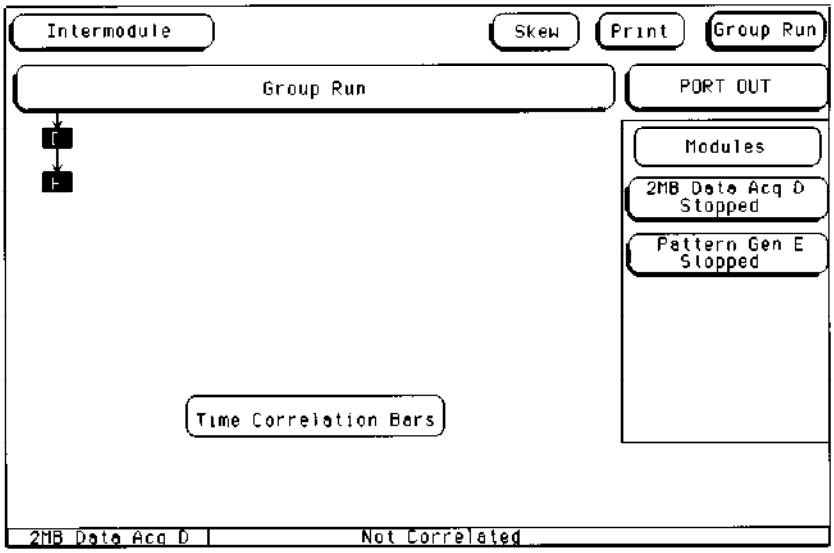


Figure 15. Intermodule Bus Menu Set Up

At this point all of the setups and configurations are complete and the measurement is ready to run. Pressing the Group Run will cause the following sequence to occur. First, the pattern generator will hold the microprocessor in the RUN state. This will result in AS becoming active. When AS asserts for the first time, the first valid address is sampled by the data acquisition card. Because the trigger condition on the HP 16542A is a DON'T CARE, this first sample is also recognized as the

trigger condition and immediately sends out an arm signal to the pattern generator. The first line of the pattern generator listing, WAIT IMB, detects the arm which, in turn, allows the remainder of the vectors to be sent out on each external clock cycle (rising edge of AS). The pattern generator finally sends out the last vector which asserts the $\overline{\text{HALT}}$. This results in the assertion of the 68020 $\overline{\text{HALT}}$ line. At this time approximately 1M of valid addresses have been captured by the HP 16542A data acquisition card. A typical data capture is shown in figure 16.

Note that the total number of clocked pattern generator vectors is equal to 1047548 (REPEAT count plus two for the first and last vector). This is slightly less than the **Mem Length**, 1047552, specified in the 2 MB Data Acq Trigger menu. The reason for this difference is because there is some uncertainty as to the exact time that the microprocessor will assert $\overline{\text{HALT}}$ once the pattern generator sends out the HALT vector. This uncertainty can result in an indeterminate number of states being stored between acquisitions.

In this measurement, when running multiple acquisitions, the total number of states acquired between runs only varied by one state. To avoid the possibility of missing any clocked states, the measurement should be set up so that the microprocessor always halts before the HP 16542A acquisition is allowed to complete. This will ensure that contiguous 1M state blocks are acquired. Because the acquisition is never allowed to complete, it is necessary to **Stop** the acquisition run before you upload the data to a computer.

As an example, from our setups, the total number of states captured by the HP 16542A acquisition card is 1047551 (states 0-1047550). Refer to figure 16. This state count is one state less than specified in the **Mem Length** field, figure 14. Basically, the $\overline{\text{HALT}}$ line from the pattern generator is programmed to send out the $\overline{\text{HALT}}$ vector before the HP 16542A card has completed its acquisition. Again, this is done to prevent missed states.

If you were to increase the total number of pattern generator vector counts so that **Mem Length** states are acquired, that is, set the last REPEAT count to greater than or equal to 251, you would capture the specified **Mem Length** states. But, you would risk missing a few states after the HP 16542A has filled its memory. If a few lost states are not a concern, set the REPEAT count so that the acquisition always completes.

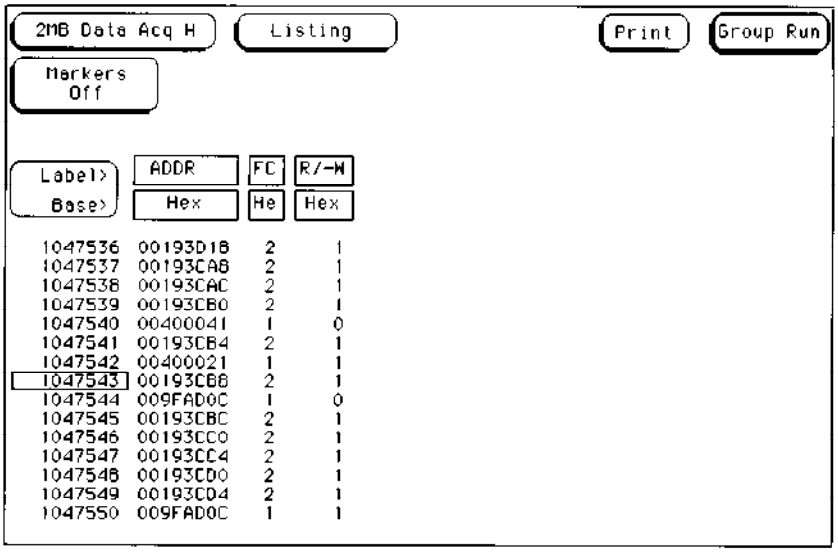


Figure 16. Example Data Capture From 68020 Target System

The 1 Mbyte trace can then be uploaded to a computer, via HP-IB, using the program at the end of this application note. To capture subsequent 1 Mbyte blocks, you need to repeat the process of pressing Group Run and uploading data, until the desired number of contiguous 1 Mbyte blocks is acquired. The data that has been uploaded to the computer can then be used as input to the cache simulation program.

Conclusion

Capturing actual system data for cache analysis has been difficult, if not impossible, in the past. However, using statistically-generated data is often less desirable due to the number of system characteristics that cannot be quantified statistically.

The HP 16542A deep memory data acquisition card has the capture speed necessary to get system data, and it has the deep memory needed to capture large blocks of data. This is particularly important given the amount of data it takes to adequately quantify a system cache. After capturing the data, it is easily transferred to a host computer for analysis by a cache simulation program.

Example Programs

Not shown in these programs are the HP-IB communication procedures: Init_IO, Read_IO, Write_IO, and Close_IO. See the Examples Chapter of the HP 16542A Programmer's Guide for details of these procedures.

```
/* Upload Programming Example */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_SIZE 16000  /* Maximum buffer size for
                           Read_IO and Write_IO */
#define TRUE 1
#define FALSE 0

/* IO prototypes */

int Init_IO( void );
int Read_IO( int, char *, size_t );
int Write_IO( int, char *, size_t );
int Close_IO( int );

#define CARDC 3      /* HP 16542A occupies card slot C */

void Initialize_16500( void );
int Select( int );
int Read_data( const char * );

void
main( void )
{

   /* Initialize IO port */

   hpib_id = Init_IO();

   /* Initialize the 16500 logic analyzer */

   Initialize_16500();

   /* Select the HP16542A card */

   Select( CARDC );

   /* Upload the captured Cache data from the analyzer
    * and store it in a file named "cache.dat"
    */

   Read_data( "cache.dat" );

   /* Close the IO port */

   Close_IO( hpib_id );
}
```

```
/*****************************************************
 * Function Name: Initialize_16500                   *
 * Passed Parameters: None                           *
 * Return Value: None                                *
 * Description:                                      *
 *   The function initializes the HP16500 logic      *
 * analyzer by clearing all status registers and     *
 * enabling all Service Request Status and Standard  *
 * Event Status register bits.                       *
 *****************************************************/
void
Initialize_16500( void )
{
   char id[80];
   int term;

   /* Clear the 16500 */

   Write_IO( hpib_id, "*CLS", 4 );

   /* Enable all Service Request Status Register bits */

   Write_IO( hpib_id, "*SRE 255", 8 );

   /* Enable all Standard Event Status Register bits */

   Write_IO( hpib_id, "*ESE 255", 8 );
}

/*****************************************************
 * Function Name: Select                             *
 * Passed Parameters: card_number - integer          *
 * Return Value: TRUE - successful completion        *
 *               FALSE - unsuccessful completion     *
 * Description:                                      *
 *      The function selects which module in the     *
 * HP16500 card cage receives commands.  This function *
 * must be invoked before sending any commands to the  *
 * HP16542A 100MHz State Analyzer Card.              *
 *****************************************************/
int
Select( int card_number )
{
   char command[10];

   sprintf( command, ":SELECT %d", card_number );

   Write_IO( hpib_id, command, strlen( command ) );

   return( TRUE );
}
```

```
/*********************************************************
 * Function Name: Read_data                              *
 * Passed Parameters:                                    *
 *     data_file_name - char pointer to the name of the  *
 *                      data file to which the binary    *
 *                      data read from the logic         *
 *                      analyzer is stored.              *
 * Return Value: TRUE - successful completion.......... *
 * Description:                                          *
 *      Open the file specified by the data_file_name    *
 * parameter and write the binary data from the logic    *
 * analyzer to this file.  Returns TRUE when all         *
 * information has been read.                            *
 *********************************************************/
int
Read_data( const char *data_file_name )
{
   char command[20];
   char buffer[MAX_SIZE];
   int bytes;
   FILE *data_file;

   data_file = fopen( data_file_name, "wb" );

   /* Turn off system header information
    * which is not required for binary
    * data upload
    */

   strcpy( command, ":SYSTEM:HEADER OFF" );
   Write_IO( hpib_id, command, strlen( command ) );

   strcpy( command, ":SYStem:DATA?" );
   Write_IO( hpib_id, command, strlen( command ) );

   bytes = Read_IO( hpib_id, buffer, (size_t) MAX_SIZE );
   while( bytes == MAX_SIZE )
   {
      fwrite( buffer, 1, sizeof( buffer ), data_file );
      bytes = Read_IO( hpib_id, buffer, (size_t) MAX_SIZE );
   }

   fwrite( buffer, 1, (size_t) bytes, data_file );

   fclose( data_file );
   return( TRUE );
}
```

HEWLETT PACKARD

For more information, call your local HP sales office listed in your telephone directory or an HP regional office listed below for the location of your nearest sales office.

United States of America:
Rockville, MD
(301) 670-4300

Rolling Meadows, IL
(708) 255-9800

Fullerton, CA
(714) 999-6700

Atlanta, GA 30339
(404) 980-7351

Canada:
(416) 678-9430

Japan:
(8113) 3335 8192

Latin America:
Mexico
(525) 202-0155

Brazil
(11) 709 1444

Australia/New Zealand:
(03) 895-2895

Far East:
Hong Kong
(852) 848-7070
Korea
(2) 769 0800
Taiwan
(2) 717 9524
Singapore
(65) 291 8554
India
(11) 690 355
PRC
(1) 505-3888

In Europe, Africa and Middle East, please call your local HP sales office or representative:

Austria/South East Area:
(0222) 2500-0

Belgium and Luxembourg:
(02) 761 31 11

Denmark:
(45) 99 10 00

Finland:
(90) 88 721

France:
(1) 69.82.65.00

Germany:
(06172) 16 0

Greece:
(01) 68 28 811

Ireland:
(01) 2844633

Israel:
(03) 5380 333

Italy:
(02) 95 300 930

Netherlands:
(020) 547 6669

Norway:
(02) 87 97 00

Portugal:
(11) 301 73 30

South Africa:
(011) 806 1000

Spain:
900 123 123

Sweden:
(08) 750 20 00

Switzerland:
(057) 31 21 11

Turkey:
(90-1) 4 125 83 13

United Kingdom:
(0344) 362 867

For countries not listed, contact Hewlett-Packard, International Sales Branch, Geneva, Switzerland
Tel: +41-22-780-7111
Fax: +41-22-780-7535

Technical information in this document is subject to change without notice

Printed in U.S.A.
September 1992
5091-5446E