SCI Europe number: P25257
Deliverable number: D2.2.1a
Contractual date: 30th September, 1998
Work package: 2.2.1

Document version: 2.0
Document status: Complete
Confidentiality: Consortium
Document date: October, 1998

Deliverable

Prototype Tracer

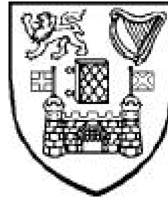
Partner: Trinity College Dublin
Author(s): B.A.Coghlan, M.Manzke, E.Barnstedt, R.Cunniffe, J.Dukes
Editor: B.A.Coghlan

Keywords: SCI, tracer, analyzer

Abstract:

In this document we present the technical manual for the prototype tracer developed within the project.

esprit project
25257



ESPRIT Project P25257 SCIEurope

Deliverable D 2.2.1a

Prototype Tracer

October 1998

Dr.B.A.Coghlan
Department of Computer Science
Trinity College Dublin
coghlan@cs.tcd.ie

M.Manzke
Department of Computer Science
Trinity College Dublin
michael.manzke@cs.tcd.ie

E.Barnstedt
R.Cunniffe
J.Dukes
Department of Computer Science
Trinity College Dublin

Introduction

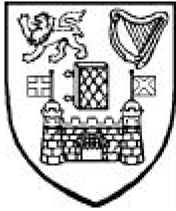
The definition of Task 2.2.1 is as follows :

Task	2.2.1	Test Tools Development		
Market and User Need	There are no commercially available SCI test tools on the market for the SCI community today.			
Objectives	To develop the first generation of tracing and debugging tools for use in work package 3 Applications.			
Approach	The tools will be based on needs identified in the Test Requirements Specification from Task 2.1. There will probably be developed two tools – one tool able to trace the SCI traffic and either show online or store the results. This tool will be based as much as possible on present hardware and software platforms. The other tool will be able to send and receive SCI traffic according to some traffic profile in order to load systems with traffic without using real nodes. The prototype tools will be evaluated during the debugging phase of the Embedded Avionics System demonstrator in Task 3.3, and the results will be summarised in a report.			
Lead Partner	Trinity	24 person months		
Other Partners	D.E. SINTEF	4 person months 12 person months		
Major deliverables	D 2.2.1 D 2.2.2 D 2.2.3	Q4 Q6 Q8	Trinity SINTEF Trinity	Prototype Tracer/Analyzer Traffic Generation Tool Tracer/Analyzer Mk.II

The objective of this document is to present the Technical Manual for the Prototype Tracer. This represents the hardware and software resources of the Prototype Tracer/Analyzer. The software for the Prototype Analyzer will be presented in a later document.

The Prototype Tracer is not specifically oriented towards SCI, but instead provides general purpose deep trace facilities. B-Link traces will be acquired via a probe card supplied by Dolphin, that attaches to their SCI cards via elastomeric connectors, and breaks out the B-Link signals to a number of connectors that will accept cables for a HP16500 series logic analyser. Trinity will design a probe adapter that will attach to these to distribute the B-Link signals to a multiple of the DT200.1 Deep Trace boards. Pending the probe adapter, the B-Link signals will be connected directly from the probe card to two DT200.1 boards; this will allow some experimentation into detailed trigger/trace needs using the trigger/trace facilities of the DT200.1 boards.

SCI Europe



Computer Architecture Group
Department of Computer Science
Trinity College Dublin
Ireland

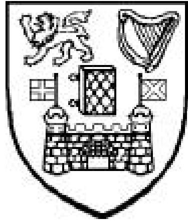
Tel : +353-1-6081765
Fax : +353-1-6772204

Deep Trace DT200.1

Technical Manual

B.A.Coghlan
P.O'Carroll
M.Manzke
E.Barnstedt
R.Cunniffe
J.Dukes

SCI Europe



Deep Trace DT200.1

Technical Manual

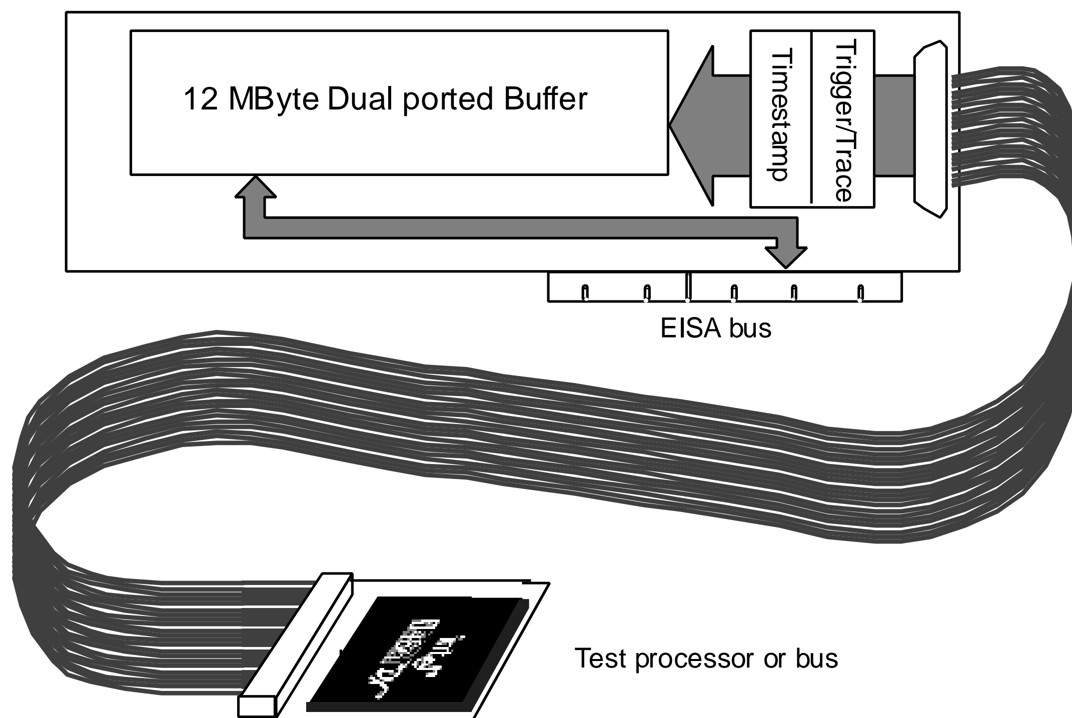
Introduction

Systems performance analysis involves the collection and analysis of traces obtained from a target computer system. By presenting views that illustrate the target's behaviour versus time, a designer can determine how efficiently the target system is operating. These views are traditionally in the form of histograms, charts or tables of statistics. Trace analysis can reveal many interesting system characteristics such as where a processor is spending most of its time, how long critical routines take to execute, how often a cache contains the desired information, how well a cache coherency protocol performs, whether there is excessive contention for locks, how well the load is balanced among processors, and so on. If performance bottlenecks can be located, corrective action can hopefully be taken.

Traditional logic analysers collect traces in real-time, but their capacity is generally limited to the order of 1to 8K samples. With time this is improving, for example, a new trace module has been introduced for the Hewlett Packard HP16500A logic analyser - a fully configured system comprising five HP16542A boards is capable of collecting 1,000,000 x 80bit samples @100MHz. Users cannot write analysis software for the HP16500A itself, but the contents of the trace memory can be transferred to a PC, albeit via a relatively slow RS232 or HP-IB interface for analysis.

The DT200.1 Deep Tracer is a modular data collection system designed sp ecifically for gathering very long state traces for performance analysis of processor and I/O busses. In a minimal configuration the data collection system consists of two modules, a sampler board which stores the trace and a preprocessor module to collect data from the target bus. This allows traces to be generated from a variety of busses by using different preprocessor modules. This document describes the technical details of the EISA based sampler board DT200.1.

Conceptually the system functions as a very deep FIFO. Data is sampled via the 48 bit serial interface on the positive edge of a clock and stored in a 12MByte VRAM buffer. The buffer wraps around and can be read out via the EISA interface without stopping the sample clock. A block diagram of the trace system is shown below.



Features

- 2 Million x 48bit samples
  12MBytes Dual Ported VRAM

- Max Sample Rate 50MHz on Serial Port
  Max data rate 300Mbytes/second.

- Standard EISA interface on Parallel Port
  33MByte/sec bursting slave allows data to be read out transparently and asynchronously without stopping trace. Autoinitialization on power up. Buffer can be mapped anywhere in EISA address space.

- Real Time Trigger and Trace functions
  64k x 12 Trace SRAM gives a minimum of four arbitrary trace ranges

- Expandable in Width and Depth
  An interboard connector allows multiple boards to cascaded to increase the sample width, in multiples of 48bits, up to 192bits wide. It is also possible to interleave boards to increase buffer depth in multiples of two million. This also increases the maximum sample rate.

- Integral Timestamping
  48bit 50MHz timestamp counter plus serial clock synchronizing logic. Interboard connector allows distributed synchronisation of multiple boards.

- Bidirectional
  Can be programmed to output samples synchronously with an external clock.

Programming Model

This section contains all the information necessary to write low level code to drive the sampler board. Sample code and header files are presented later in this document. It is recommended that driver software uses the code provided or other trusted routines to ensure the integrity of the collected traces.

Register Summary

The hexadecimal addresses of all the programmable registers are shown below. They are all mapped into Slot Specific EISA I/O space. The actual I/O address is calculated by substituting the Slot number (0..15) for the upper four bits. Slot number is represented by 'Z'.

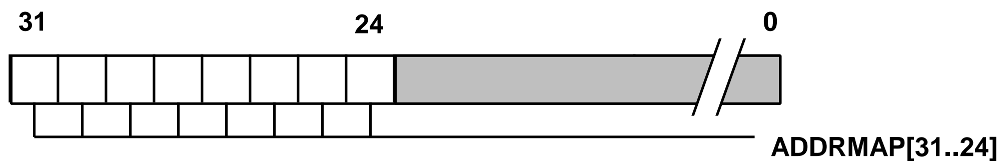
offset	mnemonic	r/w	operation
0xZ000	RESET	w	Hard Reset of entire Board
0xZ004	ADDRMAP	r/w	Write EISA Address Map
0xZ008	MODE	r/w	Mode Control Register
0xZ00C	TRIGCONFIG	r/w	Trigger/Trace Select Reg
0xZ010	TIMERESET	w	TimeStamp Counter Reset
0xZ014	STOPCOUNT	r/w	Trigger Row Counter
0xZ018	HEADPTR	r/w	Head Pointer Counter
0xZ01C	OUTPUTENA	r/w	Output Enables
0xZ020	STATUS	r	Trace Status
0xZ080	EISAID	r	EISA ID for Slot initialisation

0xZ000 RESET : Board Reset

Writing any value to this register resets all registers, timestamp counter, EISA interface and internal state machines. After a system reset or power-UP, the VRAM buffer will not respond until :

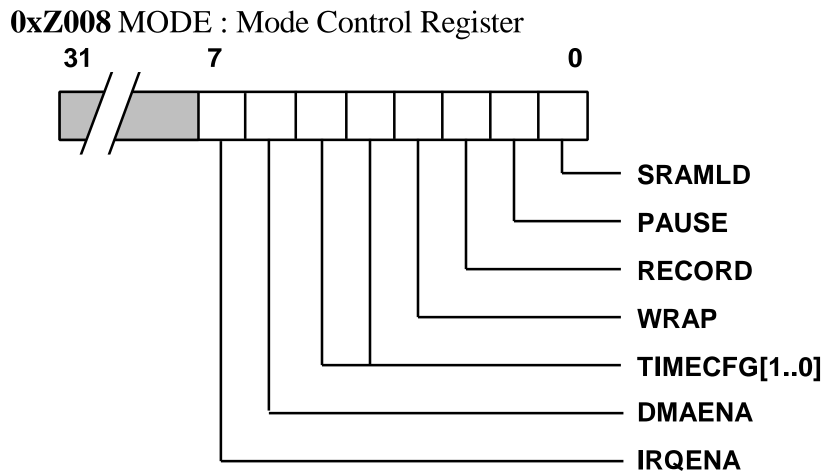
1. the board is RESET by a write to 0xZ000, and then
2. a write is made to any register but the RESET register.

0xZ004 ADDRMAP : EISA Address Map Register



This register specifies the upper 8 bits of the VRAM buffer address map in EISA memory space. It must be written before the buffer can be accessed. After a system reset or power-UP, the VRAM buffer will not respond until first the board is first RESET and then a write is made to any register but the RESET register.

0xZ008 MODE : Mode Control Register



There are 8 mode bits in this register - all are active when non zero. Currently the two most significant bits are not implemented.

SRAMLD enables the trace SRAM to be read and written. It disables sampling but does not stop the timestamp counter.

PAUSE stops the sampler. It is buffered on an open collector line which can be connected to other boards over the inter-board connector. This allows multiple boards to start and stop sampling together.

RECORD specifies the direction of the serial port. RECORD=1 means the board samples the serial bus synchronously with the serial clock. RECORD=0 causes the serial buffers to drive the contents of the sample memory onto the serial bus synchronously to the serial clock. Writing to this bit turns around the VRAM serial ports and resets the output buffer enables - hence it should be set to its intended value before the output buffer enables are configured, i.e. immediately after writing RECORD=1, the timestamp counter outputs will be disabled (but the counter itself is unaffected). Note that since RECORD=1 either enables either the sample input buffers or the timestamp counter outputs (depending upon OUTPUTENA[5..0]), the SRAM cannot be accessed while RECORD=1.

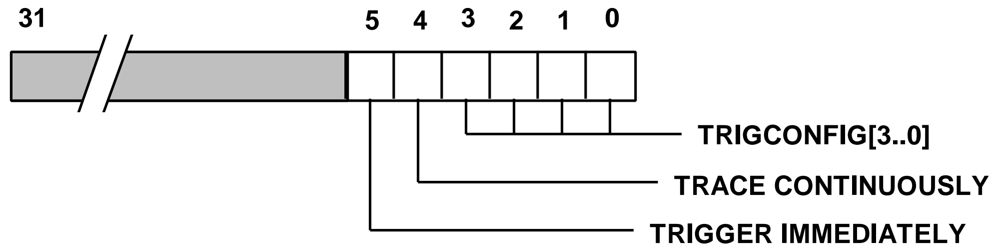
TIMECFG[1..0] specifies the divisor for the timestamp counter: 1, 2, 4 or 8.

WRAP causes the board to ingnore the STOPCOUNT counter and continue sampling when it reaches zero.

DMAENA enables the board to request the host machine to perform DMA on EISA DMA channel 7 when the STOPCOUNT counter reaches zero. Currently, DMAENA is not implemented.

IRQENA causes the board to assert EISA interrupt IRQ12. The interrupt is level triggered so that multiple boards can share the same interrupt. Currently, IRQENA is not implemented.

0xZ00C TRIGCONFIG : Trigger/Trace Channel Configuration

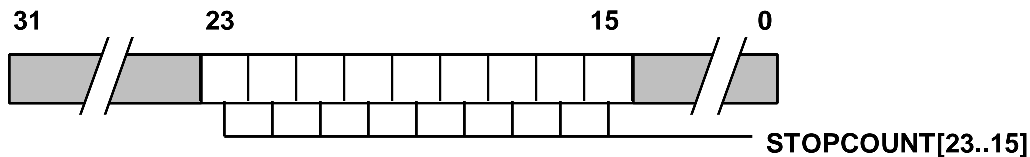


The four least sigificant bits select the four SRAM trigger/trace channels. A logic 0 in TRIGCONFIG[3..0] selects the channel as a trace and logic 1 selects it as a trigger channel. The TRACE CONTINOUSLY bit forces acceptance of every sample. The TRIGGER IMMEDIATELY bit forces the STOPCOUNT to begin counting down immediately.

0xZ010 TIMERESET : Reset TimeStamp Counter

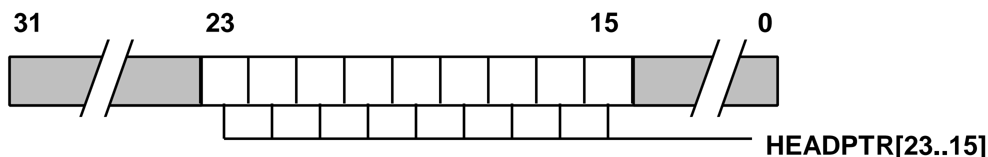
Writing any value to this location resets the timestamp counter. The signal is driven by an open collector buffer onto the inter-board connector so that multiple boards can be synchronised.

0xZ014 STOPCOUNT : Sample Stop Counter



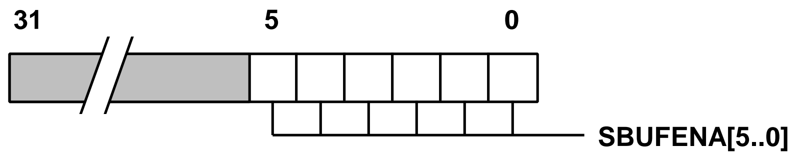
This register holds the absolute number of samples to be stored before the the sampling stops. The buffer stores samples incrementally in blocks of 2048x48 wrapping around to the base address when it reaches the top of the the buffer. As soon as the SRAM detects a trigger value the stop counter is enabled. The counter decrements with every second block of 2048 samples stored. Loading 0x00000000 into this register causes the board to stop immediately after the trigger so that the trigger value is at the end of the trace. Writing 0x00FF8000 into STOPCOUNT means the the trigger value will occur at the start of the trace. Bits [31..24] and [14..0] will always read as a logic 1. If it is possible to atomically update this register, then very long traces may be generated provided the data can be read out of the EISA interface as fast as it is acquired. Writing to this register will reset the timestamp counter.

0xZ018 HEADPTR : Head Transfer Pointer



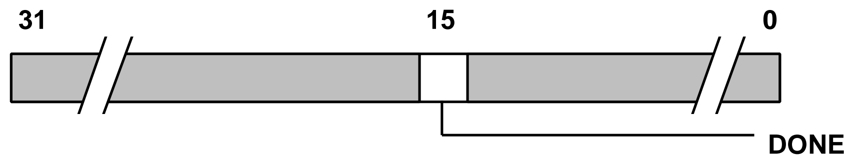
This location holds the 24 bit address offset where the next block of 2048 samples will be stored. Bits [31..24] and [14..0] will always read as a logic 1. Writing to this register will reset the timestamp counter.

0xZ01C OUPUTENA : Serial Buffer Output Enables



Each of the active bits in this register represents a different byte in the serial port. A logic 1 in OUTPUTENA[5..0] enables the corresponding byte[5..0] of the sample. Disabling an individual sample buffer enables the corresponding byte of the timestamp register, if the RECORD mode bit is set. As a precaution against damaging the buffers this register is reset every time the direction of the sample buffers is changed by toggling the RECORD bit., i.e. toggling RECORD turns around the VRAM serial ports and resets the output buffer enables - hence RECORD should be set to its intended value before the output buffer enables are configured.

0xZ020 STATUS : Trace Status Register



This read only register indicates whether the STOPCOUNT has reached zero. The DONE bit will be a logic zero when the STOPCOUNT counter is non-zero, and a logic 1 when the STOPCOUNT counter reaches zero. Bits [31..16] and [14..0] should always be treated as undefined values.

0xZ080 EISAID : Slot Specific EISA ID Register

This read only register contains the unique EISA ID for the board. This allows the board to be initialised on power up by the system board from initialisation values stored in the BIOS CMOS RAM. It also allows the driver software to scan the EISA slot specific I/O addresses to identify which slots contain trace boards, and therefore at what addresses. Currently this register reads out a value of `0x01209212`. A sample EISA configuration file is presented later in this document.

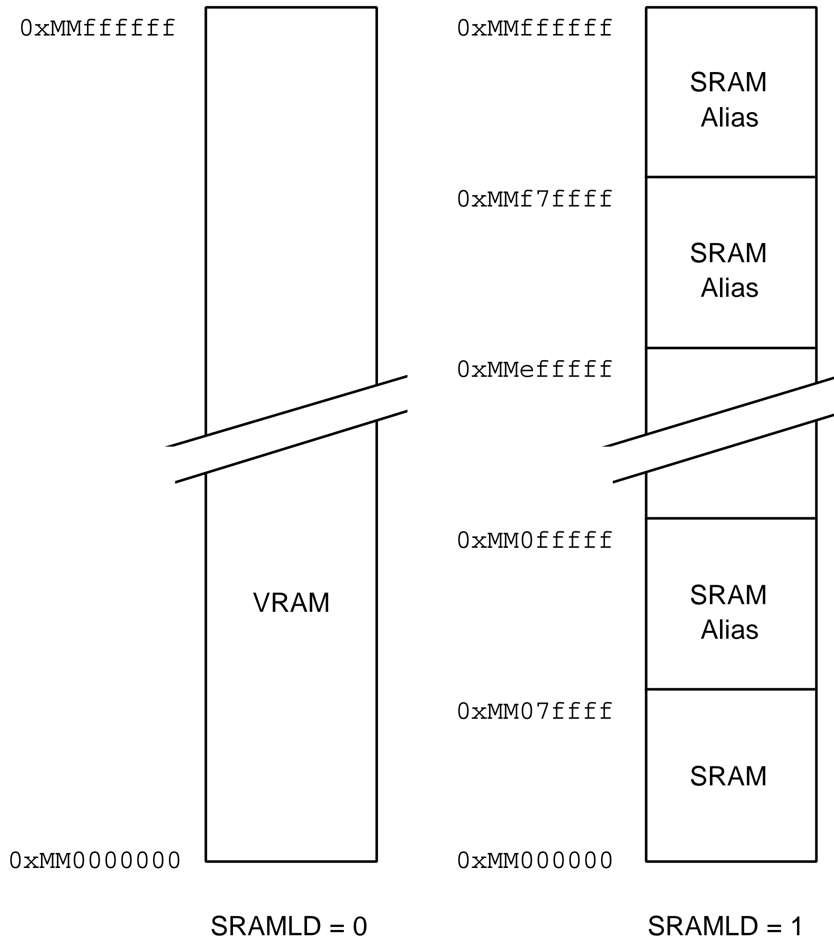
TimeStamping

There is a 48 bit synchronous timestamp counter which can be enabled byte by byte onto the sample data stream. OUTPUTENA[5..0] enables byte[5..0] of the counter. The bytes that are not enabled will count anyway. The cycle time is selected by the MODE[5..4] register bits. The following table indicates the clock period for each of the combinations:

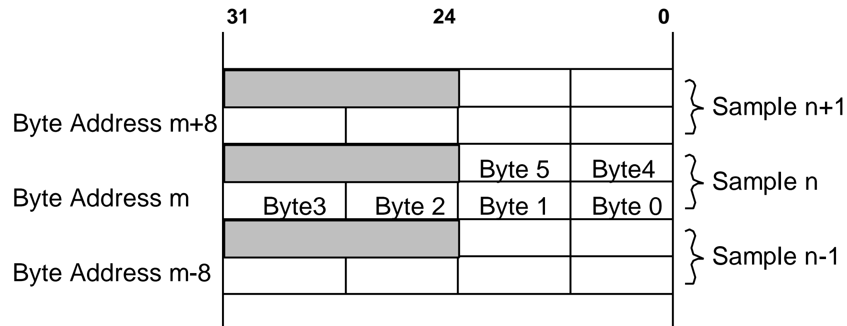
MODE[5..4]	period
00	20nS
01	40nS
10	80nS
11	160nS

Sample Buffer Memory Map

The Sample buffer is mapped simply into EISA memory space according to the value of the ADDRMAP register and the SRAMLD bit in the MODE register. As a precaution, the memory map of the board is disabled at power up and after a board reset (after a power- UP or a write to RESET). The SRAMLD bit determines whether the VRAM buffer or the trace SRAM is mapped into EISA memory space. The two cannot be accessed concurrently. The diagram below illustrates the memory map.

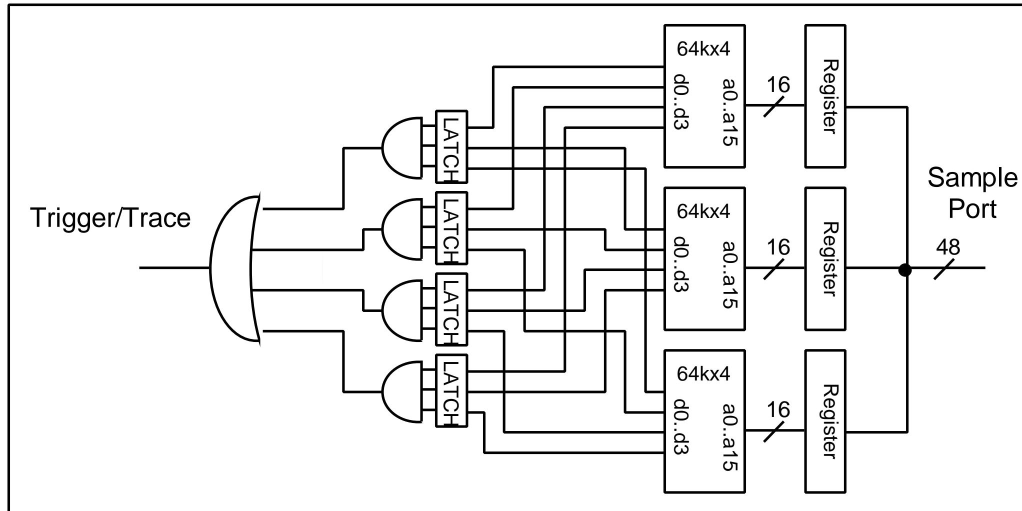


Each 48 bit sample is aligned on a 64 bit boundary in memory as shown below:



Trigger/Trace Setup

A simplified schematic of the trigger and trace system is shown below.

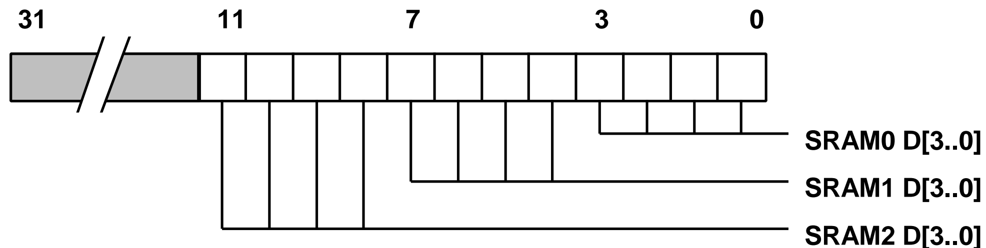


Essentially, the 48 bit sample data path is broken into three contiguous groups of 16 bits which are connected to the address lines of three 64kx4 SRAMs. SRAM0 a[15..0] is connected to sample data sd[15..0], SRAM1 a[15..0] is connected to sd[31..16] and SRAM2 a[15..0] to sd[47..32]. Each SRAM data line is ANDed with the corresponding data lines from the other two SRAMs and these products are ORed together to give the trigger or trace result.

A triplet of data bits from each SRAM (e.g. d0 on SRAM0 & SRAM1 & SRAM2) is called a *trace channel* and is individually selectable to be a *trigger* (if trigger value is matched, stop sampling when STOPCOUNT register reaches zero) or a *trace* (store sample only if trace value is matched).

Channels are selected as trace or trigger by writing the TRIGCONFIG register. Bits [3..0] of this register individually select the four trace channels as trigger (if set to one) or trace (if reset to zero).

To read or write the SRAM, first the SRAMLD bit of the MODE register must be set, and the RECORD bit must be reset. This disables sampling and maps the SRAM into the lower 128k of the EISA memory address space. Usually PAUSE would also be set. All three SRAMs are read/written together. SRAM0 data is buffered to EISA D[3..0], SRAM1 to EISA D[7..4] and SRAM2 to EISA D[11..9].



By programming the SRAMs with the right values it is possible to trace and/or trigger on multiple different values and even arbitrary ranges of values.

Trace Example 1: Trace/Trigger on a Single Value (D = x)

Suppose the 48 bit value to trace/trigger on is `0x123456789ABC`

Choose a trigger/trace channel, say channel #1 (bit zero in each SRAM) :

Write a logic 1 into the SRAMLD and PAUSE bits in the MODE register, and a logic 0 into the RECORD bit :

```
(I/O 0xZ008) := (I/O 0xZ008) AND 0xFFFFFFFB
(I/O 0xZ008) := (I/O 0xZ008)  OR 0x00000003
```

Set up the configuration bit for channel #1

```
(I/O 0xZ00C) := (I/O 0xZ00c) AND 0xFFFFFFFE      (Trace)

(I/O 0xZ00C) := (I/O 0xZ00c)  OR 0x00000001      (Trigger)
```

Break the value into three blocks of 16 bits and shift left by 2 to calculate the SRAM address :

```
0x1234 << 2 = 0x048D0
0x5678 << 2 = 0x259E0
0x9ABC << 2 = 0x26AF0
```

Write the trigger bitmap into SRAM :

```
(Mem 0xMMX048D0) := 0xXXXXX100
(Mem 0xMMX259E0) := 0xXXXXX010
(Mem 0xMMX26AF0) := 0xXXXXX001
```

Trace Example 2: Trace/Trigger on Multiple Values (D = x | y)

There are two options to load multiple values into the SRAM. It is possible to use a different channel per value but this limits the number of values to four. However if the values happen to occur in the same 64kSample block. (e.g. the two values `0x12345678DEAD` and `0x12345678BEEF` occur in the same 64kSample block) they can both be put into the same channel.

Trace Example 3: WildCarding or "Don't Care" bits.

Sometimes a trace or trigger channel requires that certain bits be wild cards or "don't care". In these cases it is necessary to program the SRAM such that the data read back is high regardless of the state of the wildcard bit. In effect all this means is that a one must be written into the SRAM at the addresses corresponding to the sample data being high and also at the addresses where the sample data is low. Suppose that in the first example bit five must be wildcarded. The trace value is thus:

`0x123456789A$C` where $ == binary `10X1`

The procedure is the same as before except two addresses are written into SRAM0 the addresses `0xMMX26AF0` & `0xMMX26A70`.

When two wildcards occur (in the same 64kSample block), there must be four addresses written, and with three wildcard bits, eight addresses and so on. To wildcard sixteen bits, all 64k addresses must be filled with ones in the correct channel.

All updates to the trace SRAM should be ORed in to avoid upsetting the data in different channels.

Trace Example 3: Trace/Trigger on a Range (x = D = y)

There are two cases:

Case1: Range occurs entirely within one 64kSample block

Trigger on range `0x12345678BEEF` = Sample = `0x12345678DEAD`

Calculate bitmap as before and write bits into SRAM2 & SRAM1. SRAM0 must have a range of addresses written. The Start and finishing addresses are:

`0xBEEF` << 2 = `0x2FBBC`

`0xDEAD` << 2 = `0x37AB4`

Note that writing `0x001` into these addresses may overwrite the information in SRAM1 & SRAM2 so the value must be ORed in. The following pseudocode illustrates this:

```
FOR i := 0xbeef to 0xdead DO
      BEGIN
            temp = Mem(0xMMX00000) + (i<<2));
            temp = temp OR 0x001;
            Mem(0xMMX00000) + (i<<2)) = temp;
      END
```

As with multiple single values, multiple ranges can be placed in the same channel provided they all fall within the same block. Ranges that consist of exact multiples of 64kSamples can also fit in one channel. In this case the lower sixteen bits of the sample data are wildcarded.

Case 2: Range Overlaps a 64kSample boundary:

If the range overlaps a 64kSample boundary the range must be broken into two ranges and each programmed into a different channel. e.g. the two values `0x12346578ABCD` and `0x12345679ABCD` specify a range that overlaps one boundary. In this example the range can be broken into two ranges, viz:

`0x12345678ABCD - 0x12345678FFFF`

`0x123456790000 - 0x12345679ABCD`

These ranges will each fit into separate channels as in Case 1 above.

Setup Sequences

The interactions between RECORD, SRAMLD and OUTPUTENA[5..0] mean that the various registers must be written in an order that precludes these interactions.from undoing the effect of prior writes. As an example, let us consider the sequence of actions needed for starting a trace.

1. Firstly the PAUSE and SRAMLD of the MODE register should be set to a logic 1 and the RECORD bit reset to a logic 0. PAUSE=1 stops tracing. RECORD=0 disables both the sample input buffers and the timestamp buffers, which is a necessary condition for accessing the SRAM with SRAMLD=1.

```
(I/O 0xZ008) := (I/O 0xZ008) AND 0xFFFFFFFB
(I/O 0xZ008) := (I/O 0xZ008)  OR 0x00000003
```

2. Next the trigger and trace patterns can be preloaded into the SRAM as discussed above.

3. Then the trigger/trace configuration can be written to TRIGCONFIG[5..0].

   `(I/O 0xZ00C) := 0x0000000X` (where `X` is the desired value)

4. After this, the STOPCOUNT can be written.

   `(I/O 0xZ014) := 0x00XXX000` (where `XXX` is the desired value)

5. Then the HEADPTR can be written.

   `(I/O 0xZ018) := 0x00XXX000` (where `XXX` is the desired value)

6. At this point the RECORD bit needs to be set, before OUTPUTENA is written to, so that when the latter is done, it won't be undone by a subsequent toggle of the RECORD bit.

```
(I/O 0xZ008) := (I/O 0xZ008)  OR 0x00000004
```

7. Now OUTPUTENA[5..0] can be written, to enable the sample input buffers or timestamp buffers as required.

   `(I/O 0xZ01C) := 0x0000000X` (where `X` is the desired value)

8. Finally the PAUSE and SRAMLD bits can be reset to allow tracing to begin.

```
(I/O 0xZ008) := (I/O 0xZ008) AND 0xFFFFFFFC
```

DMA and Transparent Operation

The EISA interface has been designed as a 33MByte/sec bursting slave that allows data to be read out transparently and asynchronously without affecting data acquisition. The WRAP bit of the MODE register causes the board to ingnore the STOPCOUNT counter and continue sampling when it reaches zero. If this bit is set, then the VRAM buffer wraps around and can be continuously read out via the EISA interface without stopping the sample clock. If it is possible to atomically update the STOPCOUNT register, then very long traces may be generated, provided the data can be read out of the EISA interface, and stored, as fast as it is acquired.

The DONE bit of the STATUS register allows a very simple loop to be constructed :

```
... set up registers, etc.
/* now start tracing to bottom half of VRAM */
(I/O 0xZ008) := (I/O 0xZ008) AND 0xFFFFFFFD       (clear PAUSE)
(I/O 0xZ018) := 0x00000000            (HEADPTR <- bottom half of VRAM)
(I/O 0xZ014) := 0x00800000            (STOPCOUNT <- half VRAM size)
while(1)
{
  do
  {
     sleep(1)
  }while !(DONE)
  /* then start tracing to top half of VRAM */
  (I/O 0xZ018) := 0x00800000          (HEADPTR <- top half of VRAM)
  (I/O 0xZ014) := 0x00800000          (STOPCOUNT <- half VRAM size)
  /* and concurrently read and store bottom half of VRAM */
  read_and_store(0x00000000)
  do
  {
     sleep(1)
  }while !(DONE)
  /* go back to tracing to bottom half of VRAM */
  (I/O 0xZ018) := 0x00000000          (HEADPTR <- bottom half of VRAM)
  (I/O 0xZ014) := 0x00800000          (STOPCOUNT <- half VRAM size)
  /* and concurrently read and store top half of VRAM */
  read_and_store(0x00800000)
}
```

Generally the reading and storing would be done via DMA. The DMAENA bit of the MODE register enables the board to request the host machine to perform DMA on EISA DMA channel 7 when the STOPCOUNT counter reaches zero - the DMA handler would normally do the starting of the next acquisition at the same time. Completion of the DMA transfer can be signalled via an interrupt. The IRQENA bit of the MODE register causes the board to assert EISA interrupt IRQ12. The interrupt is level triggered so that multiple boards can share the same interrupt.

Currently, however, neither DMAENA nor IRQENA are implemented. Neither are burst transfers enabled.

Alternatively, the functionality can be provided within a program loop. The Intel 80x86 string copy instruction is interruptable, allows very long strings to be copied from source to destination, and issues reads and writes as fast as they can be issued. It does, however, consume processor cycles. This will seriously affect a single-issue procesor, but is acceptable for multi-issue processors such as the PentiumPro or Pentium II. Unfortunately the EISA bus is not found on systems that accomodate the latter two processors.

Note that the TRACE CONTINOUSLY bit of the TRIGCONFIG register is not related to this usage of the board. This bit, when set, forces acceptance of every sample. Since one of the provisos for the above is that the data can be read out of the EISA interface, and stored, as fast as it is acquired, use of the TRACE CONTINOUSLY bit is only likely to exacerbate the difficulty of satisfying this proviso.

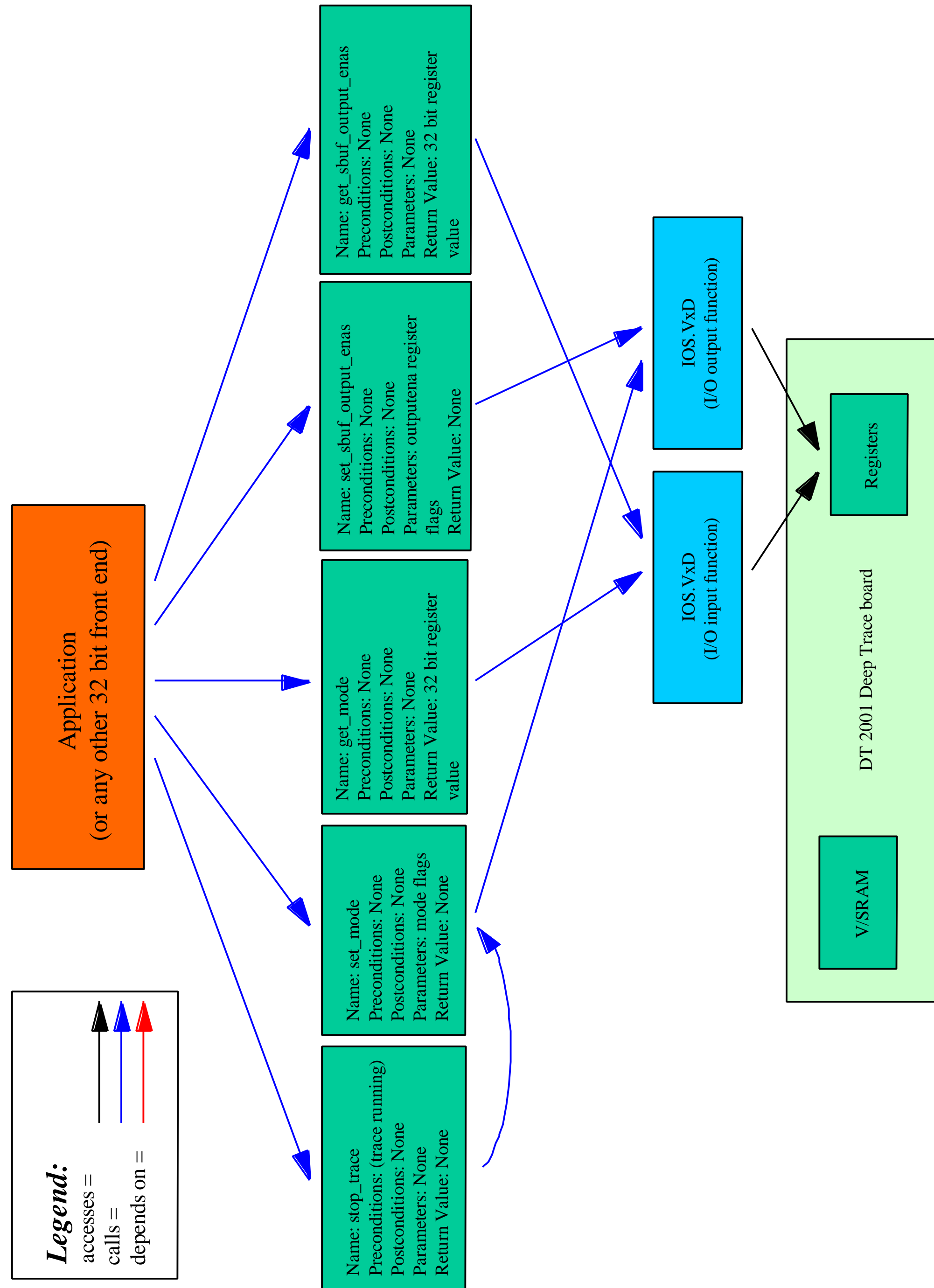
Applications Programming Interface (API)

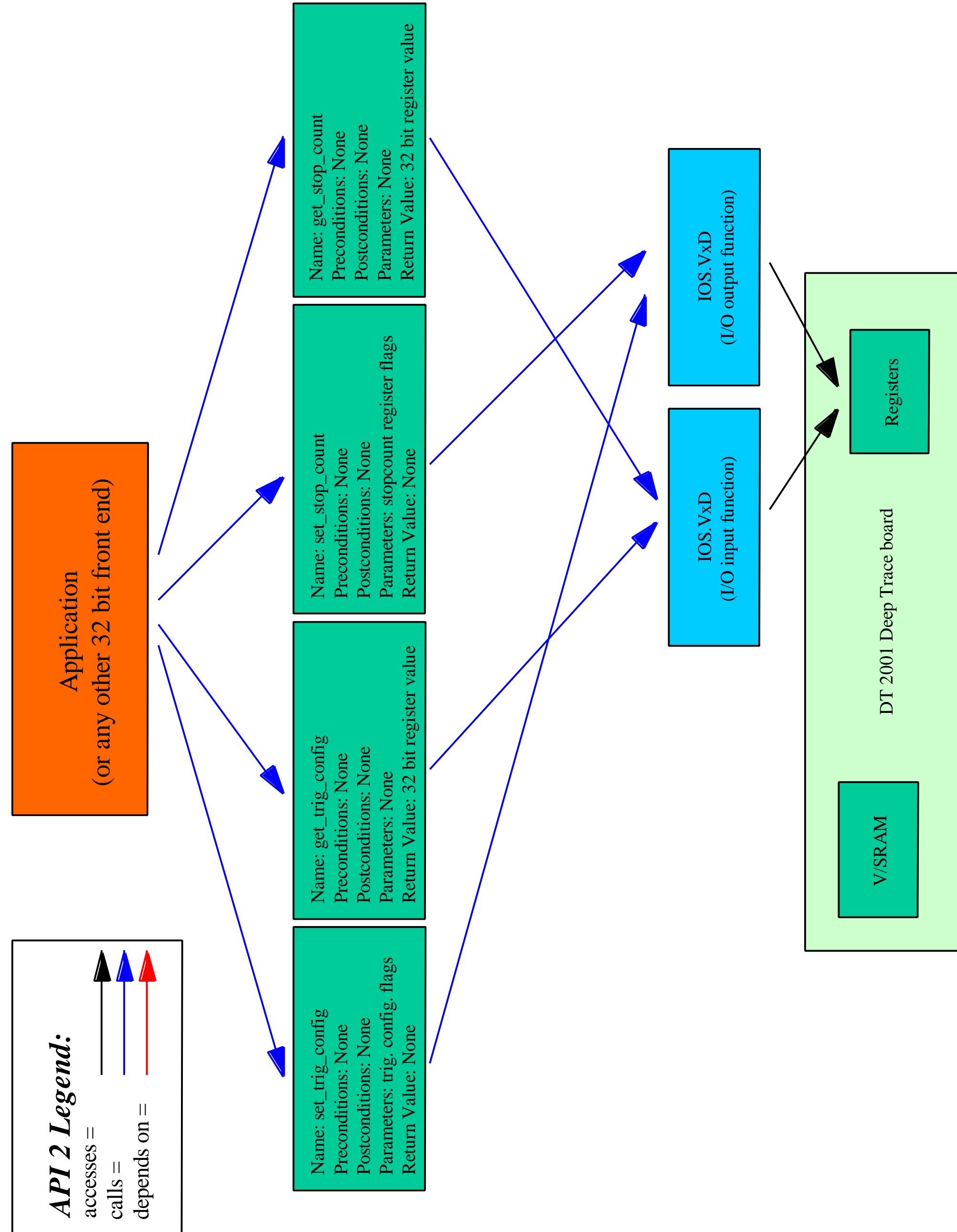
In a Microsoft Windows 95 or NT environment, device access may be implemented via functions that are loaded when needed from a Dynamic Link Library (DLL), i.e. they are dynamically linked to the application during execution. The DLL then calls device drivers to perform the I/O. A generic DLL can be defined that will serve as an API for both Windows 95 and NT environments; an example is given in the next section. Unfortunately a generic device driver cannot be defined for both Windows 95 and NT, since their requirements are different. Hence these are separately discussed further below.

The appropriate API functions for the DT200.1 are :

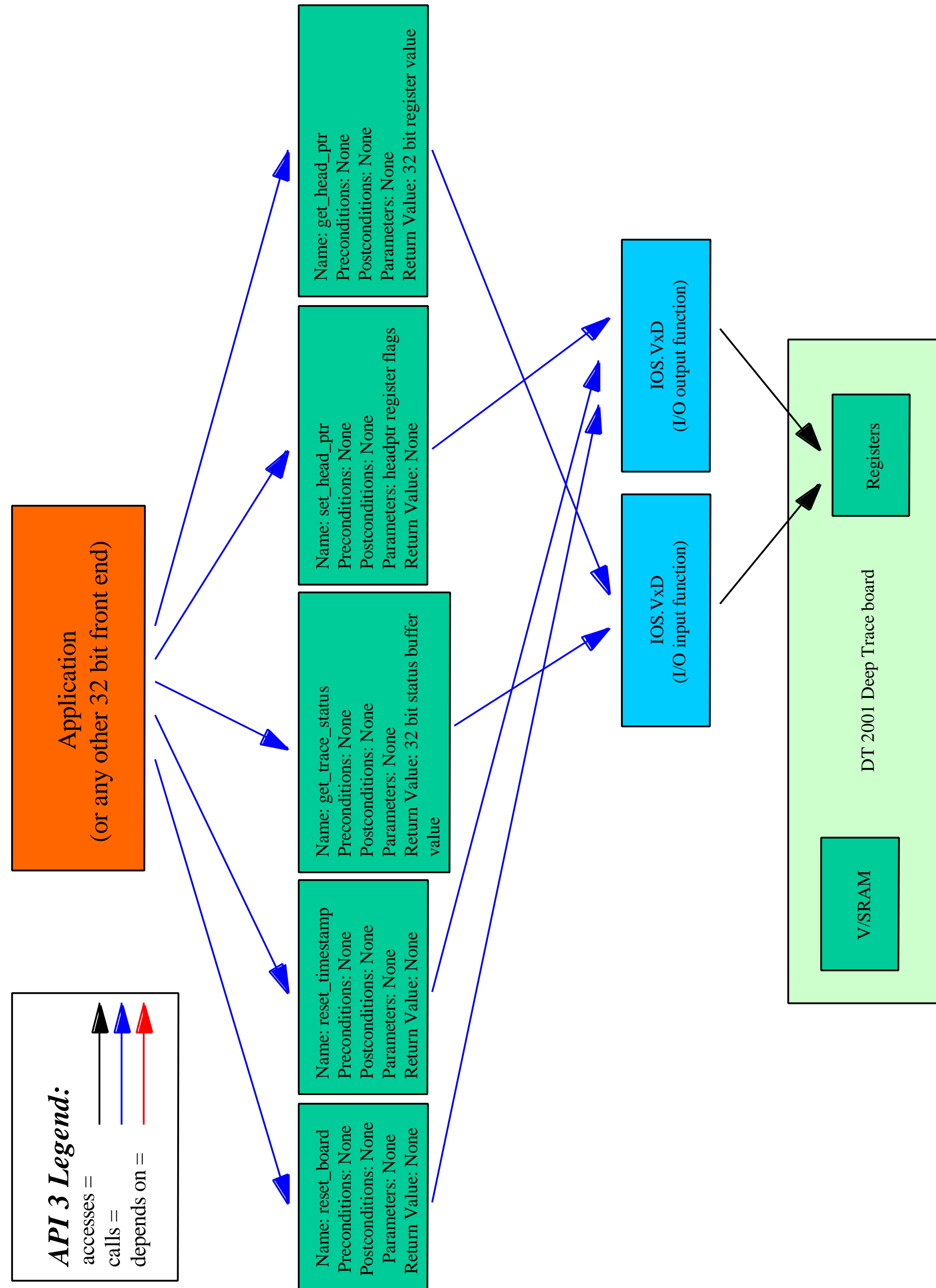
```
void reset_board(void);                    //Reset board
void set_map(unsigned long address);       //Writes to ADDRMAP
                                           //  register
unsigned long get_map(void);               //Read the current
                                           //  address mapping
void set_mode(unsigned long flags);        //Writes to the MODE
                                           //  register
unsigned long get_mode(void);              //Read the current mode
void set_trig_config(unsigned long flags); //Write to the trigger
                                           // config register
unsigned long get_trig_config(void);       //Read from the trigger
                                           //  config register
void reset_timestamp(void);                //Reset the timestamp
                                           //  counter
void set_stop_count(unsigned long flags);  //Write to the
                                           //  STOPCOUNT register
unsigned long get_stop_count(void);        //Read from the
                                           //  STOPCOUNT register
void set_head_ptr(unsigned long flags);    //Write to the
                                           //  HEADPTR register
unsigned long get_head_ptr(void);          //Read from the
                                           //  HEADPTR register
void set_sbuf_output_enas(unsigned long flags); //Write to the
                                           //  OUTPUTENA register
unsigned long get_sbuf_output_enas(void);  //Read from the
                                           //  OUTPUTENA register
unsigned long get_trace_status(void);      //Read from the
                                           //  STATUS buffer
unsigned long get_EISA_ID(void);           //Read from the EISA
                                           //  ID register
void get_EISAID(char *text_ID);            //Read from EISA ID
                                           //  register amd decode
void stop_trace(void);                     //Stop trace
unsigned long *map_RAM(void);              //Map VRAM and SRAM
                                           //  into linear memory,
                                           //  return address 0x0
                                           //  on error
void unmap_RAM(void);                      //Unmap VRAM and SRAM

// read out VRAM or SRAM (start address specified)
//   and store it in array:
BOOL store_RAM(unsigned long *array,
               unsigned long start_address_offset,
               unsigned long size_in_bytes);
```

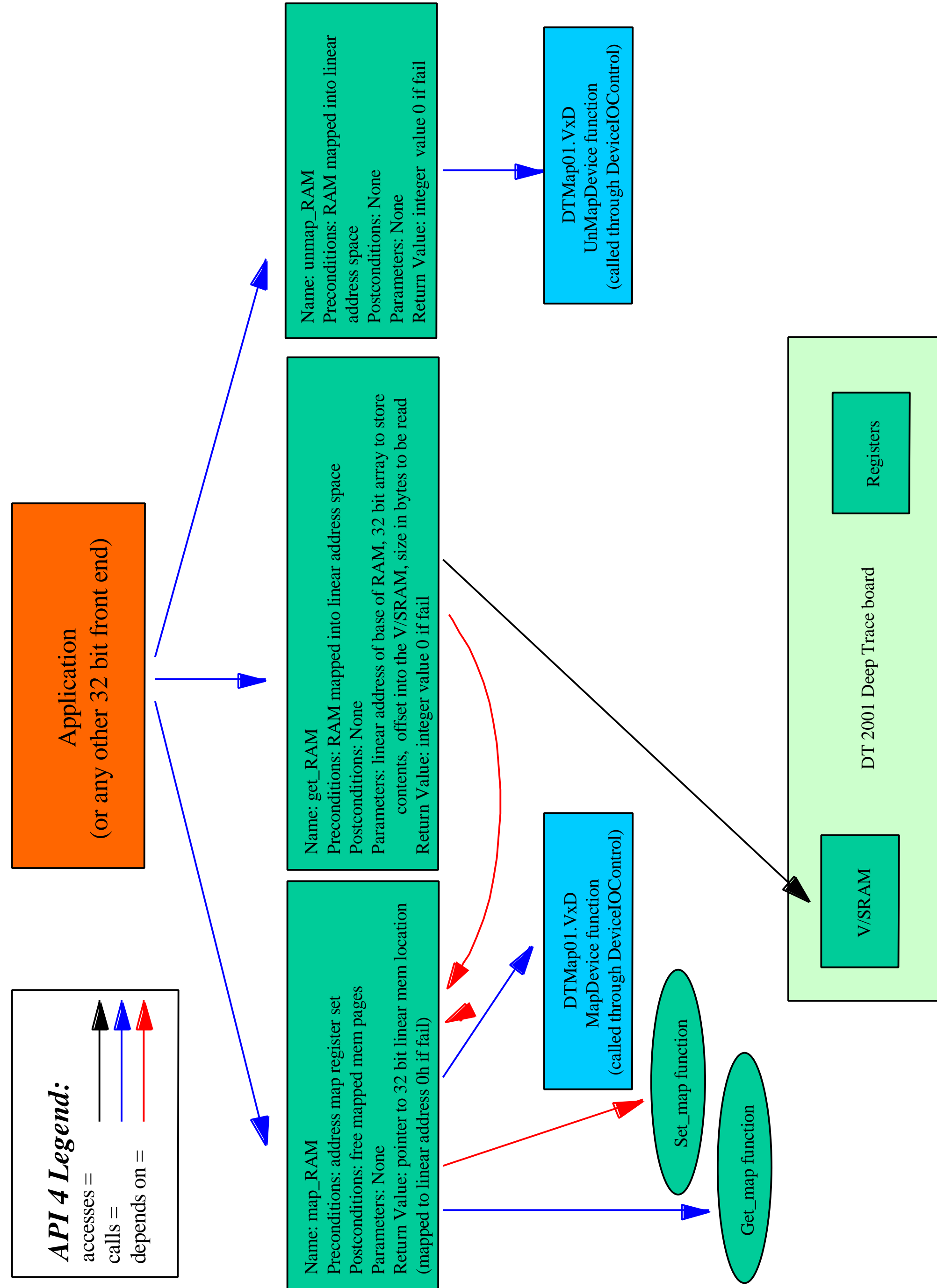
These API functions provide an applications interface suitable for programs written in high level languages like C or Java, such as the example Java application given further again below. The API is illustrated in the following four diagrams :

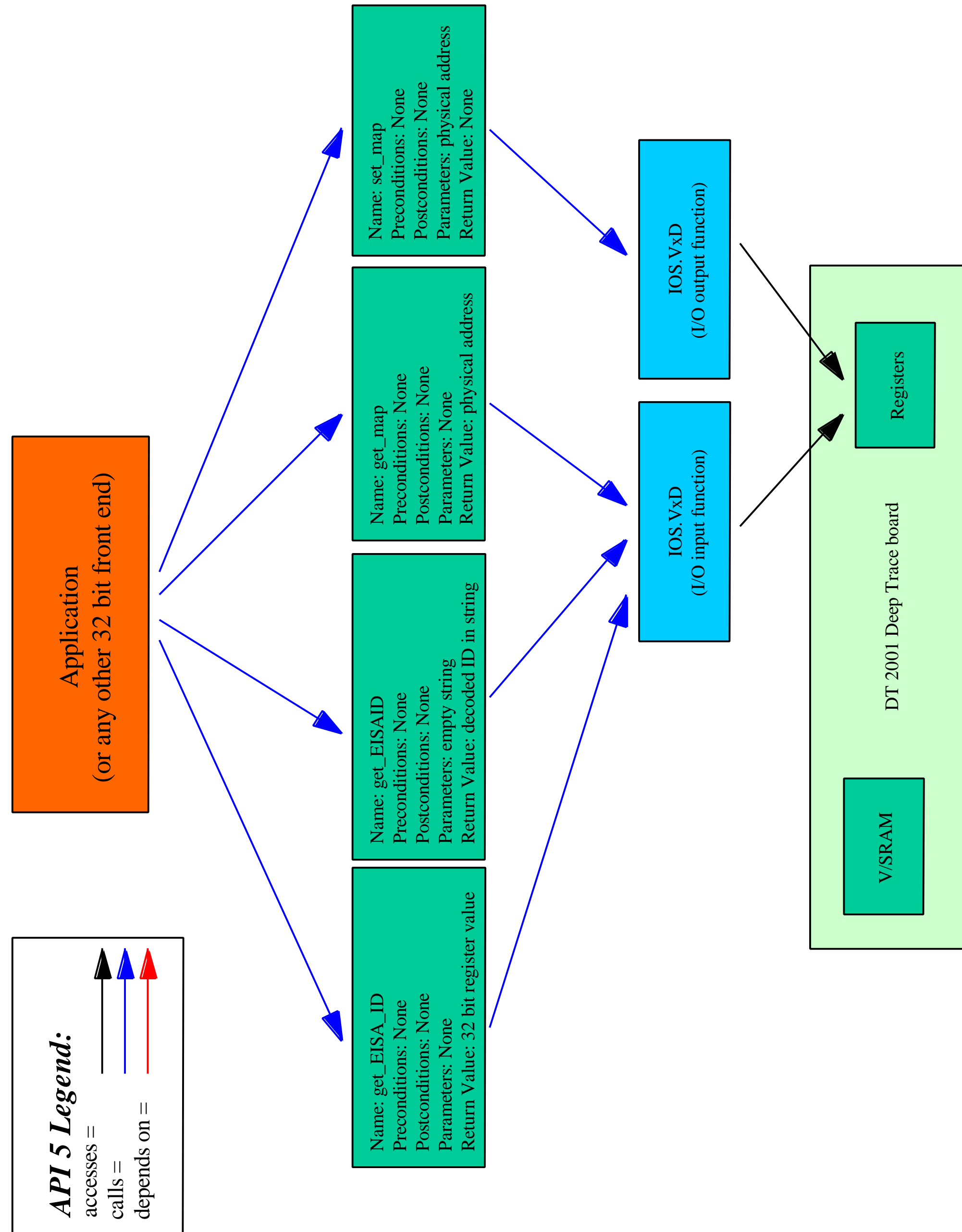




API 2 Legend:
accesses =
calls =
depends on =
Application
(or any other 32 bit front end)
Name: set_trig_config
Preconditions: None
Postconditions: None
Parameters: trig. config. flags
Return Value: None
Name: get_trig_config
Preconditions: None
Postconditions: None
Parameters: None
Return Value: 32 bit register value
Name: set_stop_count
Preconditions: None
Postconditions: None
Parameters: stopcount register flags
Return Value: None
Name: get_stop_count
Preconditions: None
Postconditions: None
Parameters: None
Return Value: 32 bit register value
IOS.VxD
(I/O output function)
IOS.VxD
(I/O input function)
Registers
DT 2001 Deep Trace board
V/SRAM


API 3 Legend:
accesses =
calls =
depends on =
Application
(or any other 32 bit front end)
Name: get_head_ptr
Preconditions: None
Postconditions: None
Parameters: None
Return Value: 32 bit register value
Name: set_head_ptr
Preconditions: None
Postconditions: None
Parameters: headptr register flags
Return Value: None
Name: get_trace_status
Preconditions: None
Postconditions: None
Parameters: None
Return Value: 32 bit status buffer value
Name: reset_timestamp
Preconditions: None
Postconditions: None
Parameters: None
Return Value: None
Name: reset_board
Preconditions: None
Postconditions: None
Parameters: None
Return Value: None
IOS.VxD
(I/O output function)
IOS.VxD
(I/O input function)
Registers
DT 2001 Deep Trace board
V/SRAM


API 4 Legend:
accesses =
calls =
depends on =
Application
(or any other 32 bit front end)
Name: unmap_RAM
Preconditions: RAM mapped into linear address space
Postconditions: None
Parameters: None
Return Value: integer value 0 if fail
DTMap01.VxD
UnMapDevice function
(called through DeviceIOControl)
Name: get_RAM
Preconditions: RAM mapped into linear address space
Postconditions: None
Parameters: linear address of base of RAM, 32 bit array to store contents, offset into the V/SRAM, size in bytes to be read
Return Value: integer value 0 if fail
Name: map_RAM
Preconditions: address map register set
Postconditions: free mapped mem pages
Parameters: None
Return Value: pointer to 32 bit linear mem location (mapped to linear address 0h if fail)
DTMap01.VxD
MapDevice function
(called through DeviceIOControl)
Set_map function
Get_map function
DT 2001 Deep Trace board
Registers
V/SRAM


Application
(or any other 32 bit front end)
Name: set_map
Preconditions: None
Postconditions: None
Parameters: physical address
Return Value: None
Name: get_map
Preconditions: None
Postconditions: None
Parameters: None
Return Value: physical address
Name: get_EISAID
Preconditions: None
Postconditions: None
Parameters: empty string
Return Value: decoded ID in string
Name: get_EISA_ID
Preconditions: None
Postconditions: None
Parameters: None
Return Value: 32 bit register value
IOS.VxD
(I/O output function)
IOS.VxD
(I/O input function)
Registers
DT 2001 Deep Trace board
V/SRAM
API 5 Legend:
accesses =
calls =
depends on =

Generic Dynamic Link Library (DLL)

A generic DLL **definition file** `MapDev.h` for the DT200.1 is as follows :

```
// MAPDEV.h - include file for VxD MAPDEV
// Copyright (c) 1996 Vireo Software, Inc.

#ifndef NotVxD

#include <vtoolsc.h>

#define MAPDEV_Major          1
#define MAPDEV_Minor          0
#define MAPDEV_DeviceID       UNDEFINED_DEVICE_ID
#define MAPDEV_Init_Order     UNDEFINED_INIT_ORDER

#define LPVOID PVOID

#endif

// This is the request structure that applications use
// to request services from the MAPDEV VxD.

typedef struct _MapDevRequest
{
   DWORD  mdr_ServiceID;         // supplied by caller
   LPVOID mdr_PhysicalAddress;   // supplied by caller
   DWORD  mdr_SizeInBytes;       // supplied by caller
   LPVOID mdr_LinearAddress;     // returned by VxD
   WORD   mdr_Selector;          // returned if 16-bit caller
   WORD   mdr_Status;            // MDR_xxxx code below
} MAPDEVREQUEST, *PMAPDEVREQUEST;

#define MDR_SERVICE_MAP       CTL_CODE(FILE_DEVICE_UNKNOWN, 1,
                              METHOD_NEITHER, FILE_ANY_ACCESS)
#define MDR_SERVICE_UNMAP     CTL_CODE(FILE_DEVICE_UNKNOWN, 2,
                              METHOD_NEITHER, FILE_ANY_ACCESS)

#define MDR_STATUS_SUCCESS    1
#define MDR_STATUS_ERROR      0
```

The associated **C++ source file** `MapDev.cpp` for the DLL is :

```
#include <windows.h>
#include <winioctl.h>
#include <conio.h>
#include <memory.h>
#define NotVxD
#include "DT2001.h"
#include "DTMap01.h"

/*
// DLL Entry Point disabled since not used
//   Note: The DLL entry point functoin is disables at the
//         moment because Visual Basic doesn't seem to be
//         able to call it; (no loadlib function provided).
```

```
BOOL WINAPI DllEntryPoint (HINSTANCE hDLL,
                           DWORD dwReason,
                           LPVOID Reserved)
{
   BOOL Success = TRUE;

   switch (dwReason)
   {
      case DLL_PROCESS_ATTACH:
      {
         //reset_board();
         //set_map(PHY_ADDR);//set physical address of VRAM/SRAM
         //map_RAM();//set linear address of VRAM/SRAM
         break;
      }
      case DLL_PROCESS_DETACH:
      {
         //unmap_RAM();
         //reset_board();
         break;
      }
      case DLL_THREAD_ATTACH:  break;
      case DLL_THREAD_DETACH:  break;
   }
   return Success;
}
*/

//Global Declarations:
HANDLE hDevice;          //handle for VxD

//resets board
__declspec( dllexport ) void __stdcall
                        reset_board(void)
{
   _outpd(BASE_ADDR | REG_RESET, ANYVALUE);
}

//Writes to ADDRMAP register
//  input: is physical address the VRAM/SRAM is to appear at
__declspec( dllexport ) void __stdcall
                        set_map(unsigned long address)
{
   _outpd(BASE_ADDR | REG_ADDRMAP, address);
}

//Read the current physical address mapping
__declspec( dllexport ) unsigned long __stdcall
                        get_map(void)
{
   return (_inpd(BASE_ADDR | REG_ADDRMAP) & 0xFF000000);
}

//Writes to the MODE register
__declspec( dllexport ) void __stdcall
                        set_mode(unsigned long flags)
{
   _outp(BASE_ADDR | REG_MODE, (int)flags);
}

//Read the current mode
```

```
__declspec( dllexport ) unsigned long __stdcall
                        get_mode(void)
{
   return (((unsigned long) _inp(BASE_ADDR | REG_MODE)) &
                                 0x000000 0FF);
}

//Write to the trigger config register
__declspec( dllexport ) void __stdcall
                        set_trig_config(unsigned long flags)
{
   _outp(BASE_ADDR | REG_TRIGCONFIG, (int)flags);
}

//Read from the trigger config register
__declspec( dllexport ) unsigned long __stdcall
                        get_trig_config(void)
{
   return (((unsigned long) _inp(BASE_ADDR | REG_TRIGCONFIG)) &
                                 0x0000003F);
}

//Reset the timestamp counter
__declspec( dllexport ) void __stdcall
                        reset_timestamp(void)
{
   _outpd(BASE_ADDR | REG_TIMERESET, ANYVALUE);
}

//Write to the STOPCOUNT register
__declspec( dllexport ) void __stdcall
                        set_stop_count(unsigned long flags)
{
   _outpd(BASE_ADDR | REG_STOPCOUNT, flags);
}

//Read from the STOPCOUNT register
__declspec( dllexport ) unsigned long __stdcall
                        get_stop_count(void)
{
   return (_inpd(BASE_ADDR | REG_STOPCOUNT) & 0x00FF8000);
}

//Write to the HEADPTR register
__declspec( dllexport ) void __stdcall
                        set_head_ptr(long int head_ptr)
{
   _outpd(BASE_ADDR | REG_HEADPTR, head_ptr);
}

//Read from the HEADPTR register
__declspec( dllexport ) unsigned long __stdcall
                        get_head_ptr(void)
{
   return (_inpd(BASE_ADDR | REG_HEADPTR) & 0x00FF8000);
}

//Write to the OUTPUTENA register
__declspec( dllexport ) void __stdcall
                        set_sbuf_output_enas(unsigned long flags)
{
```

```
    _outp(BASE_ADDR | REG_OUTPUTENA, (int)flags);
}

//Read from the OUTPUTENA register
__declspec( dllexport ) unsigned long __stdcall
                        get_sbuf_output_enas(void)
{
    return (((unsigned long)_inp(BASE_ADDR | REG_OUTPUTENA)) &
                                0x0000003F);
}

//Read from the STATUS buffer
__declspec( dllexport ) unsigned long __stdcall
                        get_trace_status(void)
{
    return (_inpd(BASE_ADDR | REG_STATUS) & 0x00008000);
}

//Read from the EISA ID register
__declspec( dllexport ) unsigned long __stdcall
                        get_EISA_ID(void)
{
    return _inpd(BASE_ADDR | REG_EISAID);
}

//Read from EISA ID register amd decode string
//  input: pointer to a char array (min 7 chars long)
__declspec( dllexport ) unsigned long __stdcall
                        get_EISAID(char *text_ID)
{
    unsigned int chars;
    unsigned int digits;
    unsigned long reg = _inpd(BASE_ADDR | REG_EISAID);
    chars = reg;
    digits = reg >> 16;

    text_ID[0] = (char) ( ( chars & 0x007C) >>  2) + 0x40;
    text_ID[1] = (char) ((( chars & 0x0003) <<  3) |
                         (( chars & 0xE000) >> 13))+ 0x40;
    text_ID[2] = (char) ( ( chars & 0x1F00) >>  8) + 0x40;
    text_ID[3] = (char) ( (digits & 0x00F0) >>  4) + '0';
    text_ID[4] = (char) ( (digits & 0x000F) >>  0) + '0';
    text_ID[5] = (char) (  (digits & 0xF000) >> 12) + '0';
    text_ID[6] = (char) ( (digits & 0x0F00) >>  8) + '0';
    text_ID[7] = 0;
    return 1;
}

//Stop trace
__declspec( dllexport ) void __stdcall
                        stop_trace(void)
{
    set_mode(PAUSE);
}

//Map VRAM and SRAM into linear memory, returns a pointer
// to base (32bit)
__declspec( dllexport ) unsigned long __stdcall
                        map_RAM(void)
{
    DWORD cbBytesReturned;//count of bytes returned from VxD
```

```
    MAPDEVREQUEST req;     //VxD request structure
    PVOID inBuf[1];        //buffer for DevIOCrtl pointer
                           // to req structure

    const PCHAR VxDName = "\\\\.\\DTMAP01.VXD";
    const PCHAR VxDNameAlreadyLoaded = "\\\\.\\DTMAP01";


    hDevice = CreateFile(VxDName, 0,0,0,
                          CREATE_NEW, FILE_FLAG_DELETE_ON_CLOSE, 0);
    if (hDevice == INVALID_HANDLE_VALUE)
       hDevice = CreateFile(VxDNameAlreadyLoaded, 0,0,0,
                          CREATE_NEW, FILE_FLAG_DELETE_ON_CLOSE, 0);
    if (hDevice == INVALID_HANDLE_VALUE)
       return (unsigned long) GetLastError();//error!
    else
    {
       //set up request structure:
       req.mdr_ServiceID = MDR_SERVICE_MAP;
       req.mdr_PhysicalAddress = (PVOID) get_map();
       req.mdr_SizeInBytes = VRAM_SIZE;
       inBuf[0] = &req;

       //call Win32 API Message DeviceIOControl:
       if(! DeviceIoControl(hDevice, MDR_SERVICE_MAP, inBuf,
                             sizeof(PVOID), NULL, 0,
                             &cbBytesReturned, NULL))
          return 0;
       else
          return (unsigned long) req.mdr_LinearAddress;//Success!
    }
}

//Unmap VRAM and SRAM
//uses the VxD, frees the linear address space
__declspec ( dllexport ) int __stdcall
                         unmap_RAM(void)
{
    DWORD cbBytesReturned; //count of bytes returned from VxD
    MAPDEVREQUEST req;      //VxD request structure
    PVOID inBuf[1];
    const PCHAR VxDName = "\\\\.\\DTMAP01.VXD";
    const PCHAR VxDNameAlreadyLoaded = "\\\\.\\DTMAP01";
    if (hDevice == INVALID_HANDLE_VALUE)
       hDevice = CreateFile(VxDNameAlreadyLoaded, 0,0,0,
                         CREATE_NEW, FILE_FLAG_DELETE_ON_CLOSE, 0);
    if (hDevice == INVALID_HANDLE_VALUE)
    {
       return 0;//error! can't get a handle on VxD
    }
    else
    {
       //set up request structure:
       req.mdr_ServiceID = MDR_SERVICE_UNMAP;
       req.mdr_PhysicalAddress = (void *) get_map();
       req.mdr_SizeInBytes = VRAM_SIZE;
       inBuf[0]=&req;
       //call Win32 API Message DeviceIOControl:
       if (! DeviceIoControl(hDevice, MDR_SERVICE_UNMAP, inBuf,
                              sizeof(PVOID), NULL, 0,
                              &cbBytesReturned, NULL))
```

```
      {
         return 0;//error! call to VxD failed
      }
      else
      {
         return 1; //success!
      }
   }
}

//read out SRAM or VRAM and store in array
//input: array of 32bit storage locations, offset address
//       into the VRAM/SRAM and the #bytes to read out
//output: TURE if success, FALSE if fail
__declspec( dllexport ) int __stdcall
                        get_RAM(PBYTE Map_Address,
                                unsigned long *array,
                              unsigned long start_address_offset,
                                unsigned long size_in_bytes)
{
   int success = TRUE;
   //boudary checking:
   if (get_mode() & SRAMLD)
   {
      //access SRAM
      if ((start_address_offset + size_in_bytes) > SRAM_SIZE)
         success = FALSE;
   }
   else
   {
      //access VRAM
      if ((start_address_offset + size_in_bytes) > VRAM_SIZE)
         success = FALSE;
      else
         memcpy(array,Map_Address + start_address_offset,
                size_in_bytes);//read out RAM and store in array
   }
   return success;
}
```

An appropriate **compiler description file** `MapDev.dsp` would be :

```
# Microsoft Developer Studio Project File - Name="DT2001"
  - Package Owner=<4>
# Microsoft Developer Studio Generated Build File, Format
  Version 5.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Dynamic-Link Library" 0x0102

CFG=DT2001 - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project
!MESSAGE using NMAKE, use the Export Makefile command and run :
!MESSAGE
!MESSAGE NMAKE /f "DT2001.MAK".
!MESSAGE
!MESSAGE You can specify a configuration when running NMAKE
!MESSAGE by defining the macro CFG on the command line.
```

```
!MESSAGE For example:
!MESSAGE
!MESSAGE NMAKE /f "DT2001.MAK" CFG="DT2001 - Win32 Debug"
!MESSAGE
!MESSAGE Possible choices for configuration are:
!MESSAGE
!MESSAGE "DT2001 - Win32 Release"
!MESSAGE           (based on "Win32 (x86) Dynamic-Link Library")
!MESSAGE "DT2001 - Win32 Debug"
!MESSAGE           (based on "Win32 (x86) Dynamic-Link Library")
!MESSAGE

# Begin Project
# PROP Scc_ProjName ""
# PROP Scc_LocalPath ""
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe

!IF  "$(CFG)" == "DT2001 - Win32 Release"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 0
# PROP BASE Output_Dir "Release"
# PROP BASE Intermediate_Dir "Release"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 0
# PROP Output_Dir "Release"
# PROP Intermediate_Dir "Release"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D
  "_WINDOWS" /YX /FD /c
# ADD CPP /nologo /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D
  "_WINDOWS" /YX /FD /c
# ADD BASE MTL /nologo /D "NDEBUG" /mktyplib203 /o NUL /win32
# ADD MTL /nologo /D "NDEBUG" /mktyplib203 /o NUL /win32
# ADD BASE RSC /l 0x1809 /d "NDEBUG"
# ADD RSC /l 0x1809 /d "NDEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib
  comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib
  uuid.lib odbc32.lib odbccp32.lib /nologo /subsystem:windows
  /dll /machine:I386
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib
  comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib
  uuid.lib odbc32.lib odbccp32.lib /nologo /subsystem:windows
  /dll /machine:I386

!ELSEIF  "$(CFG)" == "DT2001 - Win32 Debug"

# PROP BASE Use_MFC 0
# PROP BASE Use_Debug_Libraries 1
# PROP BASE Output_Dir "Debug"
# PROP BASE Intermediate_Dir "Debug"
# PROP BASE Target_Dir ""
# PROP Use_MFC 0
# PROP Use_Debug_Libraries 1
```

```
# PROP Output_Dir "Debug"
# PROP Intermediate_Dir "Debug"
# PROP Target_Dir ""
# ADD BASE CPP /nologo /MTd /W3 /Gm /GX /Zi /Od /D "WIN32" /D
  "_DEBUG" /D "_WINDOWS" /YX /FD /c
# ADD CPP /nologo /MTd /W3 /Gm /GX /Zi /Od /D "WIN32" /D
  "_DEBUG" /D "_WINDOWS" /YX /FD /c
# ADD BASE MTL /nologo /D "_DEBUG" /mktyplib203 /o NUL /win32
# ADD MTL /nologo /D "_DEBUG" /mktyplib203 /o NUL /win32
# ADD BASE RSC /l 0x1809 /d "_DEBUG"
# ADD RSC /l 0x1809 /d "_DEBUG"
BSC32=bscmake.exe
# ADD BASE BSC32 /nologo
# ADD BSC32 /nologo
LINK32=link.exe
# ADD BASE LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib
  comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib
  uuid.lib odbc32.lib odbccp32.lib /nologo /subsystem:windows
  /dll /debug /machine:I386 /pdbtype:sept
# ADD LINK32 kernel32.lib user32.lib gdi32.lib winspool.lib
  comdlg32.lib advapi32.lib shell32.lib ole32.lib oleaut32.lib
  uuid.lib odbc32.lib odbccp32.lib /nologo /subsystem:windows
  /dll /debug /machine:I386 /pdbtype:sept

!ENDIF

# Begin Target

# Name "DT2001 - Win32 Release"
# Name "DT2001 - Win32 Debug"
# Begin Source File

SOURCE=.\DT2001.cpp
# End Source File
# Begin Source File

SOURCE=.\dt2001.def
# End Source File
# Begin Source File

SOURCE=.\DT2001.h
# End Source File
# End Target
# End Project
```

Windows 95 Virtual Device Driver (VxD)

Since the board uses both I/O for the registers and normal memory space for the VRAM and SRAM, the most efficient implementation in a Microsoft Windows 95 is to use the `IOS.VxD` driver for the I/O space and write a custom driver for the memory mapping for the VRAM and SRAM. The memory map driver is called `DTMap01.VxD`. The `IOS.VxD` driver comes with Windows 95 and is located in the `SYSTEM` folder of a Windows 95 system. It is also sensible to use the *Win32 API* to program the driver since it allows easy porting of the driver to Windows NT, as well as being the fastest and most reliable API for Microsoft products. The Win32 API uses control messages to access VxDs, in particular `DeviceIOControl`.. The driver is derived from an example found in the *VToolsD* programming package (Vireo Software Inc.). This is called `MapDev.VxD` and is located in the `Examples` folder of the VToolsD installation.

A sample **definition file** `DTMap01.h` would be :

```
// DTMAP01.h - include file for VxD DTMap01

#ifndef NotVxD

#include <vtoolsc.h>

#define MAPDEV_Major          1
#define MAPDEV_Minor          0
#define MAPDEV_DeviceID       UNDEFINED_DEVICE_ID
#define MAPDEV_Init_Order     UNDEFINED_INIT_ORDER

#define LPVOID PVOID

#endif

// This is the request structure that applications use
// to request services from the MAPDEV VxD.

typedef struct _MapDevRequest
{
   DWORD  mdr_ServiceID;          // supplied by caller
   LPVOID mdr_PhysicalAddress;    // supplied by caller
   DWORD  mdr_SizeInBytes;        // supplied by caller
   LPVOID mdr_LinearAddress;      // returned by VxD
   WORD   mdr_Selector;           // returned if 16-bit caller
   WORD   mdr_Status;             // MDR_xxxx code below
} MAPDEVREQUEST, *PMAPDEVREQUEST;

#define MDR_SERVICE_MAP       CTL_CODE(FILE_DEVICE_UNKNOWN, 1,
                              METHOD_NEITHER, FILE_ANY_ACCESS)
#define MDR_SERVICE_UNMAP     CTL_CODE(FILE_DEVICE_UNKNOWN, 2,
                              METHOD_NEITHER, FILE_ANY_ACCESS)

#define MDR_STATUS_SUCCESS    1
#define MDR_STATUS_ERROR      0
```

The associated **C source file** `DTMap01.c` for the VxD is :

```
//Deep Trace VxD driver for Windows95
//Erich Barnstedt

//VERSION V1.0

#define PAGENUM(p)     (((ULONG)(p)) >> 12)
#define PAGEOFF(p)     (((ULONG)(p)) & 0xFFF)
#define PAGEBASE(p)    (((ULONG)(p)) & ~0xFFF)
#define _NPAGES_(p, k) ((PAGENUM((char*)p+(k-1))
                        - PAGENUM(p)) + 1)

#define   DEVICE_MAIN
#include  "dtmap01.h"
#undef    DEVICE_MAIN

//Create device descriptor block (DDB):
Declare_Virtual_Device(MAPDEV)

//Handle "DeviceIOControl" control message:
DefineControlHandler(W32_DEVICEIOCONTROL, OnW32Deviceiocontrol);

//Control dispatcher:
BOOL __cdecl ControlDispatcher(
   DWORD dwControlMessage,
   DWORD EBX,
   DWORD EDX,
   DWORD ESI,
   DWORD EDI,
   DWORD ECX)
{
   START_CONTROL_DISPATCH
      ON_W32_DEVICEIOCONTROL(OnW32Deviceiocontrol);
   END_CONTROL_DISPATCH

   return TRUE;
}

// Device Mapping Function:
PVOID MapDevice(PVOID PhysAddress, DWORD SizeInBytes)
{
   PVOID Linear;

   //calc number of mem pages required:
   ULONG nPages = _NPAGES_(PhysAddress, SizeInBytes);

   //set aside linear mem space:
   Linear = PageReserve(PR_PRIVATE,nPages,PR_FIXED);

   //do physical to linear translation (map into user space):
   PageCommitPhys(PAGENUM(Linear),nPages,PAGENUM(PhysAddress),
                  PC_INCR | PC_USER |PC_WRITEABLE);

   return (PVOID) ((ULONG)Linear+PAGEOFF(PhysAddress));
}

// Device Unmapping Function:
// Input: Base of linear memory block
// Output: None
VOID UnmapDevice(PVOID LinearAddress)
```

```
{
   //free linear memory:
   PageFree((MEMHANDLE)PAGEBASE(LinearAddress),0);
}

// W32_DEVICEIOCONTROL Handler for Win32 Applications:
// Input: The dioc_InBuf field of the parameter structure
// contains a pointer to the request structure
// Output: Fields of the request structure are updated
// according to the request.
DWORD OnW32Deviceiocontrol(PIOCTLPARAMS p)
{
   PMAPDEVREQUEST pReq;

   switch (p->dioc_IOCtlCode)
   {
   case DIOC_OPEN:
   case DIOC_CLOSEHANDLE:
      break;
   //map request:
   case MDR_SERVICE_MAP:
      pReq = *(PMAPDEVREQUEST*)p->dioc_InBuf;
      pReq->mdr_LinearAddress =
                       MapDevice(pReq->mdr_PhysicalAddress,
                       pReq->mdr_SizeInBytes);
      if (pReq->mdr_LinearAddress == NULL)
         pReq->mdr_Status = MDR_STATUS_ERROR;
      else
         pReq->mdr_Status =  MDR_STATUS_SUCCESS;

      break;
   //unmap request:
   case MDR_SERVICE_UNMAP:
      pReq = *(PMAPDEVREQUEST*)p->dioc_InBuf;
      UnmapDevice(pReq->mdr_LinearAddress);
      pReq->mdr_Status = MDR_STATUS_SUCCESS;
      break;
   default:
      return ERROR_INVALID_FUNCTION;
   }

   return DEVIOCTL_NOERROR;
}
```

An appropriate **makefile** `DTMap01.mak` needs to invoke special tools from the VxD development environment :

```
# DTMAP01.mak - makefile for VxD DTMap01

DEVICENAME = DTMAP01
DYNAMIC = 1
FRAMEWORK = C
DEBUG = 1
OBJECTS = DTMap01.OBJ

!include $(VTOOLSD)\include\vtoolsd.mak
!include $(VTOOLSD)\include\vxdtarg.mak

DTMap01.OBJ: DTMap01.c DTMap01.h
```

Windows NT Device Driver

An example Windows NT device driver is shown below. The **definition file** `mapmem.h` is as follows :

```
#define FILE_DEVICE_MAPMEM  0x00008000
#define MAPMEM_IOCTL_INDEX  0x800

// Define our own private IOCTL
#define MDR_SERVICE_MAP CTL_CODE(FILE_DEVICE_MAPMEM,
                                 MAPMEM_IOCTL_INDEX,
                                 METHOD_BUFFERED,
                                 FILE_ANY_ACCESS)
#define MDR_SERVICE_UNMAP CTL_CODE(FILE_DEVICE_MAPMEM,
                                   MAPMEM_IOCTL_INDEX+1,
                                   METHOD_BUFFERED,
                                   FILE_ANY_ACCESS)

// Our user mode app will pass an initialized structure like
// this down to the kernel mode driver

typedef struct _MapDevRequest
{
    int            mdr_Service_ID;
   unsigned long  *mdr_PhysicalAddress;
   int            mdr_SizeInBytes;
   unsigned long  *mdr_LinearAddress;
   unsigned char  mdr_Selector;
   unsigned char  mdr_Status;
} MAPDEVREQUEST, *PMAPDEVREQUEST;

#define MDR_STATUS_SUCCESS    1
#define MDR_STAUTS_ERROR      0
#define MAPDEV_Major          1
#define MAPDEV_Minor          0
#define MAPDEV_DeviceID       UNDEFINED_DEVICE_ID
#define MAPDEV_Init_Order     UNDEFINED_INIT_ORDER
```

The **C source file** `mapmem.c` is :

```
/*++
    A simple driver sample which shows how to map physical memory
    into a user-mode process's adrress space using the
    Zw*MapViewOfSection APIs.

Environment:

    kernel mode only

Notes:

    For the sake of simplicity this sample does not attempt to
    recognize resource conflicts with other drivers/devices.
    A real-world driver would call IoReportResource usage to
    determine whether or not the resource is available, and if
    so, register the resource under it's name.
```

```
--*/

#include "ntddk.h"
#include "mapmem.h"
#include "stdarg.h"

//function prototypes:
NTSTATUS MapMemDispatch(IN PDEVICE_OBJECT DeviceObject,
                        IN PIRP Irp);
VOID MapMemUnload(IN PDRIVER_OBJECT DriverObject);
NTSTATUS MapMemMapTheMemory(IN PDEVICE_OBJECT DeviceObject,
                            IN OUT PVOID ioBuffer,
                            IN ULONG inputBufferLength,
                            IN ULONG outputBufferLength);

#if DBG
#define MapMemKdPrint(arg) DbgPrint arg
#else
#define MapMemKdPrint(arg)
#endif

//no changes made in this routine:
NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject,
                     IN PUNICODE_STRING RegistryPath)
/*
Routine Description: Installable driver initialization entry
 point. This entry point is called directly by the I/O system.
 Arguments:
     DriverObject - pointer to the driver object
     RegistryPath - pointer to a unicode string representing the
                    path to driver-specific key in the registry
 Return Value:
    STATUS_SUCCESS if successful,
    STATUS_UNSUCCESSFUL otherwise
*/
{
   PDEVICE_OBJECT deviceObject = NULL;
   NTSTATUS       ntStatus;
   WCHAR          deviceNameBuffer[] = L"\\Device\\MapMem";
   UNICODE_STRING deviceNameUnicodeString;
   WCHAR          deviceLinkBuffer[] = L"\\DosDe vices\\MAPMEM";
   UNICODE_STRING deviceLinkUnicodeString;

   MapMemKdPrint (("MAPMEM.SYS: entering DriverEntry\n"));

   // Create an EXCLUSIVE device object (only 1 thread at a time
   // can make requests to this device)
   RtlInitUnicodeString (&deviceNameUnico deString,
                         deviceNameBuffer);
   ntStatus = IoCreateDevice(DriverObject, 0,
                             &deviceNameUnicodeString,
                             FILE_DEVICE_MAPMEM, 0, TRUE,
                             &deviceObject);

   if (NT_SUCCESS(ntStatus))
   {
      // Create dispatch points for device control, create
      // and close.
      DriverObject->MajorFunction[IRP_MJ_CREATE] =
                                   MapMemDispatch;
      DriverObject->MajorFunction[IRP_MJ_CLOSE] =
```

```
                                MapMemDispatch;
      DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] =
                                MapMemDispatch;
      DriverObject->DriverUnload = MapMemUnload;

      // Create a symbolic link, e.g. a name that a Win32 app can
      // specify to open the device
      RtlInitUnicodeString(&deviceLinkUnicodeString,
deviceLinkBuffer);

      ntStatus = IoCreateSymbolicLink (&deviceLinkUnicodeString,
                                       &deviceNameUnicodeString);

      if (!NT_SUCCESS(ntStatus))
      {
         // Symbolic link creation failed- note this & then
         // delete the device object (it's useless if a Win32
         // app can't get at it).
         MapMemKdPrint(("MAPMEM.SYS: IoCreateSymbolicLink
                         failed\n"));
         IoDeleteDevice (deviceObject);
      }
   }
   else
   {
      MapMemKdPrint (("MAPMEM.SYS: IoCreateDevice failed\n"));
   }
   return ntStatus;
}

//no changes made, works as of 7/7/98 (tested from Control Panel)
NTSTATUS MapMemDispatch(IN PDEVICE_OBJECT DeviceObject,
                        IN PIRP Irp)
/*
Routine Description: Process the IRPs sent to this device.
  Arguments:
        DeviceObject - pointer to a device object
        Irp          - pointer to an I/O Request Packet
Return Value:
        STATUS_SUCCESS if successful,
        STATUS_UNSUCCESSFUL otherwise
*/
{
   PIO_STACK_LOCATION irpStack;
   PVOID              ioBuffer;
   ULONG              inputBufferLength;
   ULONG              outputBufferLength;
   ULONG              ioControlCode;
   NTSTATUS           ntStatus;

   // Init to default settings- we only expect 1 type of
   // IOCTL to roll through here, all others an error.
   Irp->IoStatus.Status      = STATUS_SUCCESS;
   Irp->IoStatus.Information = 0;

   // Get a pointer to the current location in the Irp. This is
   // where the function codes and parameters are located.
   irpStack = IoGetCurrentIrpStackLocation(Irp);

   // Get the pointer to the input/output buffer and it's length
   ioBuffer = Irp->AssociatedIrp.SystemBuffer;
```

```
inputBufferLength =
       irpStack->Parameters.DeviceIoControl.InputBufferLength;
outputBufferLength =
       irpStack->Parameters.DeviceIoControl.OutputBufferLength;

switch (irpStack->MajorFunction)
{
case IRP_MJ_CREATE:
   MapMemKdPrint (("MAPMEM.SYS: IRP_MJ_CREATE\n"));
   break;

case IRP_MJ_CLOSE:
   MapMemKdPrint (("MAPMEM.SYS: IRP_MJ_CLOSE\n"));
   break;

case IRP_MJ_DEVICE_CONTROL:
   ioControlCode =
       irpStack->Parameters.DeviceIoControl.IoControlCode;
   switch (ioControlCode)
   {
   case MDR_SERVICE_MAP:
      Irp->IoStatus.Status = MapMemMapTheMemory (DeviceObject,
             ioBuffer, inputBufferLength, outputBufferLength);
      if (NT_SUCCESS(Irp->IoStatus.Status))
      {
         // Success! Set the following to sizeof(PVOID) to
         // indicate we're passing valid data back.
         Irp->IoStatus.Information = sizeof(PVOID);
         MapMemKdPrint (("MAPMEM.SYS: memory successfully
                          mapped\n"));
      }
      else
      {
         Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
         MapMemKdPrint (("MAPMEM.SYS: memory map failed
                          :(\n"));
      }
      break;

   case MDR_SERVICE_UNMAP:
      if (inputBufferLength >= sizeof(PVOID))
      {
         Irp->IoStatus.Status =
                ZwUnmapViewOfSection ((HANDLE) -1,
                                      *((PVOID *) ioBuffer));
         MapMemKdPrint (("MAPMEM.SYS: memory successfully
                          unmapped\n"));
      }
      else
      {
         Irp->IoStatus.Status = STATUS_INSUFFICIENT_RESOURCES;
         MapMemKdPrint (("MAPMEM.SYS: ZwUnmapViewOfSection
                          failed\n"));
      }
      break;

   default:
      MapMemKdPrint (("MAPMEM.SYS: unknown
                       IRP_MJ_DEVICE_CONTROL\n"));
      Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
      break;
```

```
        }
        break;
    }
    // DON'T get cute and try to use the status field of
    // the irp in the return status.  That IRP IS GONE as
    // soon as you call IoCompleteRequest.
    ntStatus = Irp->IoStatus.Status;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);

    // We never have pending operation so always return the
    // status code.
    return ntStatus;
}


//no changes made, works as of 7/7/98 (tested from Control Panel)
VOID MapMemUnload(IN PDRIVER_OBJECT DriverObject)
/*
Routine Description: Just delete the associated device & return.
  Arguments:
        DriverObject - pointer to a driver object
  Return Value:
        None
*/
{
    WCHAR deviceLinkBuffer[]  = L"\\DosDevices\\MAPMEM";
    UNICODE_STRING deviceLinkUnicodeString;

    // Free any resources
    //??????

    // Delete the symbolic link
    RtlInitUnicodeString (&deviceLinkUnicodeString,
                          deviceLinkBuffer);
    IoDeleteSymbolicLink (&deviceLinkUnicodeString);

    // Delete the device object
    MapMemKdPrint (("MAPMEM.SYS: unloading\n"));
    IoDeleteDevice (DriverObject->DeviceObject);
}



NTSTATUS MapMemMapTheMemory(IN PDEVICE_OBJECT DeviceObject,
                            IN OUT PVOID IoBuffer,
                            IN ULONG InputBufferLength,
                            IN ULONG OutputBufferLength)
/*
Routine Description:
    Given a physical address, maps this address into a user
    mode process's address space
Arguments:
    DeviceObject       - pointer to a device object
    IoBuffer           - pointer to the I/O buffer
    InputBufferLength  - input buffer length
    OutputBufferLength - output buffer length
Return Value:
    STATUS_SUCCESS if sucessful, otherwise
    STATUS_UNSUCCESSFUL,
    STATUS_INSUFFICIENT_RESOURCES,
    (other STATUS_* as returned by kernel APIs)
```

```
*/
{
   //PPHYSICAL_MEMORY_INFO ppmi =
   //                (PPHYSICAL_MEMORY_INFO) IoBuffer;
   PMAPDEVREQUEST pmdr = (PMAPDEVREQUEST) IoBuffer;
   INTERFACE_TYPE     interfaceType;
   ULONG              busNumber;
   PHYSICAL_ADDRESS   physicalAddress;
   ULONG              length;
   UNICODE_STRING     physicalMemoryUnicodeString;
   OBJECT_ATTRIBUTES  objectAttributes;
   HANDLE             physicalMemoryHandle  = NULL;
   PVOID              PhysicalMemorySection = NULL;
   ULONG              inIoSpace, inIoSpace2;
   NTSTATUS           ntStatus;
   PHYSICAL_ADDRESS   physicalAddressBase;
   PHYSICAL_ADDRESS   physicalAddressEnd;
   PHYSICAL_ADDRESS   viewBase;
   PHYSICAL_ADDRESS   mappedLength;
   BOOLEAN            translateBaseAddress;
   BOOLEAN            translateEndAddress;
   PVOID              virtualAddress;

   if ( ( InputBufferLength  < sizeof (MAPDEVREQUEST) ) ||
      ( OutputBufferLength < sizeof (PVOID) ) )
   {
      MapMemKdPrint (("MAPMEM.SYS: Insufficient input or output
                       buffer\n"));
      ntStatus = STATUS_INSUFFICIENT_RESOURCES;
      goto done;
   }

   interfaceType = 2; //EISA only
   busNumber     = 0; //MAY have to switch to 1 on Niestein!
   physicalAddress.HighPart= (LONG) 0x00000000;
   physicalAddress.LowPart = (LONG) pmdr->mdr_PhysicalAddress;
   inIoSpace = inIoSpace2  = 0;
   length                  = pmdr->mdr_SizeInBytes;

   // Get a pointer to physical memory...
   // - Create the name
   // - Initialize the data to find the object
   // - Open a handle to the oject and check the status
   // - Get a pointer to the object
   // - Free the handle

   RtlInitUnicodeString (&physicalMemoryUnicodeString,
                         L"\\Device\\PhysicalMemory");
   InitializeObjectAttributes (&objectAttributes,
                               &physicalMemoryUnicodeString,
                               OBJ_CASE_INSENSITIVE,
                               (HANDLE) NULL,
                               (PSECURITY_DESCRIPTOR) NULL);
   ntStatus = ZwOpenSection (&physicalMemoryHandle,
                             SECTION_ALL_ACCESS,
                             &objectAttributes);
   if (!NT_SUCCESS(ntStatus))
   {
      MapMemKdPrint (("MAPMEM.SYS: ZwOpenSection failed\n"));
      goto done;
   }
```

```
ntStatus = ObReferenceObjectByHandle (physicalMemoryHandle,
                                      SECTION_ALL_ACCESS,
                                      (POBJECT_TYPE) NULL,
                                      KernelMode,
                                      &PhysicalMem orySection,

(POBJECT_HANDLE_INFORMATION) NULL);
if (!NT_SUCCESS(ntStatus))
{
   MapMemKdPrint (("MAPMEM.SYS: ObReferenceObjectByHandle
                    failed\n"));
   goto close_handle;
}

// Initialize the physical addresses that will be translated
physicalAddressEnd = RtlLargeIntegerAdd (physicalAddress,

RtlConvertUlongToLargeInteger(length));

// Translate the physical addresses.
translateBaseAddress = HalTranslateBusAddress
                         (interfaceType,
                         busNumber,
                         physicalAddress,
                         &inIoSpace,
                         &physicalAddressBase);

 translateEndAddress = HalTranslateBusAddress
                         (interfaceType,
                         busNumber,
                         physicalAddressEnd,
                         &inIoSpace2,
                         &physicalAddressEnd);

if (!(translateBaseAddress && translateEndAddress))
{
   MapMemKdPrint (("MAPMEM.SYS: HalTranslatephysicalAddress
                    failed\n"));
   ntStatus = STATUS_UNSUCCESSFUL;
   goto close_handle;
}
// Calculate the length of the memory to be mapped
mappedLength = RtlLargeIntegerSubtract (physicalAddressEnd,
                                        physicalAddressBase);

// If the mappedlength is zero, somthing very weird happened
// in the HAL since the Length was checked against zero.

if (mappedLength.LowPart == 0)
{
   MapMemKdPrint (("MAPMEM.SYS: mappedLength.LowPart == 0\n"));
   ntStatus = STATUS_UNSUCCESSFUL;
   goto close_handle;
}
length = mappedLength.LowPart;

// If the address is in io space, just return the address,
// otherwise go through the mapping me chanism
if (inIoSpace) // will never be called in our implementation
               // since we never map I/O space!
{
```

```
            *((PVOID *) IoBuffer) = (PVOID) physicalAddressBase.LowPart;
        }
        else
        {
            // initialize view base that will receive the physical
            // mapped address after the MapViewOfSection call.
            viewBase = physicalAddressBase;

            // Let ZwMapViewOfSection pick an address
            virtualAddress = NULL;

            // Map the section
            ntStatus = ZwMapViewOfSection (physicalMemoryHandle,
                                           (HANDLE) -1,
                                           &virtualAddress,
                                           0L,
                                           length,
                                           &viewBase,
                                           &length,
                                           ViewShare,
                                           0,
                                         PAGE_READWRITE | PAGE_NOCACHE);

            if (!NT_SUCCESS(ntStatus))
            {
                MapMemKdPrint (("MAPMEM.SYS: ZwMapViewOfSection
                                 failed\n"));
                goto close_handle;
            }

            // Mapping the section above rounded the physical address
            // down to the nearest 64 K boundary. Now return a virtual
            // address that sits where we want by adding in the offset
            // from the beginning of the section.
            (ULONG) virtualAddress +=
                           (ULONG) physicalAddressBase.LowPart -
                           (ULONG) viewBase.LowPart;
            pmdr->mdr_LinearAddress = (ULONG *) virtualAddress;
            //return the IoBuffer with the message structure in it:
            *((PVOID *) IoBuffer) = virtualAddres s;
        }
        ntStatus = STATUS_SUCCESS;

    close_handle:
        ZwClose (physicalMemoryHandle);

    done:
        return ntStatus;
    }
```

The driver must have an associated **registry entry file** `mapmem.reg`, such as :

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\MAPMEM]
"Type"=dword:00000001
"Start"=dword:00000003
"Group"="Extended base"
"ErrorControl"=dword:00000001
```

For further information see :

1. Microsoft Windows NT 4.0 Driver Developer Kit online help files.

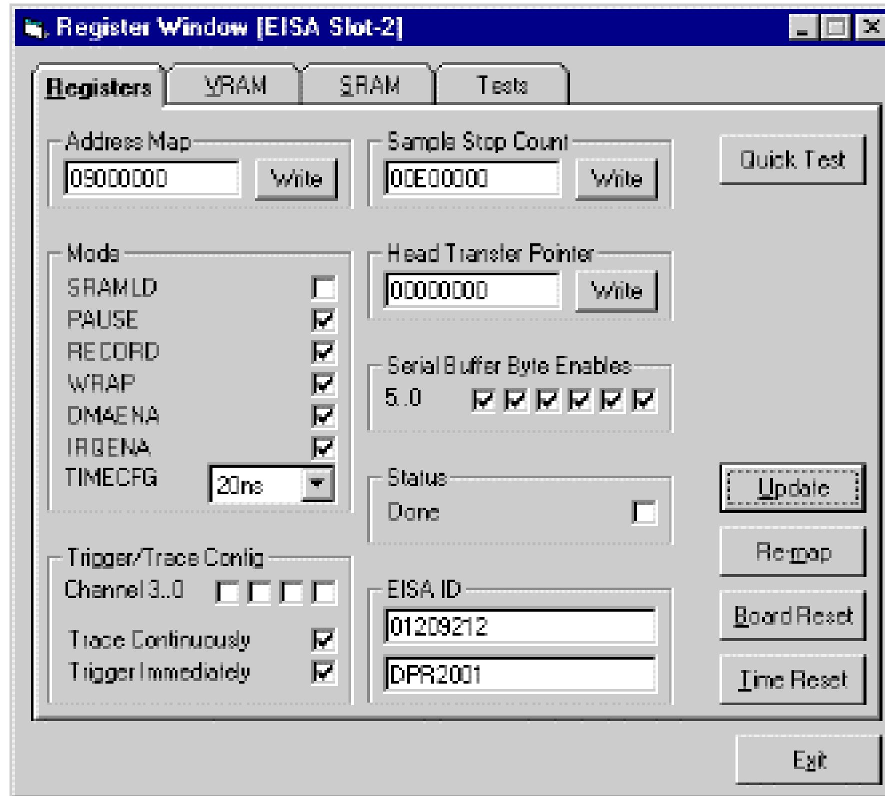
2. Microsoft Win32 Software Developer Kit online help files.

3. "The Windows NT Kernel-Mode Device Driver Cook Book, featuring Visual C++" by Ruediger R. Asche, found on the MSDN Library CD Version January 1997

4. "Writing Windows NT Kernel-Mode Drivers in C++" by Ruediger R. Asche found on the MSDN Library CD Version January 1997

Applications Software

An example of a software application is described below. This is designed to present the following tracer control panel. This application demonstrates how to control the DT200.1 via a windowed graphical user interface (GUI) that calls the API functions described above, and is very useful for testing and debugging.



The design specifications allow the user to :

(a) Read from and write to registers
(b) Read from VRAM
(c) Read from and write to SRAM
(d) Perform various board test procedures (including repeated stress testing)
(e) Save board configurations, test procedures and trigger information
(f) Save the VRAM contents to a file
(g) Analyse VRAM data to check board function ality and log results to a file

The interface consists of three main parts, a register window, a VRAM listing window and an SRAM listing window.

The register window displays the contents of all of the registers. The address map, stop count, head pointer and ID registers are displayed as test boxes containing hexadecimal data. The rest of the registers are displayed as check boxes indicating individual bit values. The time divisor is displayed in a drop down list with possible values of 20, 40, 80 and 160 ns.

In the VRAM listing window, the contents of a range of VRAM addresses can be displayed in a VRAM list box by entering the start and end addresses in text boxes and clicking on an Update

button. For example, entering hexadecimal 0 in the `From` box and hexadecimal 1000 in the `To` box will display the contents of the VRAM from address 0 to 1000 in the list box, one longword per row.

The SRAM window works in the same way.

When the program is started the board is reset, the address map register is set to the default value and the Deep Trace memory is mapped to a linear address to make it accessible by the program.

The control panel shown above is actually produced by a stand-alone Visual Basic (Version 5.0) application. Visual Basic programs are comprised of many small files that make them difficult to describe. Consequently an equivalent Java program is presented below. This software generates nearly identical results.

DT200.1 Control Panel Software using Java

This program, `DTApp`, is a stand-alone Java application with a user interface and the ability to control the DT200.1 Deep Trace Board through the `dt2001.dll` library. It is designed so that it can be used as a development platform for JNI (Java Native Interface) and JDBC (Java Database Connectivity).

At the top level the application consists of a Window with a card panel and a panel to control the cards. Each card consists of a number of panels implemented as separate classes. For example, there is a class for the Address Register panel. This allows the user interface code and the register control code to be kept in the same place. Thus by simply removing this class and the two lines which instantiate it, all address register functionality can be removed entirely from the application. This includes JNI code for the address register. This organization makes it easier to understand the code, as well as allowing the program to be changed very quickly.

At the uppermost level, the `DTControlWindow` class serves as a container for all of the other objects. It extends the `Frame` class to implement a window for the application. It also contains the `main()` entry point function for the application. A `CardLayout` manager is used to arrange the components (a panel for controlling the display of the cards and a panel for each card). There is no trace board functionality implemented at this level.

The `ControlPanel` class implements one of the cards contained in the `DTControlWindow`. It uses a `GridBagLayout` to arrange the DT200.1 register panels and a button bar. The `ControlPanel` handles events passed up from the button bar. This gives the button bar control of the registers.

There is a register panel class for each DT200.1 register. Each of these classes is able to make calls to the DLL functions which control the board using JNI. This means that each register panel exists as a component that can be removed and placed in new applications. It may be possible to alter these components to make their display optional. This would allow an application with higher level functionality to use the components but hide them from the user.

The `DataPanel` is a container for objects that deal with the trace data. The only such object at the moment is a `DatabasePanel`.

The `DBControlPanel` implements database connectivity using a JDBC-ODBC bridge. It has three member functions: a constructor, a function that stores a range of the Deep Trace VRAM in

a database and a function that closes the database connection (see the section **Database Connectivity** for more information).

The `ButtonPanel` class implements the button bar used in the `ControlPanel` class. All of the button events are passed up to the parent class.

Java Native Interface (JNI)

The Java Native Interface (JNI) provides a method of using Java code together with native code (e.g. a DLL written in C++). It is possible to perform two-way communication between Java code and native code but here we are only interested in calling functions in the `DT2001j.dll` library[1].

Native code libraries for use in Java Applications/Applets can be created by following a multi-step process :

1. Write the Java program. Declare any native methods in the classes where they are used. In a static code segment, make a call to the `loadLibrary()` function. This loads the native library into the Java class and maps the native method declaration to its implementation. For example[2] :

```
//
// AddressMapPanel
//
class AddressMapPanel extends Panel {

   // [JNI] Native (DLL) method declaration
   public native void setmap(int value);
   public native int getmap();

   static {
      System.loadLibrary("DT2001J");
   }
...
```

2. Compile the Java code written in the previous step :

```
> javac "DTControlWindow.java"
```

3. Create the header file for the native library. This will be used later to write the code for the native library. Use the `javah` program with the `-jni` argument :

```
> javah -jni "DTControlWindow.class"
```

   If the Java source file contained more than one class with native declarations, then multiple header files will be produced, one for each class. To make life easier, all of the header files can be merged into one. To do this, take one of the header files and copy all of the function prototypes from the other files into this one, and then perhaps give it a name that makes more

---

1 The DT2001j.dll library is a version of the DT2001.dll library with support for JNI. It allows the DT200.1 trace board to be controlled from the Java application. When this document was written, not all of the functionality implemented in DT2001.dll had been transferred to the JNI version (DT2001j.dll).

2 The `loadLibrary()` method appends (or prefixes) extensions to the library name in a system dependant way. For example, in a Windows95 environment, the entension `.dll` would be appended to `DT2001j` to give `DT2001j.DLL`. The user should not append this extension.

sense[3]. Make sure to leave the pre-processor instructions in the header file (the comment about not altering the file can be ignored!). The function prototypes can be altered later if required.

4. Write the native method implementation using the function prototypes created in Step 3. Remember to include the `jni.h` header file[4].

5. Compile the native source code to create the shared library.

Java arrays and strings take a different format to arrays in C and C++. This means that arrays and strings passed from a Java program to a native method cannot be manipulated directly. The `jni.h` library includes some functions that allow native methods to use Java arrays and strings but this requires some extra code to be written. Refer to the Sun documentation on JNI.

Database Connectivity

JDBC[5] provides a standard API for the development of database applications in Java. A single Java application written using JDBC can be used to control different types of database, e.g. Sybase, Oracle, etc..

To use JDBC, a DBMS driver is required for each type of database accessed from the program. For the purposes of the DT200.1 Java application, a JDBC-ODBC bridge (category 1 driver) is used. This driver is distributed with the JDBC package from Sun. To set up basic database connectivity on a local machine :

1. Load the JDBC-ODBC bridge driver.
2. Find the driver to process the URL.
3. Create a property list with the information required to connect to the database.
4. Connect to the driver.
5. Execute the SQL query.

The following code segment illustrates the above procedure :

```
//
// Simple Database Connectivity
//
// Instantiate this class from another method
//
class DBControl {

   String url;     // Store the URL provided
   Connection con;

   public DBControl(String DBName) {

      url = DBName;
```

3 Try to avoid underscore characters when naming native methods in the Java code. The `javah` program creates new names by prefixing some information to the identifier and using underscores. If underscores are also used in the original method declaration, the native function names will become harder to follow.

4 The path containing the `jni.h` header file may need to be added to the include path of the compiler used to compile the `.dll` file.

5 Microsoft Visual J++ does not support JNI. As a result of this, there is also no support for JDBC. JDBC is the Sun API that gives Java Applications/Applets database functionality.

```
    try {
       // Load the JDBC-ODBC bridge driver
       Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

       // Find the driver to process the URL
       Driver jdbcDriver = DriverManager.getDriver(url);

       // Create and set up the property list with the
       // correct security information
       Properties propertyList = new Properties();
       propertyList.put("user", "dtuser");
       propertyList.put("password", "dtpassword");

       // This can be used to provide verbose logging
       DriverManager.setLogStream(System.out);

       // Connect to the driver
       con = DriverManager.getConnection(url, propertyList);

    }
    catch (SQLException ex) {
       System.out.println("Exception while trying to connect");
       while (ex != null) {
          // The following information is essential for
          // finding errors
          System.out.println("SQLState: " + ex.getSQLState());
          System.out.println("Message: " + ex.getMessage());
          System.out.println("Vendor: " + ex.getErrorCode());
          System.out.println("URL: " + url);
          ex = ex.getNextException();
          System.out.println("");
       }
    }
    catch (java.lang.Exception ex) {
       ex.printStackTrace();
    }
 }

 // Try to store some sample data in the database.
 // The test database should have six integer fields per record.
 public void storeData() {
    try {
       // Create a statement
       Statement pstmt = con.createStatement();

       // Execute a SQL statement
       // Use executeQuery for queries and executeUpdate
       // for altering the database (executeQuery expects
       // a ResultSet to be returned)
       pstmt.executeUpdate("INSERT INTO TraceBoardData
                            VALUES (10,20,30,40,50,60)");
    }
    catch (SQLException ex) {
       System.out.println("Exception while executing query");
       while (ex != null) {
          System.out.println("SQLState: " + ex.getSQLState());
          System.out.println("Message: " + ex.getMessage());
          System.out.println("Vendor: " + ex.getErrorCode());
          ex = ex.getNextException();
          System.out.println("");
       }
```

```
        }
    }
}
```

This example uses `Statement` objects to execute SQL statements. Every SQL statement passed to the DBMS must undergo certain procedures before it is executed. This causes undesirable delays. A `Prepared Statement` is similar to a `Statement` but it is pre-compiled using value placeholders that can be filled in later. This allows multiple statements, which differ only in the values passed to the database, to be executed very quickly.

However, this method is still not particularly fast. One alternative may be to buffer data in a large temporary file as it is being traced and add this data to the database at a later stage. Another possibility is BLOBs – Binary Large OBjects[6]. The entire trace data from a single DT200.1 (12MB) could be stored as a single record in a database using a binary stream (see the Java documentation).

In the code given below, the database consists of five 16bit integer fields per record, called `Board1Counter`, `Board1Data1`, `Board1Data2`, `Board2Data1` and `Board2data2`. These would need to be changed if the data storage method were considered too slow. The code also contains a known flaw in the database connectivity: the SQL error message (`invalid property value (null)`) is sometimes generated, after which data storage terminates.

To set up ODBC on Windows95, invoke 32 bit ODBC in the Windows control panel. In the User tab specify a database and call it `dtdb`. Give it a description and in the Advanced tab set the username to `dtuser` and the password to `dtpassword`. Any changes to these settings will require changes to the code until support for user input or the windows registry is added to the application.

Remote Method Invocation

The JDBC-ODBC bridge is ideally suited to a three-tier system using RMI. Such a system could consist of a database controlled by an RMI server which receives requests from a remote client, possible an applet. The RMI server could be a Java application running on a machine with a HTTP server. For this arrangement, the application could be split into two parts as follows :

(a) The program could be turned into an applet.
(b) All calls to the `dt2001.dll` library functions could be replaced with code to communicate with an RMI server application.
(c) The calls to the DLL could be placed in an RMI server application which could service requests from the client applet.
(d) All of the database functionality could be transferred to the RMI server .

Software Support for Multiple Trace Boards

This would require some changes to both the Java application and the DLL. EISA devices can be identified by their I/O base address, which is slot dependent. Any DLL function calls that wishes to read/write from/to registers would need to either supply an identifier which the DLL could associate with a base address (possibly obtaining this from the registry) or the base address itself. The VRAM base addresses for each board could then be obtained by a call to the `getmap()` function, specifying an identifier or base address. Using an identifier is probably the most generic since it adds another layer of abstraction.

---

[6] Microsoft Access does not support BLOBs so this method would require some other database.

Execution

The computer used to compile the Java code must have the Java `bin` directory in its search path. To compile the Java code, enter :

```
> javac "DTControlWindow.java"
```

To run the program, enter :

```
> java "DTControlWindow"
```

To run the program on a different computer to the one on which it was compiled, copy the `.class` files to some directory and run the program in the usual way.

Java Sources

The **class control file** `DBControl.h` is :

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class DBControl */

#ifndef _Included_DBControl
#define _Included_DBControl
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     DBControl
 * Method:    mapRAM
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_DBControl_mapRAM
  (JNIEnv *, jobject);

/*
 * Class:     DBControl
 * Method:    unmapRAM
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_DBControl_unmapRAM
  (JNIEnv *, jobject);

/*
 * Class:     DBControl
 * Method:    getRAM
 * Signature: (I[SII)I
 */
JNIEXPORT jint JNICALL Java_DBControl_getRAM
  (JNIEnv *, jobject, jint, jshortArray, jint, jint);

#ifdef __cplusplus
}
#endif
#endif
```

The application's native **Java source file** `DTControlWindow.java` is as follows :

```
/*-----------------------------------------*\
| DTApp                                     |
| Deep Trace 200.1 Java Test Application    |
|                                           |
| DTControlWindow.java                      |
| Last Modified: 03-09-1998                 |
\*-----------------------------------------*/

import java.awt.*;
import java.sql.*;              // JDBC
import java.util.*;
import java.util.Date;
```

```
//
// DTControlWindow
//
// Application parent window
//

public class DTControlWindow extends Frame {

   // Not sure what this does!
   private boolean inAnApplet = true;

   // Panel to hold the different cards (REGISTERS, VRAM, ...)
   Panel cards;

   // Display current card name
   Label dsp;

   // Card select control
   Button c1;
   Button c2;

   // Panel names - also used to identify panels in the code
   final static String CONTROLPANEL = "DT Control";
   final static String DATAPANEL = "Data";

   public DTControlWindow () {

      setFont(new Font("Helvetica", Font.PLAIN, 10));

      // Panel to hold the cards
      cards = new Panel();
      cards.setLayout(new CardLayout());

      // Panel to hold choice control
      Panel choicePanel = new Panel();
      choicePanel.setLayout(new GridLayout(1,8,0,0));
      choicePanel.setBackground(Color.lightGray);
      c1 = new Button(CONTROLPANEL);
      c2 = new Button(DATAPANEL);
      dsp = new Label(CONTROLPANEL);

      // Add panel title display
      choicePanel.add(dsp);

      // Padding to force buttons to right of screen
      // There might be a better way
      choicePanel.add(new Panel());
      choicePanel.add(new Panel());
      choicePanel.add(new Panel());
      choicePanel.add(new Panel());
      choicePanel.add(new Panel());

      // Add buttons
      choicePanel.add(c1);
      choicePanel.add(c2);

      add("North", choicePanel);

      // Cards
      ControlPanel p1 = new ControlPanel();
      DataPanel p2 = new DataPanel();
```

```
      // ... more cards ...
      // ... more cards ...

      // Add the cards
      cards.add(CONTROLPANEL, p1);
      cards.add(DATAPANEL, p2);

      // Add the card container to the window
      add("Center", cards);

   }

   // Event handler to change cards
   public boolean action(Event evt, Object arg) {
      Object target = evt.target;

      if (target == c1) {
         // Change card
         ((CardLayout)cards.getLayout()).show(cards,
                                                 CONTROLPANEL);

         // Change card title
         dsp.setText(CONTROLPANEL);
         return true;
      }

      if (target == c2) {
         // Change card
         ((CardLayout)cards.getLayout()).show(cards,DATAPANEL);

         // Change card title
         dsp.setText(DATAPANEL);
         return true;
      }

      // Allow event to be 'passed up' to allow ancestors
      // to handle the event if they need to.
      return false;
   }

   // Tidy up on Application exit (not sure about
   // this, it was a cut and paste job).
   public boolean handleEvent(Event e) {
      if (e.id == Event.WINDOW_DESTROY) {
         if (inAnApplet) {
            dispose();
            return true;
         } else {
            System.exit(0);
         }
      }
      return super.handleEvent(e);
   }

   // Entry point (called when app starts)
   public static void main(String args[]) {
      DTControlWindow window = new DTControlWindow();
      window.inAnApplet = false;

      window.setTitle("Deep Trace Application
                       (Standalone Version)");
```

```
      window.resize(580,390);
      window.show();
   }
}

//
// ControlPanel
//
// Container for the DT Control card
// Contains registers and button bar
//
class ControlPanel extends Panel {

   AddressMapPanel addrmap;
   SampleStopCountPanel ssc;
   HeadTransferPtrPanel htp;
   ModePanel mode;
   TriggerTraceConfigPanel ttc;
   SerialBufferPanel sbbe;
   StatusPanel stat;
   IDPanel id;
   ButtonPanel btnbar;

   public ControlPanel() {

      // Use a GridBagLayout
      GridBagLayout gridbag = new GridBagLayout();
      GridBagConstraints c = new GridBagConstraints( );
      setLayout(gridbag);
      setBackground(Color.lightGray);

      // Default constraint for all components
      c.fill = GridBagConstraints.BOTH;

      // Set constraints for individual components and add
      // them to the panel
      addrmap = new AddressMapPanel();
      c.gridx = 0;      // X Position
      c.gridy = 0;      // Y Position
      c.gridwidth = 1;  // X Size
      c.gridheight = 1; // Y Size
      c.weightx = 0.0;  // X Weight (witchcraft, see API help)
      c.weighty = 0.0;  // Y Weight
      gridbag.setConstraints(addrmap, c); // Apply constraints
      add(addrmap);     // Add component

      ssc = new SampleStopCountPanel();
      c.gridx = 1;
      c.gridy = 0;
      c.gridwidth = 1;
      c.gridheight = 1;
      c.weightx = 0.0;
      c.weighty = 0.0;
      gridbag.setConstraints(ssc, c);
      add(ssc);

      htp = new HeadTransferPtrPanel();
      c.gridx = 2;
      c.gridy = 0;
      c.gridwidth = 1;
      c.gridheight = 1;
```

```
c.weightx = 0.0;
c.weighty = 0.0;
gridbag.setConstraints(htp, c);
add(htp);

mode = new ModePanel();
c.gridx = 0;
c.gridy = 1;
c.gridwidth = 1;
c.gridheight = 3;
c.weightx = 0.0;
c.weighty = 0.0;
gridbag.setConstraints(mode, c);
add(mode);

ttc = new TriggerTraceConfigPanel();
c.gridx = 1;
c.gridy = 1;
c.gridwidth = 1;
c.gridheight = 3;
c.weightx = 0.0;
c.weighty = 0.0;
gridbag.setConstraints(ttc, c);
add(ttc);

sbbe = new SerialBufferPanel();
c.gridx = 2;
c.gridy = 1;
c.gridwidth = 1;
c.gridheight = 3;
c.weightx = 0.0;
c.weighty = 0.0;
gridbag.setConstraints(sbbe, c);
add(sbbe);

stat = new StatusPanel();
c.gridx = 0;
c.gridy = 4;
c.gridwidth = 1;
c.gridheight = 1;
c.weightx = 0.0;
c.weighty = 0.0;
gridbag.setConstraints(stat, c);
add(stat);

id = new IDPanel();
c.gridx = 1;
c.gridy = 4;
c.gridwidth = 2;
c.gridheight = 1;
c.weightx = 0.0;
c.weighty = 0.0;
gridbag.setConstraints(id, c);
add(id);

btnbar= new ButtonPanel();
c.gridx = 3;
c.gridy = 0;
c.gridwidth = 1;
c.gridheight = 5;
c.weightx = 1.0;
```

```
        c.weighty = 1.0;
        gridbag.setConstraints(btnbar, c);
        add(btnbar);
    }

    // Component padding
    public Insets insets() {
        return new Insets(10,10,10,10);
    }

        // Draw border
    public void paint(Graphics g) {
        Dimension d = size();
        Color bg = getBackground();

        g.setColor(bg);
        g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
        g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
    }

    // This function updates all the components in the panel
    public void Update() {
        addrmap.Update();
        ssc.Update();
        mode.Update();
        htp.Update();
        ttc.Update();
        sbbe.Update();
        stat.Update();
        id.Update();
    }

    // The button bar allows all events on its buttons
    // to pass up to here (register panel). This allows
    // the event handlers for the buttons to control
    // other objects.
    public boolean action(Event e, Object arg) {

        Object target = e.target;

        // Register update button event
        if (target == btnbar.rupdate) {
            Update();
            return true;
        }

        // Board reset button event
        if (target == btnbar.breset) {
            return true;
        }

        // Time reset button event
        if (target == btnbar.treset) {
            return true;
        }
        return false;
    }
}

//
// DataPanel
```

```
//
// At the moment this just controls the database
//
class DataPanel extends Panel {

   DatabasePanel dbcapture;

   public DataPanel() {

      setLayout(new GridLayout(3,3,10,10));
      setBackground(Color.lightGray);

      dbcapture = new DatabasePanel();
      add(dbcapture);

      // Padding for rest of grid
      add(new Panel());
      add(new Panel());
      add(new Panel());
      add(new Panel());
      add(new Panel());
      add(new Panel());
      add(new Panel());
      add(new Panel());
   }

   public Insets insets() {
      return new Insets(10,10,10,10);
   }

   public void paint(Graphics g) {
      Dimension d = size();
      Color bg = getBackground();

      g.setColor(bg);
      g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
      g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
   }
}

//
// AddressMapPanel
//
class AddressMapPanel extends Panel {

   // [JNI] Native (DLL) method declaration
   public native void setmap(int value);
   public native int getmap();

   static {
      System.loadLibrary("DT2001J");
   }

   Label ttl;
   TextField adr;
   Button wrt;

   public AddressMapPanel() {

      setLayout(new BorderLayout());
      setBackground(Color.lightGray);
```

```
        ttl = new Label("Address Map");
        add("North", ttl);

        adr = new TextField(14);
        add("West", adr);

        wrt = new Button();
        wrt.setLabel("Write");
        add("East", wrt);
    }


    public Insets insets() {
        return new Insets(6,6,6,6);
    }


    public void paint(Graphics g) {
        Dimension d = size();
        Color bg = getBackground();

        g.setColor(bg);
        g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
        g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
    }


    // Event handler
    public boolean action(Event e, Object arg) {

        Object target = e.target;

        if (target == wrt) {
            Integer i;
            i = new Integer(0);
            try {
                setmap(i.parseInt(adr.getText()));
            }
            catch (NumberFormatException exception){
            }
            return true;
        }
        return false;
    }


    // Update the contents of this panel
    public void Update() {
        Integer i = new Integer(0);
        adr.setText(i.toHexString(getmap()));
    }

}


//
// HeadTransferPtrPanel
//
class HeadTransferPtrPanel extends Panel {
```

```
// [JNI] Native (DLL) method declaration
public native void setheadptr( int value);
public native int getheadptr();

static {
   System.loadLibrary("DT2001J");
}

Label ttl;
TextField htp;
Button wrt;

public HeadTransferPtrPanel() {

   setLayout(new BorderLayout());
   setBackground(Color.lightGray);

   ttl = new Label("Head Transfer Pointer");
   add("North", ttl);

   htp = new TextField(14);
   add("West", htp);

   wrt = new Button();
   wrt.setLabel("Write");
   add("East", wrt);
}

// Event handler
public boolean action(Event e, Object arg) {

   Object target = e.target;

   if (target == wrt) {
      Integer i;
      i = new Integer(0);
      try {
         setheadptr(i.parseInt(htp.getText()));
      }
      catch (NumberFormatException exception){
      }
      return true;
   }
   return false;
}

public Insets insets() {
   return new Insets(6,6,6,6);
}

public void paint(Graphics g) {
   Dimension d = size();
   Color bg = getBackground();

   g.setColor(bg);
   g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
   g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
}

// Update the contents of this panel
public void Update() {
```

```
            Integer i = new Integer(0);
            htp.setText(i.toHexString(getheadptr()));
        }
    }

    //
    // IDPanel
    //
    class IDPanel extends Panel {

        // [JNI] Native method declaration
        public native int getEISAID();
        public native String gettextEISAID();

        static {
            System.loadLibrary("DT2001J");
        }

        Label ttl, ttl2;
        TextField idnum, idtext;

        public IDPanel () {

            setLayout(new GridLayout(2,2));
            setBackground(Color.lightGray);

            ttl = new Label("EISA ID (Num)");
            add(ttl);

            idnum = new TextField(14);
            add(idnum);

            ttl2 = new Label("(Text)");
            add(ttl2);

            idtext = new TextField(14);
            add(idtext);
        }

        public Insets insets() {
            return new Insets(6,6,6,6);
        }

        public void paint(Graphics g) {
            Dimension d = size();
            Color bg = getBackground();

            g.setColor(bg);
            g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
            g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
        }

        public void Update() {
            Integer i = new Integer(0);
            String tmpstr;
            tmpstr = gettextEISAID();
            idtext.setText(tmpstr);
            idnum.setText(i.toHexString(getEISAID()));

        }
    }
```

```
//
// ModePanel
//
class ModePanel extends Panel {

   // Bit field definitions
   final static int DT_SRAMLD =     0x00000001;
   final static int DT_PAUSE =      0x00000002;
   final static int DT_RECORD =     0x00000004;
   final static int DT_WRAP =       0x00000008;
   final static int DT_IRQENA =     0x00000040;
   final static int DT_DMAENA =     0x00000080;

   final static int DT_TC =         0x00000030;
   final static int DT_TC20 =       0x00000000;
   final static int DT_TC40 =       0x00000010;
   final static int DT_TC80 =       0x00000020;
   final static int DT_TC160 =      0x00000030;

   final static String NS20 =       "TIMECFG = 20ns";
   final static String NS40 =       "TIMECFG = 40ns";
   final static String NS80 =       "TIMECFG = 80ns";
   final static String NS160 =      "TIMECFG = 160ns";

   // [JNI] Native (DLL) method declaration
   public native void setmode(int value);
   public native int getmode();

   static {
      System.loadLibrary("DT2001J");
   }

   Label ttl;
   Checkbox sramld, pause, record, wrap, irqena, dmaena;
   Choice timecfg;

   public ModePanel () {

      setLayout(new GridLayout(8,1));
      setBackground(Color.lightGray);

      ttl = new Label("Mode");
      add(ttl);

      sramld = new Checkbox("SRAMLD");
      add(sramld);

      pause = new Checkbox("PAUSE");
      add(pause);

      record = new Checkbox("RECORD");
      add(record);

      wrap = new Checkbox("WRAP");
      add(wrap);

      irqena = new Checkbox("IRQENA");
      add(irqena);

      dmaena = new Checkbox("DMAENA");
```

```
      add(dmaena);

      timecfg = new Choice();
      timecfg.addItem(NS20);
      timecfg.addItem(NS40);
      timecfg.addItem(NS80);
      timecfg.addItem(NS160);
      add(timecfg);
   }

   // Event handler
   public boolean action(Event e, Object arg) {

      Object target = e.target;

      if (target == sramld) {
         if (sramld.getState()) { // Box ticked
            setmode(getmode() | DT_SRAMLD);
         }
         else { // Box unticked
            setmode(getmode() & ~DT_SRAMLD);
         }
         return true;
      }
      if (target == pause) {
         if (pause.getState()) { // Box ticked
            setmode(getmode() | DT_PAUSE);
         }
         else { // Box unticked
            setmode(getmode() & ~DT_PAUSE);
         }
         return true;
      }
      if (target == record) {
         if (record.getState()) { // Box ticked
            setmode(getmode() | DT_RECORD);
         }
         else { // Box unticked
            setmode(getmode() & ~DT_RECORD);
         }
         return true;
      }
      if (target == wrap) {
         if (wrap.getState()) { // Box ticked
            setmode(getmode() | DT_WRAP);
         }
         else { // Box unticked
            setmode(getmode() & ~DT_WRAP);
         }
         return true;
      }
      if (target == irqena) {
         if (irqena.getState()) { // Box ticked
            setmode(getmode() | DT_IRQENA);
         }
         else { // Box unticked
            setmode(getmode() & ~DT_IRQENA);
         }
         return true;
      }
      if (target == dmaena) {
```

```
         if (dmaena.getState()) { // Box ticked
            setmode(getmode() | DT_DMAENA);
         }
         else { // Box unticked
            setmode(getmode() & ~DT_DMAENA);
         }
         return true;
      }
      if (target == timecfg) {
         setmode(getmode() & ~0x000000C0);
         switch (timecfg.getSelectedIndex()) {
         case 0 :
            break;
         case 1 :
            setmode(getmode() | DT_TC40);
            break;
         case 2 :
            setmode(getmode() | DT_TC80);
            break;
         case 3 :
            setmode(getmode() | DT_TC160);
            break;
         default :
            break;
         }
      }
      return false;
   }

      public Insets insets() {
      return new Insets(6,6,6,6);
   }

   public void paint(Graphics g) {
      Dimension d = size();
      Color bg = getBackground();

      g.setColor(bg);
      g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
      g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
   }

   // Update the contents of this panel
   public void Update() {
      int val = getmode();
      sramld.setState(((val & DT_SRAMLD) == DT_SRAMLD)
                        ? true : false);
      pause.setState(((val & DT_PAUSE) == DT_PAUSE)
                        ? true : false);
      record.setState(((val & DT_RECORD) == DT_RECORD)
                        ? true : false);
      wrap.setState(((val & DT_WRAP) == DT_WRAP)
                        ? true : false);
      irqena.setState(((val & DT_IRQENA) == DT_IRQENA)
                        ? true : false);
      dmaena.setState(((val & DT_DMAENA) == DT_DMAENA)
                        ? true : false);

      if ((val & DT_TC) == DT_TC20) {
         timecfg.select(NS20);
      }
```

```
        else if ((val & DT_TC) == DT_TC40) {
           timecfg.select(NS40);
        }
        else if ((val & DT_TC) == DT_TC80) {
           timecfg.select(NS80);
        }
        else {
           timecfg.select(NS160);
        }
     }
  }

  //
  // SampleStopCountPanel
  //
  class SampleStopCountPanel extends Panel {

     // [JNI] Native (DLL) method declaration
     public native void setstopcount(int value);
     public native int getstopcount();

     static {
        System.loadLibrary("DT2001J");
     }

     Label ttl;
     TextField ssc;
     Button wrt;

     public SampleStopCountPanel() {

        setLayout(new BorderLayout());
        setBackground(Color.lightGray);

        ttl = new Label("Sample Stop Count");
        add("North", ttl);

        ssc = new TextField(14);
        add("West", ssc);

        wrt = new Button();
        wrt.setLabel("Write");
        add("East", wrt);

     }

     // Event handler
     public boolean action(Event e, Object arg) {

        Object target = e.target;

        if (target == wrt) {
           Integer i;
           i = new Integer(0);
           try {
              setstopcount(i.parseInt(ssc.getText()));
           }
           catch (NumberFormatException exception){
           }
           return true;
        }
```

```
            return false;
        }

        public Insets insets() {
            return new Insets(6,6,6,6);
        }


        public void paint(Graphics g) {
            Dimension d = size();
            Color bg = getBackground();

            g.setColor(bg);
            g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
            g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
        }

        // Update the contents of this panel
        public void Update() {
            Integer i = new Integer(0);
            ssc.setText(i.toHexString(getstopcount()));
        }
    }

    //
    // SerialBufferPanel
    //
    class SerialBufferPanel extends Panel {

        // Bit field definitions
        final static int DT_BYTE0 =     0x00000001;
        final static int DT_BYTE1 =     0x00000002;
        final static int DT_BYTE2 =     0x00000004;
        final static int DT_BYTE3 =     0x00000008;
        final static int DT_BYTE4 =     0x00000010;
        final static int DT_BYTE5 =     0x00000020;

        // [JNI] Native (DLL) method declaration
        public native void setsbufoutputenas(int value);
        public native int getsbufoutputenas();

        static {
            System.loadLibrary("DT2001J");
        }

        Label ttl;
        Checkbox byte0, byte1, byte2, byte3, byte4, byte5;

        public SerialBufferPanel () {

            setLayout(new GridLayout(7,1));
            setBackground(Color.lightGray);

            ttl = new Label("Byte Enables");
            add(ttl);

            byte0 = new Checkbox("Byte 0");
            add(byte0);

            byte1 = new Checkbox("Byte 1");
            add(byte1);
```

```
    byte2 = new Checkbox("Byte 2");
    add(byte2);

    byte3 = new Checkbox("Byte 3");
    add(byte3);

    byte4 = new Checkbox("Byte 4");
    add(byte4);

    byte5 = new Checkbox("Byte 5");
    add(byte5);
}

// Event handler
public boolean action(Event e, Object arg) {

    Object target = e.target;

    if (target == byte0) {
        if (byte0.getState()) { // Box ticked
            setsbufoutputenas(getsbufoutputenas() | DT_BYTE0);
        }
        else { // Box unticked
            setsbufoutputenas(getsbufoutputenas() & ~DT_BYTE0);
        }
        return true;
    }

    if (target == byte1) {
        if (byte1.getState()) { // Box ticked
            setsbufoutputenas(getsbufoutputenas() | DT_BYTE1);
        }
        else { // Box unticked
            setsbufoutputenas(getsbufoutputenas() & ~DT_BYTE1);
        }
        return true;
    }
    if (target == byte2) {
        if (byte2.getState()) { // Box ticked
            setsbufoutputenas(getsbufoutputenas() | DT_BYTE2);
        }
        else { // Box unticked
            setsbufoutputenas(getsbufoutputenas() & ~DT_BYTE2);
        }
        return true;
    }
    if (target == byte3) {
        if (byte3.getState()) { // Box ticked
            setsbufoutputenas(getsbufoutputenas() | DT_BYTE3);
        }
        else { // Box unticked
            setsbufoutputenas(getsbufoutputenas() & ~DT_BYTE3);
        }
        return true;
    }
    if (target == byte4) {
        if (byte4.getState()) { // Box ticked
            setsbufoutputenas(getsbufoutputenas() | DT_BYTE4);
        }
        else { // Box unticked
```

```
               setsbufoutputenas(getsbufoutputenas() & ~DT_BYTE4);
            }
            return true;
         }
         if (target == byte5) {
            if (byte5.getState()) { // Box ticked
               setsbufoutputenas(getsbufoutputenas() | DT_BYTE5);
            }
            else { // Box unticked
               setsbufoutputenas(getsbufoutputenas() & ~DT_BYTE5);
            }
            return true;
         }
         return false;
      }

      public Insets insets() {
         return new Insets(6,6,6,6);
      }

      public void paint(Graphics g) {
         Dimension d = size();
         Color bg = getBackground();

         g.setColor(bg);
         g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
         g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
      }

      // Update the contents of this panel
      public void Update() {
         int val = getsbufoutputenas();
         byte0.setState(((val & DT_BYTE0) == DT_BYTE0)
                        ? true : false);
         byte1.setState(((val & DT_BYTE1) == DT_BYTE1)
                        ? true : false);
         byte2.setState(((val & DT_BYTE2) == DT_BYTE2)
                        ? true : false);
         byte3.setState(((val & DT_BYTE3) == DT_BYTE3)
                        ? true : false);
         byte4.setState(((val & DT_BYTE4) == DT_BYTE4)
                        ? true : false);
         byte5.setState(((val & DT_BYTE5) == DT_BYTE5)
                        ? true : false);

      }
   }

   //
   // StatusPanel
   //
   class StatusPanel extends Panel {

      // Bit field definitions
      final static int DT_DONE =      0x00008000;

      // [JNI] Native (DLL) method declaration
      public native int getstatus();

      static {
         System.loadLibrary("DT2001J");
```

```
    }

    Label ttl;
    Checkbox done;

    public StatusPanel() {

       setLayout(new GridLayout(2,1));
       setBackground(Color.lightGray);

       ttl = new Label("Status");
       add(ttl);

       done = new Checkbox("Done");
       add(done);
    }

    public Insets insets() {
       return new Insets(6,6,6,6);
    }

    public void paint(Graphics g) {
       Dimension d = size();
       Color bg = getBackground();

       g.setColor(bg);
       g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
       g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
    }

    // Update the contents of this panel
    public void Update() {
       //int val = getstatus();
       //done.setState(((val & DT_DONE) == DT_DONE)
                       ? true : false);
    }
 }

 //
 // TriggerTraceConfigPanel
 //
 class TriggerTraceConfigPanel extends Panel {

    // Bit field definitions
    final static int DT_CH0 =        0x00000001;
    final static int DT_CH1 =        0x00000002;
    final static int DT_CH2 =        0x00000004;
    final static int DT_CH3 =        0x00000008;
    final static int DT_TC =         0x00000010;
    final static int DT_TI =         0x00000020;

    // [JNI] Native (DLL) method declarations
    public native void settrigconfig(int value);
    public native int gettrigconfig();


    static {
       System.loadLibrary("DT2001J");
    }

    Label ttl;
```

```
Checkbox ch0, ch1, ch2, ch3, tc, ti;

public TriggerTraceConfigPanel () {

   setLayout(new GridLayout(7,1));
   setBackground(Color.lightGray);

   ttl = new Label("Trigger/Trace Config");
   add(ttl);

   ch0 = new Checkbox("Channel 0");
   add(ch0);

   ch1 = new Checkbox("Channel 1");
   add(ch1);

   ch2 = new Checkbox("Channel 2");
   add(ch2);

   ch3 = new Checkbox("Channel 3");
   add(ch3);

   tc = new Checkbox("Trace Continuously");
   add(tc);

   ti = new Checkbox("Trigger Immediately");
   add(ti);
}

// Event handler
public boolean action(Event e, Object arg) {

   Object target = e.target;

   if (target == ch0) {
      if (ch0.getState()) { // Box ticked
         settrigconfig(gettrigconfig() | DT_CH0);
      }
      else { // Box unticked
         settrigconfig(gettrigconfig() & ~DT_CH0);
      }
      return true;
   }
   if (target == ch1) {
      if (ch1.getState()) { // Box ticked
         settrigconfig(gettrigconfig() | DT_CH1);
      }
      else { // Box unticked
         settrigconfig(gettrigconfig() & ~DT_CH1);
      }
      return true;
   }
   if (target == ch2) {
      if (ch2.getState()) { // Box ticked
         settrigconfig(gettrigconfig() | DT_CH2);
      }
      else { // Box unticked
         settrigconfig(gettrigconfig() & ~DT_CH2);
      }
      return true;
   }
```

```
        if (target == ch3) {
           if (ch3.getState()) { // Box ticked
              settrigconfig(gettrigconfig() | DT_CH3);
           }
           else { // Box unticked
              settrigconfig(gettrigconfig() & ~DT_CH3);
           }
           return true;
        }
        if (target == tc) {
           if (tc.getState()) { // Box ticked
              settrigconfig(gettrigconfig() | DT_TC);
           }
           else { // Box unticked
              settrigconfig(gettrigconfig() & ~DT_TC);
           }
           return true;
        }
        if (target == ti) {
           if (ti.getState()) { // Box ticked
              settrigconfig(gettrigconfig() | DT_TI);
           }
           else { // Box unticked
              settrigconfig(gettrigconfig() & ~DT_TI);
           }
           return true;
        }
        return false;
     }

     public Insets insets() {
        return new Insets(6,6,6,6);
     }

     public void paint(Graphics g) {
        Dimension d = size();
        Color bg = getBackground();

        g.setColor(bg);
        g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
        g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
     }

     // Called to update the contents of this panel
     public void Update() {
        int val = gettrigconfig();
        ch0.setState(((val & DT_CH0) == DT_CH0) ? true : false);
        ch1.setState(((val & DT_CH1) == DT_CH1) ? true : false);
        ch2.setState(((val & DT_CH2) == DT_CH2) ? true : false);
        ch3.setState(((val & DT_CH3) == DT_CH3) ? true : false);
        tc.setState(((val & DT_TC) == DT_TC) ? true : false);
        ti.setState(((val & DT_TI) == DT_TI) ? true : false);
     }
  }

  //
  // ButtonPanel
  //
  class ButtonPanel extends Panel {

     Button treset, breset, rupdate, start;
```

```
   Label tsts, bcntrl, regs;

   static {
      System.loadLibrary("DT2001J");
   }

   public ButtonPanel () {
      setLayout(new GridLayout(10,1,3,3));
      setBackground(Color.lightGray);

      tsts = new Label("Tests");
      add(tsts);

      start = new Button("Start");
      add(start);

      add(new Panel());

      regs = new Label("Registers");
      add(regs);

      rupdate = new Button("Update");
      add(rupdate);

      add(new Panel());

      bcntrl = new Label("Control");
      add(bcntrl);

      treset = new Button("Time Reset");
      add(treset);

      breset = new Button("Board Reset");
      add(breset);
   }

   public Insets insets() {
      return new Insets(6,6,6,6);
   }

   public void paint(Graphics g) {
      Dimension d = size();
      Color bg = getBackground();

      g.setColor(bg);
      g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
      g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
   }
}

//
// DatabasePanel
//
class DatabasePanel extends Panel {

   DBControl db;

   Label ttl;
   Label lblfrm;
   Label lblnum;
   TextField txtfrm;
```

```
TextField txtnum;
Button exec;

public DatabasePanel() {
   GridBagLayout gridbag = new GridBagLayout();
   GridBagConstraints c = new GridBagConstraints();
   setLayout(gridbag);
   setBackground(Color.lightGray);

   c.fill = GridBagConstraints.BOTH;

   ttl = new Label("Store Data");
   ttl.setAlignment(Label.CENTER);
   c.gridx = 0;
   c.gridy = 0;
   c.gridwidth = 2;
   c.gridheight = 1;
   c.weightx = 1.0;
   c.weighty = 0.0;
   gridbag.setConstraints(ttl, c);
   add(ttl);

   lblfrm = new Label("From (Hex)");
   c.gridx = 0;
   c.gridy = 2;
   c.gridwidth = 1;
   c.gridheight = 1;
   c.weightx = 0.0;
   c.weighty = 0.0;
   gridbag.setConstraints(lblfrm, c);
   add(lblfrm);

   lblnum = new Label("Number (Hex)");
   c.gridx = 0;
   c.gridy = 4;
   c.gridwidth = 1;
   c.gridheight = 1;
   c.weightx = 0.0;
   c.weighty = 0.0;
   gridbag.setConstraints(lblnum, c);
   add(lblnum);

   txtfrm = new TextField("0");
   c.gridx = 1;
   c.gridy = 2;
   c.gridwidth = 2;
   c.gridheight = 1;
   c.weightx = 0.0;
   c.weighty = 0.0;
   gridbag.setConstraints(txtfrm, c);
   add(txtfrm);

   txtnum = new TextField("1000");
   c.gridx = 1;
   c.gridy = 4;
   c.gridwidth = 2;
   c.gridheight = 1;
   c.weightx = 1.0;
   c.weighty = 0.0;
   gridbag.setConstraints(txtnum, c);
   add(txtnum);
```

```
        exec = new Button("Store");
        c.gridx = 0;
        c.gridy = 6;
        c.gridwidth = 2;
        c.gridheight = 1;
        c.weightx = 0.0;
        c.weighty = 1.0;
        gridbag.setConstraints(exec, c);
        add(exec);

        db = new DBControl("jdbc:odbc:dtdb");
    }

    public Insets insets() {
        return new Insets(6,6,6,6);
    }

    public void paint(Graphics g) {
        Dimension d = size();
        Color bg = getBackground();

        g.setColor(bg);
        g.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
        g.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
    }

    // Event handler
    public boolean action(Event e, Object arg) {

        Object target = e.target;

        if (target == exec) {
            Integer i;
            i = new Integer(0);
            int frm, num;

            try {
                frm = i.parseInt(txtfrm.getText(), 16);
                num = i.parseInt(txtnum.getText(), 16);
            }
            catch (NumberFormatException exception){
                frm = 0;
                num = 1024;
            }
            db.captureData(frm, num);
            return true;
        }
        return false;
    }

    public void finalize() {
        db.close();
    }
}

//
// Database Connectivity
//
class DBControl {
```

```
// [JNI] Native (DLL) method declarations
public native int mapRAM();
public native int unmapRAM();
public native int getRAM(int address, short[] buf,
                         int startaddr, int bytes);

String url;
Connection con;

public DBControl(String DBName) {
   url = DBName;

   try {
      // Load bridge driver
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

      // Find driver for URL
      Driver jdbcDriver = DriverManager.getDriver(url);

      // Define properties for connections (security)
      Properties propertyList = new Properties();
      propertyList.put("user", "dtuser");
      propertyList.put("password", "dtpassword");

      // Verbose status messages
      // *** DISABLE THIS FOR WORKING VERSIONS ***
      // DriverManager.setLogStream(System.out);
      // *****************************************

      // Establish connection
      con = DriverManager.getConnection(url, propertyList);

   }
   catch (SQLException ex) {
      System.out.println("Exception while trying to connect");
      while (ex != null) {
         System.out.println("SQLState: " + ex.getSQLState());
         System.out.println("Message: " + ex.getMessage());
         System.out.println("Vendor: " + ex.getErrorCode());
         System.out.println("URL: " + url);
         ex = ex.getNextException();
         System.out.println("");
      }
   }
   catch (java.lang.Exception ex) {
      ex.printStackTrace();
   }
}

// Store data from trace board in database
// (Works for just one board at the moment)
public void captureData(int from, int samples) {

   // Linear address returned from mapRAM function
   // Serves as a handle for calls to other memory functions
   int LA = mapRAM();

   // Pre-compiled SQL statement
   PreparedStatement pstmt;

   int i;
```

```
// Buffer for trace data
short[] buffer;

// Allocate space for buffer
buffer = new short[80000];

// There seems to be  problem with very large arrays
// so the data is written to the database in blocks.

// This code may need some work

// Read data in 0x20000 blocks until there are
// less than 0x20000 samples left.
while (samples > 20000) {
   getRAM(LA, buffer, from << 3, 100000);
   samples -= 20000;
   from += 100000;

   // Store the data in the database
   try {
      pstmt = con.prepareStatement(
             "INSERT INTO TraceBoardData (Board1Counter,
              Board1Data1, Board1Data2) VALUES (?, ?, ?)");
      for (i = 0; i < 80000; i += 4) {
         pstmt.setInt(1, i >> 2);
         pstmt.setInt(2, buffer[i]);
         pstmt.setInt(3, buffer[i + 1]);
         pstmt.executeUpdate();
      }
   }
   catch (SQLException ex) {
      System.out.println("Exception while executing query");
      while (ex != null) {
         System.out.println(
                           "SQLState: " + ex.getSQLState());
         System.out.println("Message: " + ex.getMessage());
         System.out.println("Vendor: " + ex.getErrorCode());
         ex = ex.getNextException();
         System.out.println("");
      }
   }
}

// Read what's left
getRAM(LA, buffer, from << 3, samples << 3);

// Store the data in the database
try {
   pstmt = con.prepareStatement(
          "INSERT INTO TraceBoardData (Board1Counter,
           Board1Data1, Board1Data2) VALUES (?, ?, ?)");
   for (i = 0; i < samples << 2; i += 4) {
      pstmt.setInt(1, i >> 2);
      pstmt.setInt(2, buffer[i]);
      pstmt.setInt(3, buffer[i + 1]);
      pstmt.executeUpdate();
   }
}
catch (SQLException ex) {
   System.out.println("Exception while executing query");
```

```
            while (ex != null) {
               System.out.println("SQLState: " + ex.getSQ LState());
               System.out.println("Message: " + ex.getMessage());
               System.out.println("Vendor: " + ex.getErrorCode());
               ex = ex.getNextException();
               System.out.println("");
            }
         }

         if (unmapRAM() == 0) {   // Java differentiates between
                                  // ints and booleans
            System.out.println("Memory unmap error");
         }
      }

      // Close the database connection
      public void close () {
         try {
            con.close();
         }
         catch (SQLException ex) {
            System.out.println(
                    "Exception while closing database  connection");
            while (ex != null) {
               System.out.println("SQLState: " + ex.getSQLState());
               System.out.println("Message: " + ex.getMessage());
               System.out.println("Vendor: " + ex.getErrorCode());
               ex = ex.getNextException();
               System.out.println("");
            }
         }
      }
   }
```

For further information on the Java Native Interface see :

1. “Using the JNI”, Beth Stearns,
   http://www.java.sun.com/docs/books/tutorial/native1.1/index.html

For further information on Java Database Connectivity see :

1. “Storing Java in a Relational Database”, John Dreystadt, Byte, June 1998.

2. “Teach Yourself Café in 21 days”, Eric Herrmann, Chapter 15,
   http://www.developer.com/reference/library/1575211572/ch15.htm

3. “JDBC Guide: Getting Started”, Sun Microsystems Inc.,
   http://www.java.sun.com/products/jdk/1.2/docs/guide/jdbc/getstart/introTOC.doc.html

Initialisation & EISA configuration

The EISA specification requires a read only identifier to be encoded at the slot specific I/O address plus 0x80-83, i.e. at 0xZ080-0xZ083. The encoding format assumes :

```
   0xZ080[7] = 0
0xZ080[6..2] = encoded_char1[4..0]
0xZ080[1..0] = encoded_char2[4..3]
0xZ081[7..5] = encoded_char2[2..0]
0xZ081[4..0] = encoded_char3[4..0]
0xZ082[7..4] = encoded_char4[3..0]
0xZ082[3..0] = encoded_char5[3..0]
0xZ083[7..4] = encoded_char6[3..0]
0xZ083[3..0] = encoded_char7[3..0]
```

where the ASCII characters of the identifier can be constructed as follows :

```
ASCII_char1[7..0] = 0x40 + encoded_char1[4..0]
ASCII_char2[7..0] = 0x40 + encoded_char2[4..0]
ASCII_char3[7..0] = 0x40 + encoded_char3[4..0]
ASCII_char4[7..0] = 0x30 + encoded_char4[3..0]
ASCII_char5[7..0] = 0x30 + encoded_char5[3..0]
ASCII_char6[7..0] = 0x30 + encoded_char6[3..0]
ASCII_char7[7..0] = 0x30 + encoded_char7[3..0]
```

Currently this register reads out a value of 0x01209212, representing :

```
'DTR200.1'
```

where the decimal point is implied within the encoding. This is the unique EISA ID for the board. This allows the board to be initialised on power up by the system board from initialisation values stored in the BIOS CMOS RAM. It also allows the driver software to scan the EISA slot specific I/O addresses to identify which slots contain trace boards, from which their I/O addresses may be derived.

The EISA initialization is configured by an EISA configuration file, and is stored into the BIOS CMOS RAM by an EISA configuration utility, such as AMI's CFG.EXE utility. The most useful values to initiliaize are the contents of the ADDRMAP register (to set up a separate VRAM base address for each board) and the MODE register (to set the PAUSE bit). The PAUSE bit could be set unconditionally, whereas each ADDRMAP register would have to be individually configured for each installed board by the EISA configuration utility. The following EISA configuration file, !DTR2001.CFG, is more general :

```
BOARD
  ID="DTR2001"
  NAME="DT200.1 Deep Trace Board"
  MFR="Trinity College Dublin"
  CATEGORY="OTH"
  SLOT=EISA
  LENGTH=330
  AMPERAGE=1000
  SKIRT=NO
  READID=YES
  IOCHECK=INVALID
  DISABLE=UNSUPPORTED
  COMMENTS="DT200.1 Configuration File Ver. 1.0"
```

```
IOPORT(1)=0z000h       ;RESET address
  SIZE = DWORD
  INITVAL = 00000000000000000000000000000000
IOPORT(2)=0z004h       ;ADDRMAP address
  SIZE = DWORD
  INITVAL = XXXXXXXX111111111111111111111111
IOPORT(3)=0z008h       ;MODE address
  SIZE = DWORD
  INITVAL = 111111111111111111111111XXXXXXXX
IOPORT(4)=0z00Ch       ;TRIGCONF address
  SIZE = DWORD
  INITVAL = 11111111111111111111111111XXXXXX
IOPORT(5)=0z010h       ;TIMERESET address
  SIZE = DWORD
  INITVAL = 00000000000000000000000000000000
IOPORT(6)=0z014h       ;STOPCOUNT address
  SIZE = DWORD
  INITVAL = 11111111XXXXXXXXX111111111111111
IOPORT(7)=0z018h       ;HEADPTR address
  SIZE = DWORD
  INITVAL = 11111111XXXXXXXXX111111111111111
IOPORT(8)=0z01Ch       ;OUTPUTENA address
  SIZE = DWORD
  INITVAL = 11111111111111111111111111XXXXXX
IOPORT(9)=0z020h       ;STATUS address
  SIZE = DWORD
  INITVAL = 1111111111111111R111111111111111
IOPORT(10)=0z080h       ;EISAID address
  SIZE = DWORD
  INITVAL = RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR


FUNCTION = "Deep Trace Configuration"
        TYPE = "DT2001"
  HELP = "This selection sets the Deep Trace board
          post-RESET configuration."

  SUBFUNCTION = "Select VRAM Base Address"
  HELP = "This option selects the VRAM base address."
  CHOICE = "Base Address = 00"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00000000B
  CHOICE = "Base Address = 01"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00000001B
  CHOICE = "Base Address = 02"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00000010B
  CHOICE = "Base Address = 03"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00000011B
  CHOICE = "Base Address = 04"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00000100B
  CHOICE = "Base Address = 05"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00000101B
  CHOICE = "Base Address = 06"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00000110B
  CHOICE = "Base Address = 07"
```

```
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00000111B
  CHOICE = "Base Address = 08"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00001000B
  CHOICE = "Base Address = 09"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00001001B
  CHOICE = "Base Address = 0A"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00001010B
  CHOICE = "Base Address = 0B"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00001011B
  CHOICE = "Base Address = 0C"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00001100B
  CHOICE = "Base Address = 0D"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00001101B
  CHOICE = "Base Address = 0E"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00001110B
  CHOICE = "Base Address = 0F"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00001111B
  CHOICE = "Base Address = 10"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00010000B
  CHOICE = "Base Address = 11"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00010001B
  CHOICE = "Base Address = 12"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00010010B
  CHOICE = "Base Address = 13"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00010011B
  CHOICE = "Base Address = 14"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00010100B
  CHOICE = "Base Address = 15"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00010101B
  CHOICE = "Base Address = 16"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00010110B
  CHOICE = "Base Address = 17"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00010111B
  CHOICE = "Base Address = 18"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00011000B
  CHOICE = "Base Address = 19"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00011001B
  CHOICE = "Base Address = 1A"
    FREE
    INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00011010B
  CHOICE = "Base Address = 1B"
    FREE
```

```
  INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00011011B
CHOICE = "Base Address = 1C"
  FREE
  INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00011100B
CHOICE = "Base Address = 1D"
  FREE
  INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00011101B
CHOICE = "Base Address = 1E"
  FREE
  INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00011110B
CHOICE = "Base Address = 1F"
  FREE
  INIT=IOPORT(2) LOC(31 30 29 28 27 26 25 24) 00011111B

SUBFUNCTION = "Choose between VRAM or SRAM Access"
HELP = "This option chooses between SRAM access
        rather than VRAM access."
CHOICE = "VRAM Access"
  FREE
  INIT=IOPORT(3) LOC(0) 0B
CHOICE = "SRAM Access"
  FREE
  INIT=IOPORT(3) LOC(0) 1B

SUBFUNCTION = "Pause Acquisition of Data"
HELP = "This option allows data acquisition to be halted."
CHOICE = "Pause Data Acquisition"
  FREE
  INIT=IOPORT(3) LOC(1) 1B
CHOICE = "Continue Data Acquisition"
  FREE
  INIT=IOPORT(3) LOC(1) 0B

SUBFUNCTION = "Acquire/Generate Data"
HELP = "This option choose between data acquisition
        and generation. For acquisition the MODE
        register RECORD bit is set, for generation
        it is reset."
CHOICE = "Acquire (Record) Data"
  FREE
  INIT=IOPORT(3) LOC(2) 1B
CHOICE = "Generate Data"
  FREE
  INIT=IOPORT(3) LOC(2) 0B

SUBFUNCTION = "Enable/Disable VRAM Buffer Wrap"
HELP = "This option enables/disables VRAM buffer wrap."
CHOICE = "Enable Buffer Wrap"
  FREE
  INIT=IOPORT(3) LOC(3) 1B
CHOICE = "Disable Buffer Wrap"
  DISABLE = YES
  FREE
  INIT=IOPORT(3) LOC(3) 0B

SUBFUNCTION = "Timestamp Clock Divisor"
HELP = "This option allows choice of the timestamp
        clock divisor."
CHOICE = "Timestamp Divisor = 1"
  FREE
  INIT=IOPORT(3) LOC(5 4) 00B
```

```
CHOICE = "Timestamp Divisor = 2"
  FREE
  INIT=IOPORT(3) LOC(5 4) 01B
CHOICE = "Timestamp Divisor = 4"
  FREE
  INIT=IOPORT(3) LOC(5 4) 10B
CHOICE = "Timestamp Divisor = 8"
  FREE
  INIT=IOPORT(3) LOC(5 4) 11B

SUBFUNCTION = "DMA Channel 7 Enable/Disable"
HELP = "This option enables/disables DMA channel 7
        to be used by DT200.1 drivers."
CHOICE = "DMA 7 Enable"
  LINK
  DMA = 7
  SHARE = NO
  INIT=IOPORT(3) LOC(6) 1B
CHOICE = "DMA 7 Disable"
  DISABLE = YES
  FREE
  INIT=IOPORT(3) LOC(6) 0B

SUBFUNCTION = "IRQ 12 Enable/Disable"
HELP = "This option enables/disables IRQ channel 12
        to be used by DT200.1 drivers."
CHOICE = "IRQ 12 Enable"
  FREE
  INIT=IOPORT(3) LOC(7) 1B
  LINK
  IRQ = 12
  SHARE = NO
  TRIGGER = EDGE
CHOICE = "IRQ 12 Disable"
  DISABLE = YES
  FREE
  INIT=IOPORT(3) LOC(7) 0B

SUBFUNCTION = "Select Channel 0 Trigger/Trace"
HELP = "This option choose between trigger and trace
        definition for channel 0."
CHOICE = "Channel 0 Trigger"
  FREE
  INIT=IOPORT(4) LOC(0) 1B
CHOICE = "Channel 0 Trace"
  FREE
  INIT=IOPORT(4) LOC(0) 0B

SUBFUNCTION = "Select Channel 1 Trigger/Trace"
HELP = "This option choose between trigger and trace
        definition for channel 1."
CHOICE = "Channel 1 Trigger"
  FREE
  INIT=IOPORT(4) LOC(1) 1B
CHOICE = "Channel 1 Trace"
  FREE
  INIT=IOPORT(4) LOC(1) 0B

SUBFUNCTION = "Select Channel 2 Trigger/Trace"
HELP = "This option choose between trigger and trace
        definition for channel 2."
```

```
CHOICE = "Channel 2 Trigger"
  FREE
  INIT=IOPORT(4) LOC(2) 1B
CHOICE = "Channel 2 Trace"
  FREE
  INIT=IOPORT(4) LOC(2) 0B

SUBFUNCTION = "Select Channel 3 Trigger/Trace"
HELP = "This option choose between trigger and trace
        definition for channel 3."
CHOICE = "Channel 3 Trigger"
  FREE
  INIT=IOPORT(4) LOC(3) 1B
CHOICE = "Channel 3 Trace"
  FREE
  INIT=IOPORT(4) LOC(3) 0B

SUBFUNCTION = "Continuously Acquire Data"
HELP = "This option chooses between filtered and unfiltered
        tracing of data. Filtered tracing means data is
        acquired only when it matches a trace pattern.
        Unfiltered tracing means data is continuously
        acquired (the TRACE CONTINUOUSLY bit is set)."
CHOICE = "Trace Continuously"
  FREE
  INIT=IOPORT(4) LOC(4) 1B
CHOICE = "Trace on Pattern Match"
  FREE
  INIT=IOPORT(4) LOC(4) 0B

SUBFUNCTION = "Trigger Immediately"
HELP = "This option chooses between immediate triggering
        of the Stop Counter or not. Immediate triggering
        means the Stop Counter starts decrementing even
        if the data does not match a trigger pattern."
CHOICE = "Trigger Immediately"
  FREE
  INIT=IOPORT(4) LOC(5) 1B
CHOICE = "Trigger on Pattern Match"
  DISABLE = YES
  FREE
  INIT=IOPORT(4) LOC(5) 0B

SUBFUNCTION = "Set the Stop Count"
HELP = "This option sets the sample count before data
        acquistion ends (after triggering). A value of
        000000000 means that sampling will stop immediately
        after triggering, so that the trigger item is
        at the end of the trace data. A value of 111111111
        means that sampling will stop such that the
        trigger item is at the beginning of the trace data."
CHOICE = "Stop Count = 000"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000000000B
CHOICE = "Stop Count = 001"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000000001B
CHOICE = "Stop Count = 002"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000000010B
CHOICE = "Stop Count = 003"
```

```
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000000011B
CHOICE = "Stop Count = 004"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000000100B
CHOICE = "Stop Count = 005"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000000101B
CHOICE = "Stop Count = 006"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000000110B
CHOICE = "Stop Count = 007"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000000111B
CHOICE = "Stop Count = 008"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000001000B
CHOICE = "Stop Count = 009"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000001001B
CHOICE = "Stop Count = 00A"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000001010B
CHOICE = "Stop Count = 00B"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000001011B
CHOICE = "Stop Count = 00C"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000001100B
CHOICE = "Stop Count = 00D"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000001101B
CHOICE = "Stop Count = 00E"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000001110B
CHOICE = "Stop Count = 00F"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000001111B
CHOICE = "Stop Count = 010"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000011111B
CHOICE = "Stop Count = 020"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000100000B
CHOICE = "Stop Count = 030"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 000110000B
CHOICE = "Stop Count = 040"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 001000000B
CHOICE = "Stop Count = 050"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 001010000B
CHOICE = "Stop Count = 060"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 001100000B
CHOICE = "Stop Count = 070"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 001110000B
CHOICE = "Stop Count = 080"
  FREE
```

```
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 010000000B
CHOICE = "Stop Count = 090"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 010010000B
CHOICE = "Stop Count = 0A0"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 010100000B
CHOICE = "Stop Count = 0B0"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 010110000B
CHOICE = "Stop Count = 0C0"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 011000000B
CHOICE = "Stop Count = 0D0"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 011010000B
CHOICE = "Stop Count = 0E0"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 011100000B
CHOICE = "Stop Count = 0F0"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 011110000B
CHOICE = "Stop Count = 100"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 100000000B
CHOICE = "Stop Count = 140"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 101000000B
CHOICE = "Stop Count = 180"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 110000000B
CHOICE = "Stop Count = 1C0"
  FREE
  INIT=IOPORT(6) LOC(23 22 21 20 19 18 17 16 15) 111000000B

SUBFUNCTION = "Set the Head Pointer"
HELP = "This option sets the VRAM address from which data
        data storage begins."
CHOICE = "Head Pointer = 000"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000000000B
CHOICE = "Head Pointer = 001"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000000001B
CHOICE = "Head Pointer = 002"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000000010B
CHOICE = "Head Pointer = 003"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000000011B
CHOICE = "Head Pointer = 004"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000000100B
CHOICE = "Head Pointer = 005"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000000101B
CHOICE = "Head Pointer = 006"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000000110B
CHOICE = "Head Pointer = 007"
  FREE
```

```
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000000111B
CHOICE = "Head Pointer = 008"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000001000B
CHOICE = "Head Pointer = 009"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000001001B
CHOICE = "Head Pointer = 00A"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000001010B
CHOICE = "Head Pointer = 00B"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000001011B
CHOICE = "Head Pointer = 00C"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000001100B
CHOICE = "Head Pointer = 00D"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000001101B
CHOICE = "Head Pointer = 00E"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000001110B
CHOICE = "Head Pointer = 00F"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000001111B
CHOICE = "Head Pointer = 010"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000010000B
CHOICE = "Head Pointer = 020"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000100000B
CHOICE = "Head Pointer = 030"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 000110000B
CHOICE = "Head Pointer = 040"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 001000000B
CHOICE = "Head Pointer = 050"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 001010000B
CHOICE = "Head Pointer = 060"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 001100000B
CHOICE = "Head Pointer = 070"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 001110000B
CHOICE = "Head Pointer = 080"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 010000000B
CHOICE = "Head Pointer = 090"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 010010000B
CHOICE = "Head Pointer = 0A0"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 010100000B
CHOICE = "Head Pointer = 0B0"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 010110000B
CHOICE = "Head Pointer = 0C0"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 011000000B
```

```
CHOICE = "Head Pointer = 0D0"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 011010000B
CHOICE = "Head Pointer = 0E0"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 011100000B
CHOICE = "Head Pointer = 0F0"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 011110000B
CHOICE = "Head Pointer = 100"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 100000000B
CHOICE = "Head Pointer = 140"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 101000000B
CHOICE = "Head Pointer = 180"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 110000000B
CHOICE = "Head Pointer = 1C0"
  FREE
  INIT=IOPORT(7) LOC(23 22 21 20 19 18 17 16 15) 111000000B

SUBFUNCTION = "Enable Serial Buffer 0"
HELP = "This option choose between data acquisition from
        the channel 0 sample data input or timestamp
        counter."
CHOICE = "Sample"
  FREE
  INIT=IOPORT(8) LOC(0) 1B
CHOICE = "Timestamp"
  FREE
  INIT=IOPORT(8) LOC(0) 0B

SUBFUNCTION = "Enable Serial Buffer 1"
HELP = "This option choose between data acquisition from
        the channel 1 sample data input or timestamp
        counter."
CHOICE = "Sample"
  FREE
  INIT=IOPORT(8) LOC(1) 1B
CHOICE = "Timestamp"
  FREE
  INIT=IOPORT(8) LOC(1) 0B

SUBFUNCTION = "Enable Serial Buffer 2"
HELP = "This option choose between data acquisition from
        the channel 2 sample data input or timestamp
        counter."
CHOICE = "Sample"
  FREE
  INIT=IOPORT(8) LOC(2) 1B
CHOICE = "Timestamp"
  FREE
  INIT=IOPORT(8) LOC(2) 0B

SUBFUNCTION = "Enable Serial Buffer 3"
HELP = "This option choose between data acquisition from
        the channel 3 sample data input or timestamp
        counter."
CHOICE = "Sample"
  FREE
```

```
  INIT=IOPORT(8) LOC(3) 1B
CHOICE = "Timestamp"
  FREE
  INIT=IOPORT(8) LOC(3) 0B

SUBFUNCTION = "Enable Serial Buffer 4"
HELP = "This option choose between data acquisition from
        the channel 4 sample data input or timestamp
        counter."
CHOICE = "Sample"
  FREE
  INIT=IOPORT(8) LOC(4) 1B
CHOICE = "Timestamp"
  FREE
  INIT=IOPORT(8) LOC(4) 0B

SUBFUNCTION = "Enable Serial Buffer 5"
HELP = "This option choose between data acquisition from
        the channel 5 sample data input or timestamp
        counter."
CHOICE = "Sample"
  FREE
  INIT=IOPORT(8) LOC(5) 1B
CHOICE = "Timestamp"
  FREE
  INIT=IOPORT(8) LOC(5) 0B
```

Functional Overview

Conceptually the system functions as a very deep FIFO. Data is sampled via the 48 bit serial interface on the positive edge of a clock and stored in a 12MByte VRAM buffer. The buffer wraps around and can be read out via the EISA interface without stopping the sample clock. The layout is shown below.

There are nine major pals: EISAIF, STATM, DECODE, CNTCMP, CLOCK, TRIG, TIME0, TIME1 and SMUX, all MACH210-10 devices. There are four minor pals: EISADR, COLS, CLKDR and IOTR, the first two of which are AmPAL22CEV10H/4 devices, and the last two of which are Lattice GAL16V8-7 devices.

EISAIF responds to the EISA interface memory and I/O cycles, generating two major select signals, /EISAMEMCS and /EISAIOCS. It also implements the ADDRMAP register and the top 8bits of the EISAID register, i.e. EISAID[31..24].

STATM responds is essentially a large state machine that responds to /EISAMEMCS and /EISAIOCS by stepping through a sequence of states that define the logic levels for all the major memory and register control signals.

DECODE decodes the low order address signals into register select signals /RESET (RESET register), /MAPCS (ADDRMAP register), /TRCNFCS (TRIGCONFIG register), /TIMERESET (TIMERESET register), /TRREGCS (STOPCOUNT register), /HREGCS (HEADPTR register) and /HIDCS (EISAID register). /TRREGCS and /HIDCS are both activated to select the STATUS register. It also implements the MODE and OUTPUTENA registers. On a system reset, or a write to the RESET register, it activates its /RESET output to initialize the board.

CNTCMP contains the stopcount and head pointer counters, and also implements the STATUS register and EISAID[23..15].

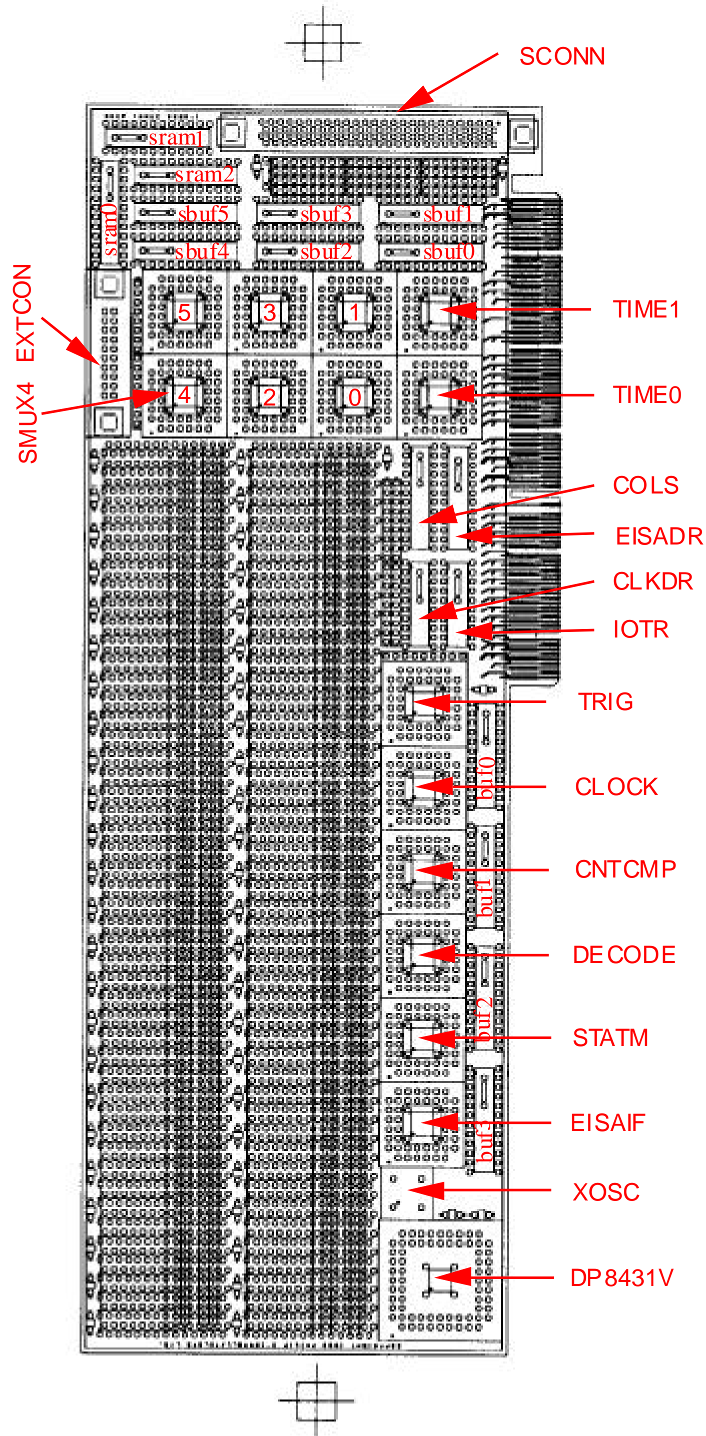
CLOCK accepts the LOCALTRACE and LOCALTRIGGER signals from the TRIG pal, combines these with EXTRA2-0 and EXTRIG2-0, and generates appropriate clocks for the CNTCMP pal. It also contains 9bit counters that track the progress of the VRAM sequential access memory (SAM) pointer, and issue a request to STATM for a VRAM transfer cycle when they roll over from `111111111` to `000000000`.

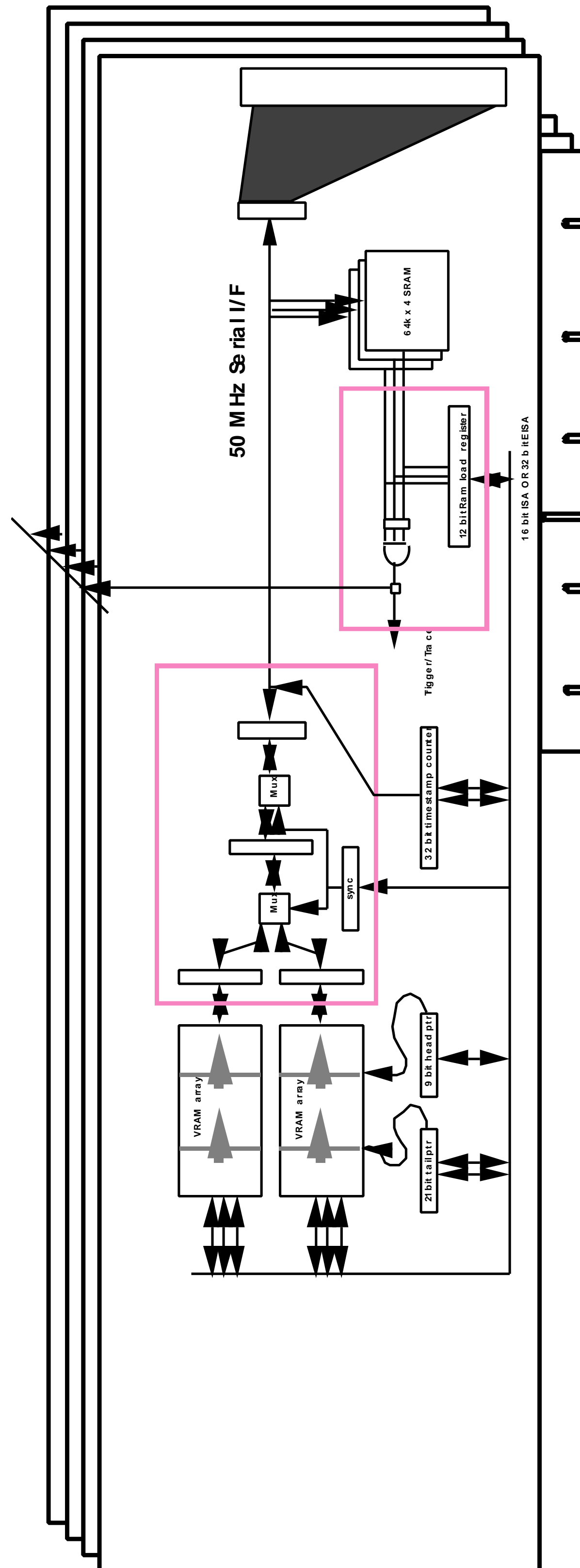
TRIG takes in the 12bit SRAM data and generates LOCALTRACE and LOCALTRIGGER signals as appropriate. It also implements the TRIGCONFIG register and EISAID[14..0], and provides the data paths for read and write accesses to the SRAMs.

TIME0 and TIME1 implement the low and high halves of the timestamp counter.

SMUX acts as a general multiplexer and demultiplexer of signals between the internal serial bus and the VRAM serial data. It also provides the address paths for read and write accesses to the SRAMs. There are six of these pals, SMUX0-5, handling byte 0-5 of the 48bit serial data.

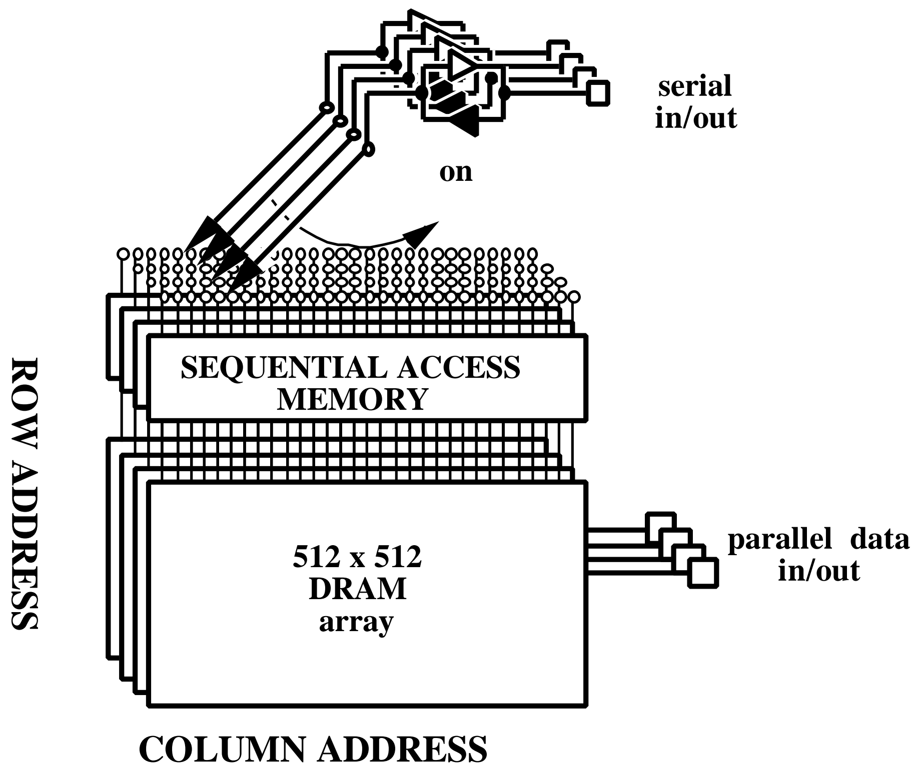
An ordinary DRAM controller, the DP8431V, is used.


SCONN
sram1
sram2
sram0
sbuf5
sbuf3
sbuf1
sbuf4
sbuf2
sbuf0
EXTCON
SMUX4
5
3
1
4
2
0
TIME1
TIME0
COLS
EISADR
CLKDR
IOTR
TRIG
CLOCK
buf0
CNTCMP
buf1
DECODE
buf2
STATM
EISAIF
buf3
XOSC
DP8431V


50 MHz Serial I/F
64k x 4 SRAM
12 bit Ram load register
16 bit ISA OR 32 bit EISA
32 bit timestamp counter
Mux
Mux
sync
9 bit head ptr
21 bit tail ptr
VRAM array
VRAM array

VRAM Logic

The video RAMs used in this project are TC524256AZ-10 devices made by Toshiba. These are 1 Mbit dual port devices, arranged as 512 rows by 512 columns by 4 bits of dynamic random access memory (DRAM), plus a 512 x 4bit sequential access memory (SAM). The SAM holds the equivalent of one row of the larger DRAM. As well as the usual DRAM read and write operations, the device supports high speed serial read and write via the SAM port. It is this serial read/write capability that is used here to sample (or output) data. The minimum serial cycle time is 30ns giving a theoretical maximum sample frequency of 33Mhz. In practice, however, a maximum serial cycle rate of 12.5Mhz is expected, as a result of striping the sampling across four banks of VRAMs during sampling.



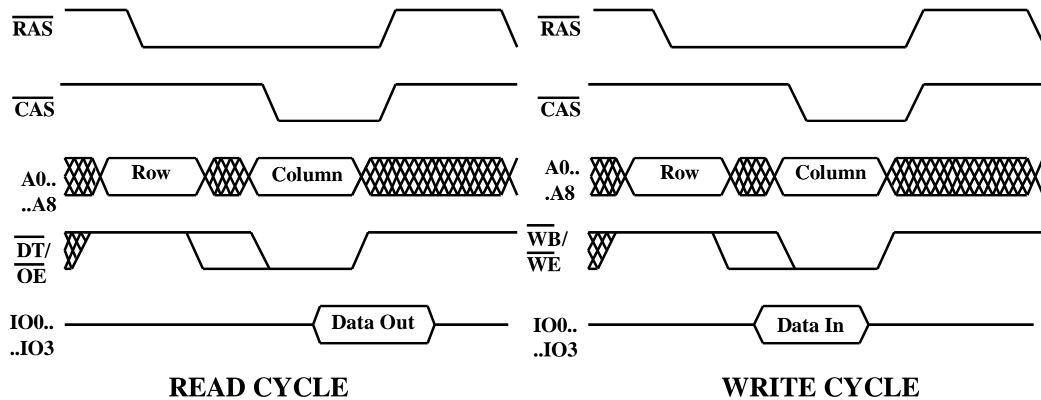
Video RAM (VRAM)

The larger DRAM array is dynamic, i.e. data decays to zero over a few milliseconds, so it must be refreshed periodically. During a refresh cycle only /RAS[7] is asserted. There is an on chip refresh counter that can be used for /CAS-before-/RAS refresh cycles (where the negative transition of /CAS occurs before the negative transition of /RAS). However, the DP8431V DRAM controller prohibits such cycles, so /RAS-only refreshing is performed, using the refresh counter on the DP8431V itself, which at least staggers this across four banks to reduce the peak refresh currents. The SAM is static, so refreshing is not required. Read and write cycles are similar to those for other DRAMs except for some timing requirements for the output enable and write enable signals.

The output enable and write enable pins are shared with two other functions, write-per-bit and data-transfer. Because of this, /DTOE (data-transfer-and-output-enable) must be high at the negative transition of /RAS during a read cycle, or a false transfer will be initiated. Similarly,

7 Active low signals are represented by the '/' convention e.g. NOT RAS = /RAS.

/WBWE (write-per-bit-and-write-enable) must be high at the negative transition of /RAS during a write cycle, or a write-per-bit cycle will be executed. This means that during normal access cycles, the negative transition of /DTOE or /WBWE must occur between the negative transitions of /RAS and /CAS.



Standard access cycles

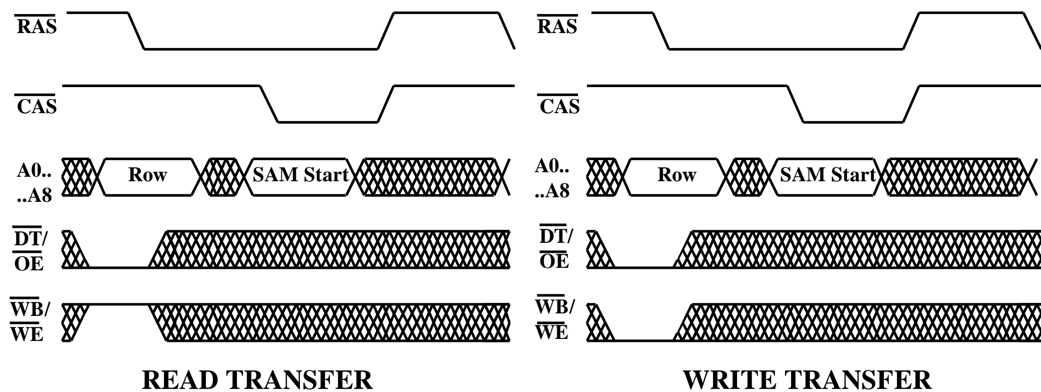
Transfer cycles are initiated by holding /DTOE low at the negative transition of /RAS. There are three types of transfer cycle :

(a) During a *read transfer*, one row of data is copied from RAM to SAM. To start a read transfer, /DTOE must be low, /SE (serial enable input) must be low, and /WBWE must be high, all at the negative transition of /RAS.

(b) During a *write transfer*, one row of data is copied from SAM to RAM. To start a write transfer, /DTOE, /SE and /WBWE must be low at the negative transition of /RAS.

(c) *Pseudo-write transfer* cycles are used to switch the serial data lines from output to input. Thus it is necessary to first perform a pseudo-write transfer cycle before data can be input via the serial port. The conditions for executing a pseudo-write transfer cycle are the same as for a write transfer except that /SE (serial enable) must be high at the negative transition of /RAS.

Transfer cycle timing is illustrated below. Note that /SE (not shown) is held low :



Transfer Cycles

During transfer cycles, the row address gives the row number to be transferred. The column address gives the new value of the SAM pointer upon completion of the tranfer cycle. Normally this would be set to zero each time.

The SAM port operates synchronously with SC, the serial clock input. At every positive transition of SC, the next word of data is input or output via SIO[3..0]. After 512 such cycles the SAM wraps around and word 0 is input or output again. In normal use the DRAM port and the SAM port operate totally independantly, except during transfer operations when data cannot be read from or written to either port. Refresh cycles are also prohibited during transfers.

Some initialisation operations are necessary before the VRAMs can be used :

(a) Before data can be read in via the SAM, a pseudo write transfer operation must be executed. This need only be done once if SIO[3..0] stay in input mode.

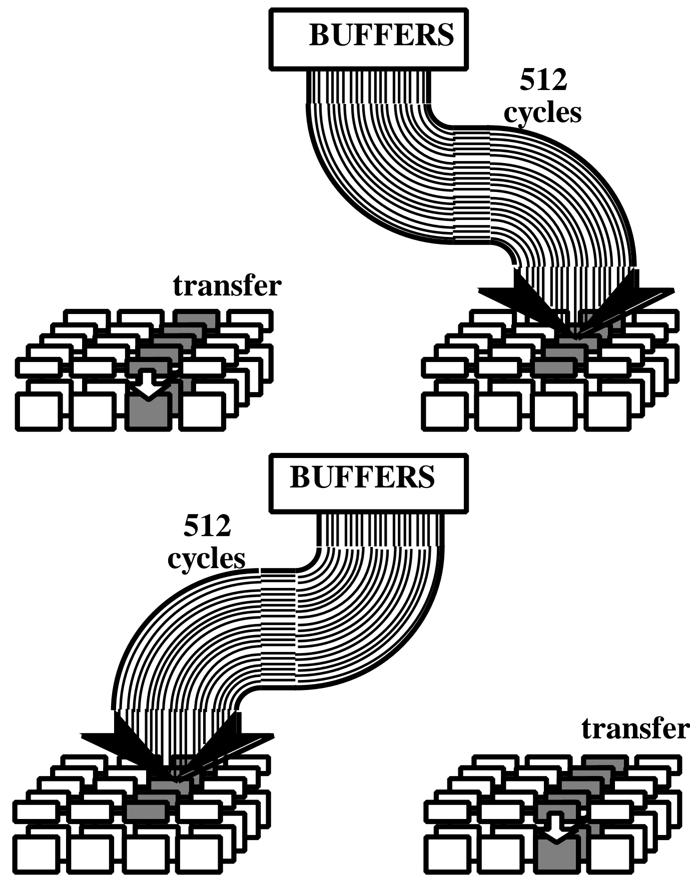
(b) On power-ON, each VRAM must undergo at least 8 transfer cycles of any type to initialise the transfer gate.

(c) At least 8 serial clock cycles must also be performed on power- ON to initialise the SAM.

These are programmed into the INIT equation in the STATM pal.

Bank Multiplexing

Because the sequential part of the Video RAMs only holds 512 x 4bit words, a transfer cycle must be executed for every 512 samples. A transfer cycle takes the same amount of time as an ordinary read or write cycle, so to avoid loss of data the VRAM is arranged in banks. For simplicity, let us describe an arrangement when sampling data for two banks, bearing in mind that the board is actually quadruply multiplexed, and can output data as well as capture input samples. While one bank is sampling data, the other bank can transfer the samples accepted during the previous 512 cycles of the serial clock. Every 512 cycles the serial data path is switched back and forth from one bank to the other, with each new transfer into successive rows of DRAM.



Data Acquisition Scheme with two VRAM banks

Multiplexing the serial lines is achieved by switching the serial clock between the two banks every 512 cycles of the sample counter. The counter itself is a 9bit synchronous counter contained in the CLOCK pal, that tracks the progress of the SAM pointer by counting the cycles left before a transfer (and a bank switch) is needed. The switch from one bank to the other must be accomplished as fast as possible if there is to be no loss of data when sampling at frequencies approaching 50Mhz.

The CLOCK pal also contains the serial clock switcher. This multiplexes the serial clock signal to the banks of VRAM so that only one bank can sample at a time. Hence there is one serial clock output for each bank. The SAM port samples every positive transition of its SC, so the switching is done on the negative transition between cycles 511 and 0. In this way, loss of data through late switching is avoided.

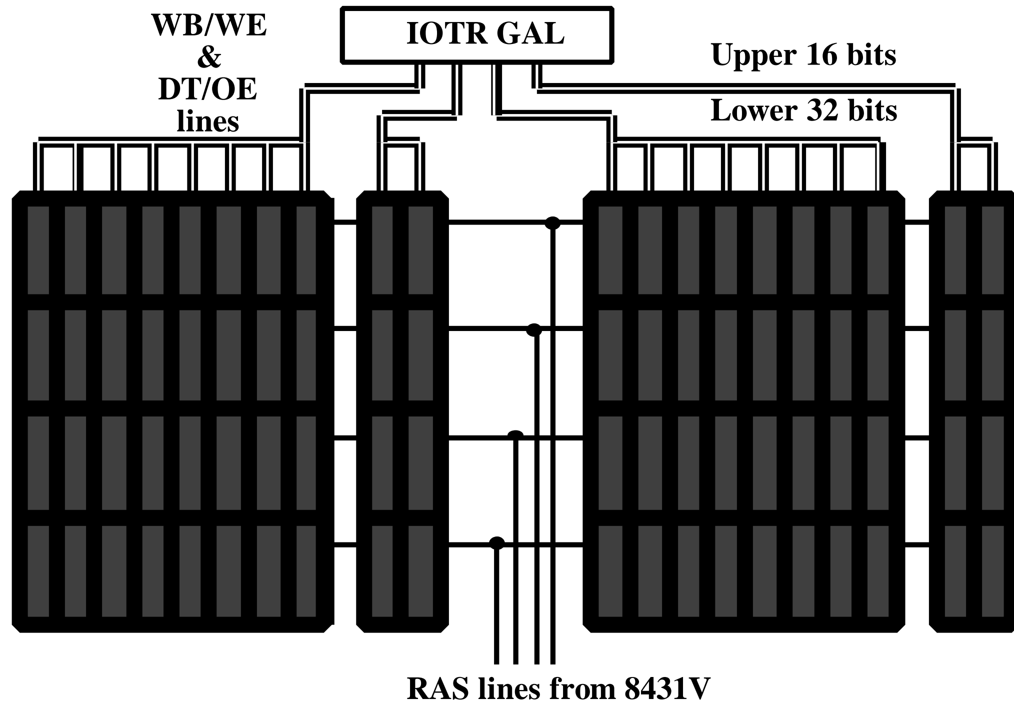
Two pals, CNTCMP and COLS, are situated between the EISA bus address and the DRAM controller row and column inputs. During normal read or write cycles the EISA bus address is routed through to the DRAM controller address inputs. In a transfer cycle, however, the transfer must be performed on the row given by the sample counter (in the CNTCMP pal). For this reason, during a transfer cycle, these bits of the counter are routed through the CNTCMP pal to the DRAM controller address inputs, and zeroes are routed through the COLS pal to reset the SAM pointer to zero.

All counter logic is qualified by two signals, /RESET and /PAUSE. /RESET asynchronously resets the counter to zero and /PAUSE temporarily halts the count and disables the serial clock SC to both banks.

VRAM Address Multiplexing

Addressing of the VRAMs is controlled by the IOTR (Input/Output & TRansfer) pal. Among the inputs to this are LA14 to select the bank during an access, TRFBANK to select the bank during a transfer and LA2 to select the upper or lower half of a sample. Inputs B0 and B1 to the DRAM controller select which of four /RAS signals is to be activated; these divide the VRAM array into four blocks of 24 VRAMs.

Each block of 24 is divided into two banks of 12 (twelve of the 4bit VRAM serial I/O ports corresponds to the 48bit sample width), which is further divided into two halves (4 upper and 8 lower). /RAS and /CAS are activated for both halves (upper 16 and lower 32) and both banks. The data from the VRAMs is multiplexed using their output-enable (/DTOE) and write-enable (/WBWE) inputs. There are eight outputs from the IOTR pal devoted to controlling these signals to the array (two signals {/DTOE and /WBWE} times two banks times two halves equals eight).



During a transfer operation both upper and lower signals are activated, so two cycle definition signals, ACC[1..0], are also used to control this, as well as the current bank (H14 from the CLOCK pal).

To summarise then, the outputs of the IOTR pal are as follows :

(a) /DTOEB0UPR16 (/DTOE Bank0 upper16), /DTOEB1UPR16, /DTOEB0LWR32 and /DTOEB1LWR32 to the VRAMs.

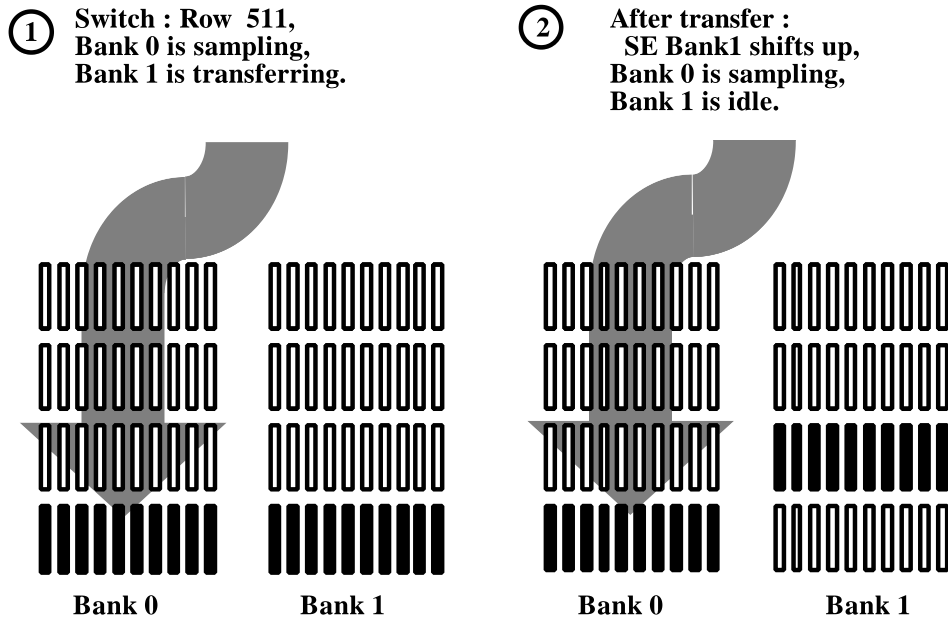
(b) /WBWEB0UPR16 (/WBWE Bank0 upper16), /WBWEB1UPR16, /WBWEB0LWR32 and /WBWEB1LWR32 to the VRAMs.

VRAM Serial Enables

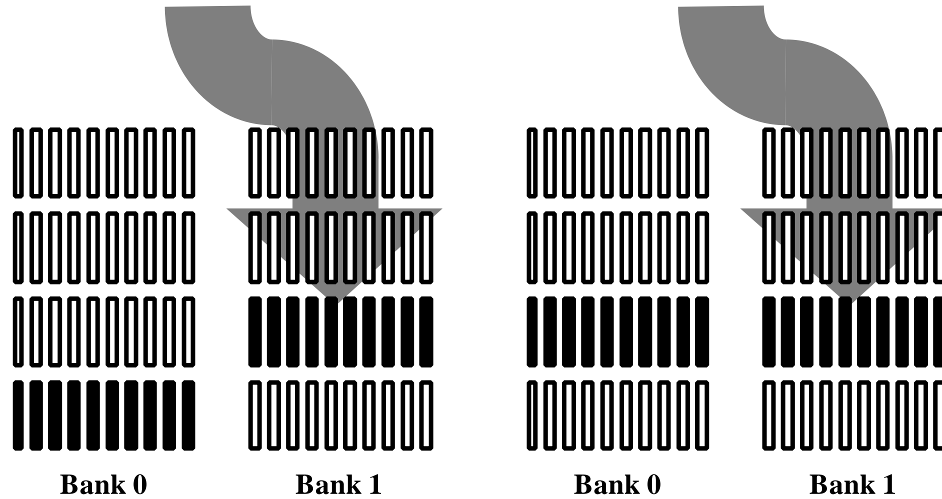
During any memory cycle, one of four /RAS lines is always strobed. This means that during every cycle at least 24 VRAMs will undergo a memory cycle, regardless of the intended cycle type or the target selection of VRAMs. The type of cycle is determined by three other signals, /DTOE, /WBWE and /SE. The first two signals have previously been described. The third, the serial enable, /SE, is controlled by the CLOCK pal.

CLOCK has eight outputs that connect to the /SE input of eight groups of twelve VRAMS. At any given instant only two outputs will be active. The first corresponds to the group of twelve that is actively sampling, and the second corresponds to the group of twelve that was sampling during the previous 512 cycles (which just executed, or is in the process of executing, a transfer cycle).

After 512 x 2 transfer cycles (each VRAM has 512 rows and there are 2 banks), new transfer cycles must be performed on a new set of VRAMs. At this point CLOCK should deactivate those /SE outputs it had activated above, so that each transfer cycle is executed by the correct block of twelve VRAMs. The sequence that must be followed is illustrated below.



Black filled boxes represent VRAMs whose /SE is active. The left diagram illustrates the situation immediately following a bank switch on row 511 (i.e. the last row in a block of VRAMs). On completion of the transfer cycle, the /SE for the next block upwards on bank 1 is activated, as shown in the right diagram, so that it is ready to accept data upon the next bank switch.



After the next 512 serial clock cycles the row count rolls over to zero and the serial data path is switched to bank 1, as in the left diagram above. Bank 1 is now accepting new samples while a transfer cycle is copying the SAM data on bank 0. After this transfer cycle, the /SE for the next block upwards on bank 0 is activated, as shown in the right diagram above, so that it is ready to accept data upon the next bank switch, and leaving it "level" with the /SE on bank 1.

The SMUX pals isolate each VRAM bank's serial data from the 48bit internal serial bus, routing through the appropriate data paths in synchronism with the serial clock. The /SE signal also acts as an output enable for the SAM ports when the SAM is switched to output data, in which case the SMUX pals route the data in the reverse direction.

Write-per-bit Write Mask

If /WBWE is low at the negative transition of /RAS during both ordinary write or write transfer cycles, then the data on the DRAM ports is interpreted as a write mask. The write is then only performed for those bits for which the corresponding mask bit is a logical 1. The write mask is valid for only one cycle.

For ordinary write cycles, the STATM pal forces /WBWE high (by deactivating /DTOEWBWE) at the negative transition of /RAS to disable the write mask. However, for write transfer cycles /WBWE must be low at the negative transition of /RAS to indicate the transfer direction, so the write mask will always be enabled for these cycles; in this case the data on the DRAM ports is pulled to a logical 1 by a set of pullup resistors. For convenience, the assignment of signals to pullup resistors is shown below :

	pullup0	pullup1	pullup2	pullup3	pullup4
r1	HD0	HD8	HD16	HD24	HSAB
r2	HD1	HD9	HD17	HD25	HSBA
r3	HD2	HD10	HD18	HD26	/ECAS0
r4	HD3	HD11	HD19	HD27	/TIMERESTOC
r5	HD4	HD12	HD20	HD28	SBUFSAB
r6	HD5	HD13	HD21	HD29	SBUFSBA

r7	HD6	HD14	HD22	HD30	/PAUSEOC
r8	HD7	HD15	HD23	HD31	/SYNC

State Machine

Amongst other things, the STATM pal performs the following functions :

(a) Arbitration between memory requests according to priority.

(b) Sequencing of signals to the VRAM array and DRAM controller.

(c) Generation of /HSTB , which is used to generate EXRDY for the EISA interface.

The state machine has four basic states, corresponding to the four types of memory cycle that can be performed on the array : IDLE_IO, TRANSFER, ACCESS and REFRESH. It is implemented as a 4bit Gray-coded sequencer, where each of the sixteen possible states is decoded to allow output signals to be activated at different stages during the cycle. Each cycle is thus a maximum of sixteen clock periods long (16 x 40ns = 640ns). The zero state is decoded as SWITCH; when in this state the state machine switches state according to the priority REFRESH > TRANSFER > ACCESS > IDLE_IO. Two cycle definition signals, ACC[1..0], are output from the STATM pal to indicate the cycle type. The four defined cycle types are encoded inside the STATM pal according to the table below. ACC[0] is then modified for external use by driving it to a logic 1 during the SWITCH state, thereby indicating the ACCESS state, which results in earlier decoding of EISA memory cycles in the DECODE pal.

	acc[1]	acc[0]
IDLE_IO	**0**	**0**
ACCESS	**0**	**1**
TRANSFER	**1**	**0**
REFRESH	**1**	**1**

IDLE_IO cycles are inserted whenever there are no requests pending, or when EISA I/O register accesses are taking place. /RAS and /CAS remain inactive. Instead STATM generates a data strobe, /DS, for I/O cycles. If another type of cycle is in progress when an I/O cycle is requested by the EISA interface (which presents the address then activates the /START signal), then the STATM pal activates /HSTB, which is used by the EISADR pal to deactivate EXRDY for the EISA interface, thereby forcing the EISA interface to wait. Once the IDLE_IO cycle begins, then EXRDY is reactivated so that the EISA cycle will complete.

Note that accesses to the SRAMs use the memory address space, not I/O space. Since this is only enabled when the SRAMLD bit of the MODE register is set, this bit is used by STATM and other pals to treat these accesses as I/O cycles, using /DS.

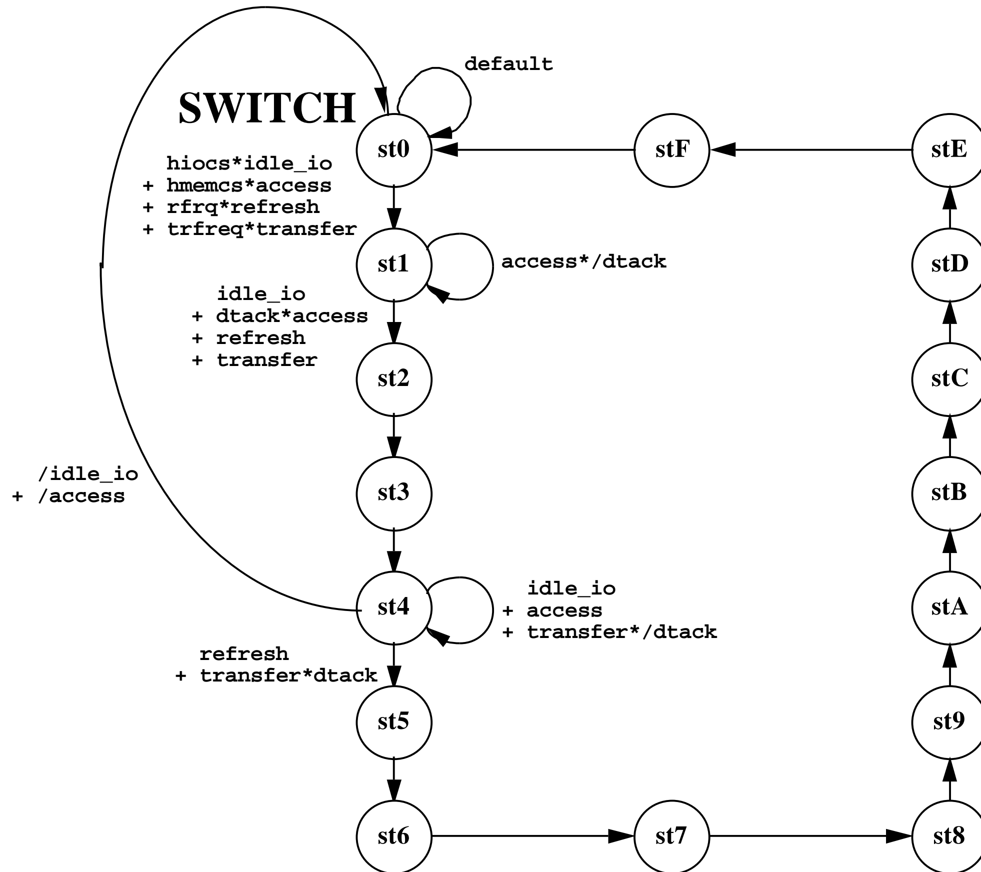
TRANSFER cycles are performed every 512 cycles of the sample clock according to the counter output A13 from the CLOCK pal. On receiving a negative transition on A13, the data-transfer-request flip flop DTREQ within the STATM pal is set, indicating that a transfer cycle is required. At the start of the transfer cycle the IOTR pal is notified, after which the /RASIN (the row-address-strobe input of the DRAM controller) is activated. This ensures that /DTOE and/or /WBWE are active at the negative transition of /RAS, so starting a transfer cycle rather than a read/write cycle.

After a number of clock cycles, /RASIN is deactivated and the transfer ends. During this time the sample count is routed through to the VRAM address inputs.

ACCESS cycles are more straightforward. The cycle is requested by the EISA interface, which presents the address then activates the /START signal. This results in the activation of the /EISAMEMCS output by the EISAIF pal. /RASIN is asserted early in the cycle, as with a transfer cycle, but this time notification to the IOTR pal is delayed so that the negative transition of /DTOE and/or /WBWE occurs in between the negative transitions of /RAS and /CAS. If the DRAM controller activates its /DTACK output (to indicate a refresh cycle is in progress), then the STATM pal activates /HSTB, which is used by the EISADR pal to deactivate EXRDY for the EISA interface, thereby forcing the EISA interface to wait. When /DTACK is deactivated then EXRDY is activated so that the cycle will complete.

REFRESH cycles occur every 16µs according to an internal refresh clock generated within the DRAM controller by a division of its CLK input. Every 16µs the DRAM controller activates its refresh request signal, RFRQ. The next cycle performed by the state machine will then be a REFRESH cycle. During the whole of this cycle, the STATM pal will activate the RFSH input of the DRAM controller. The DRAM controller looks after the rest, routing the contents of its internal refresh counter onto the VRAM address inputs.

The state transition diagram for the state machine is shown below :



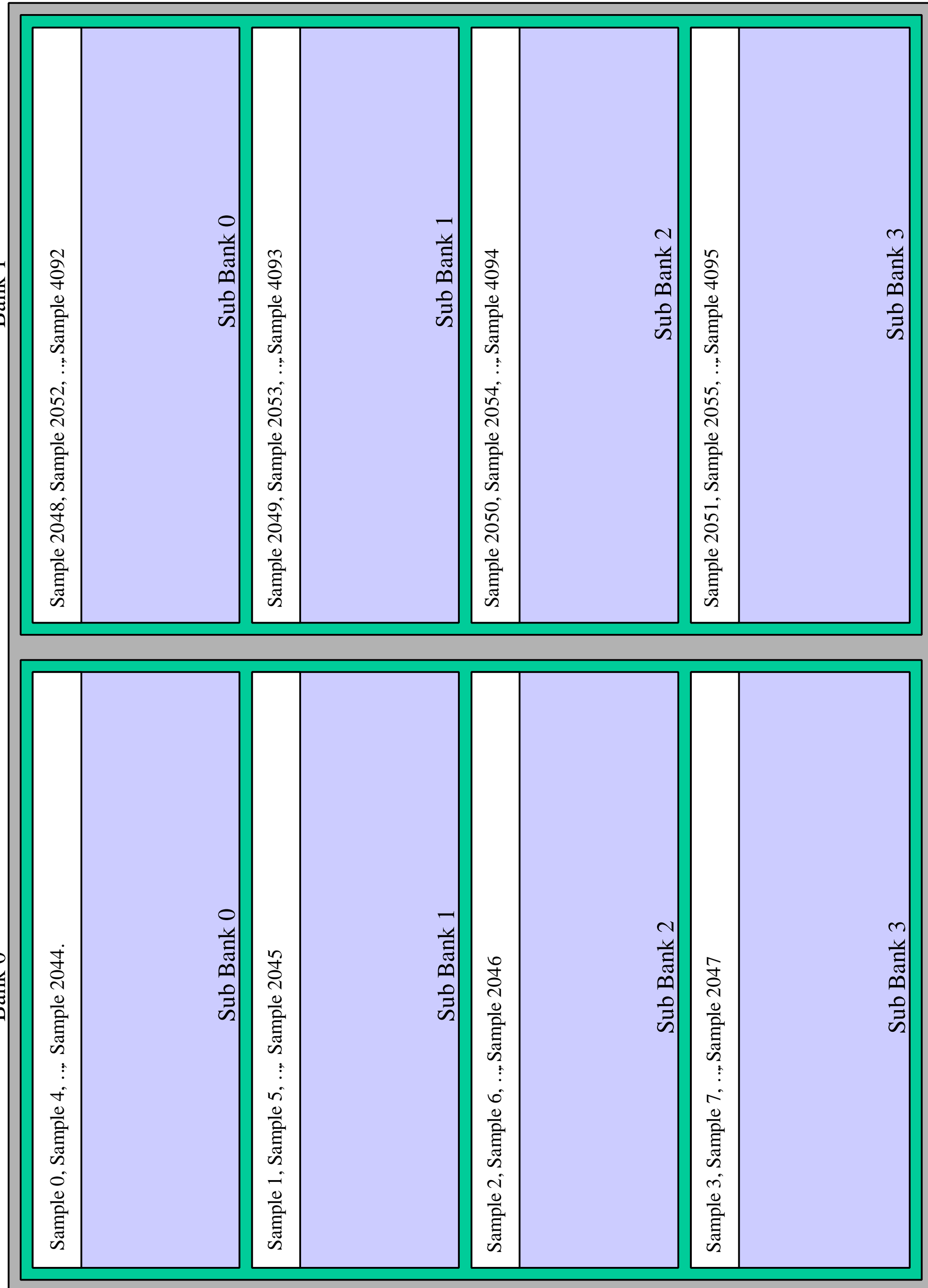
DRAM Controller Initialization

On system reset, or a write to the RESET register, the DECODE pal activates its /RESET output to initialize the board. Amongst other things, this connects to the DRAM conroller mode load input, /ML, which causes it to load its operating mode from its B1 and B0, /ECAS0, /CS, R9-0 and C9-0 inputs, which various other pals set to the following logic levels while /RESET is active :

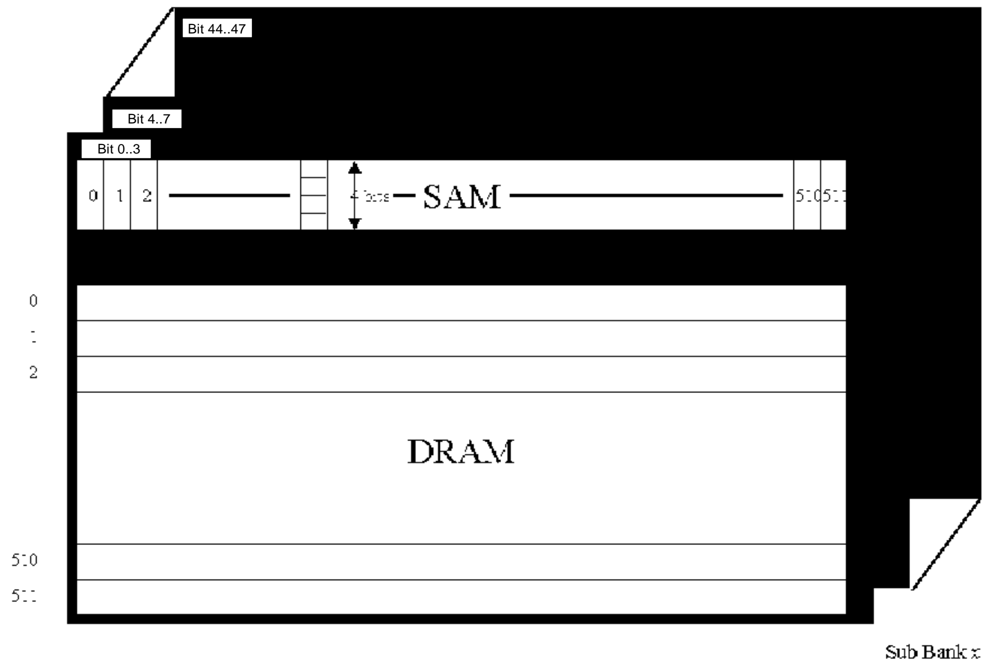
signal	value	mode
B0	0	/ADS clocks address latches
B1	1	access mode 1 /ADS starts /RAS3-0 /AREQ ends /RAS3-0
/ECAS0	1	/CAS3-0 follows /ECAS3-0 external refresh with /RFRQ (not /WE)
/CS	1	<< THERE IS A PROBLEM HERE ? >> /CS is not deliberately disabled during mode loading (nor is /AREQ or /ADS).
C9	0	/CAS3-0 are same for reads and writes
C8	1	row address hold time = 15nS
C7	1	column address hold time = 0nS
C6	1	each combination of B1 and B0 activates one of /RAS3-0 and /CAS3-0, i.e. /RAS0 and /CAS0, /RAS1 and /CAS1, etc.
C5	1	
C4	1	
C3	0	refresh clock = DELCLK / 360
C2	1	
C1	0	
C0	0	
R9	1	staggered refresh
R8	1	no address pipelining
R7	1	/DTACK generated (not /WAIT)
R6	0	/DTACK delayed by one CLK by /WAITIN=0
R5	0	no wait states /DTACK remains activated during bursts
R4	0	
R3	0	no wait states /DTACK remains activated when /RAS is
R2	0	
R1	1	/RAS activated during refresh for 4 CLKs /RAS precharge time = 3 CLKs
R0	1	

VRAM Layout

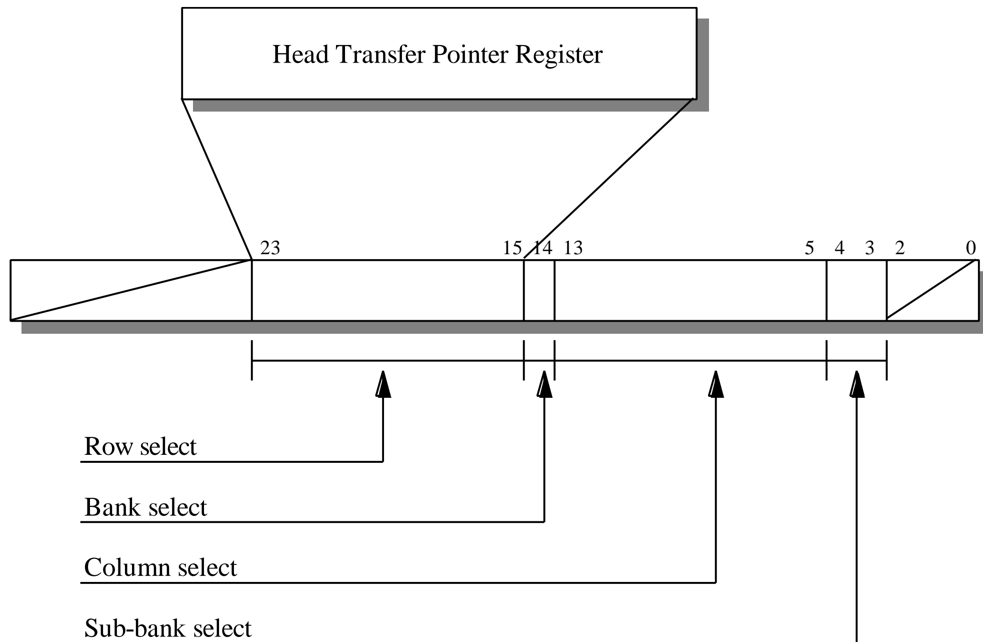
The above descriptions give a simplistic view of the VRAM structures. These are systematic but can be confusing without further explanation. The trace samples are stored in the subbanks as shown below :



Each VRAM sub-bank is effectively as follows :



Thus the rows, banks, columns and sub-banks are selected as follows :

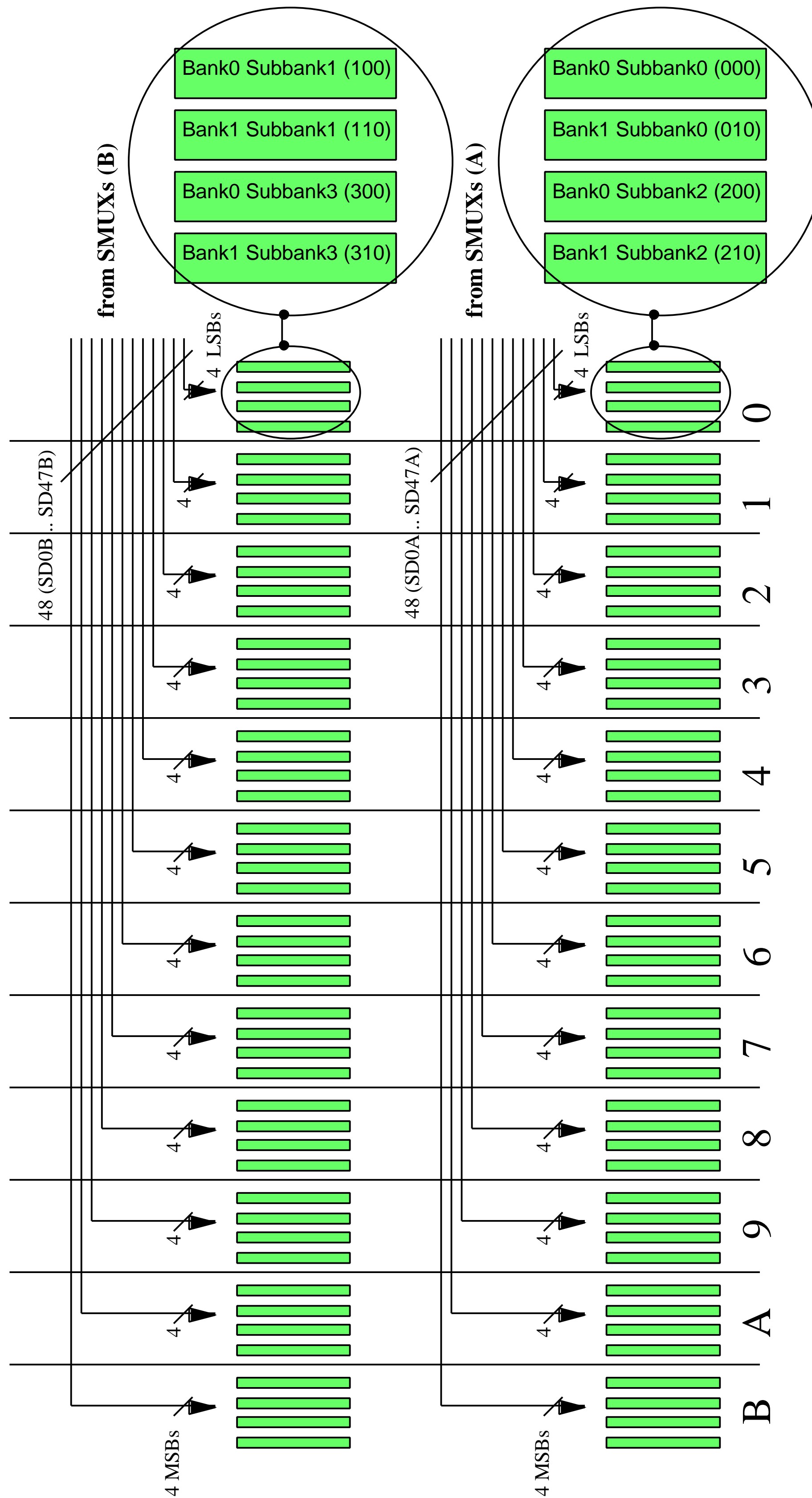


Therefore the samples are distributed amongst the sub-banks as tabulated below :

Sample #		Sub Bank 0		Sub Bank 1		Sub Bank 2		Sub Bank 3	
Row 0(Bank 0) (8Kb)	0	0	0000	1	0008	2	0010	3	0018
	1	4	0020	5	0028	6	0030	7	0038
	2	8	0040	9	0048	10	0050	11	0058
	...	...	...	...	...	...	...	...	...
	510	2040	3FC0	2041	3FC8	2042	3FD0	2043	3FD8
	511	2044	3FE0	2045	3FE8	2046	3FF0	2047	3FF8
Row 0(Bank 1)	0	2048	4000	2049	4008	2050	4010	2051	4018
	...	...	...	...	...	...	...	...	...
	511	4092	7FE0	4093	7FE8	4094	7FF0	4095	7FF8
Row1(Bank 0)	0	4096	8000	4097	8008	4098	8010	4099	8018
	...	...	...	...	...	...	...	...	...
	511	6140	BFE0	6141	BFE8	6142	BFF0	6143	BFF8
Row1(Bank 1)	0	6144	C000	6145	C008	6146	C010	6147	C018
	...	...	...	...	...	...	...	...	...

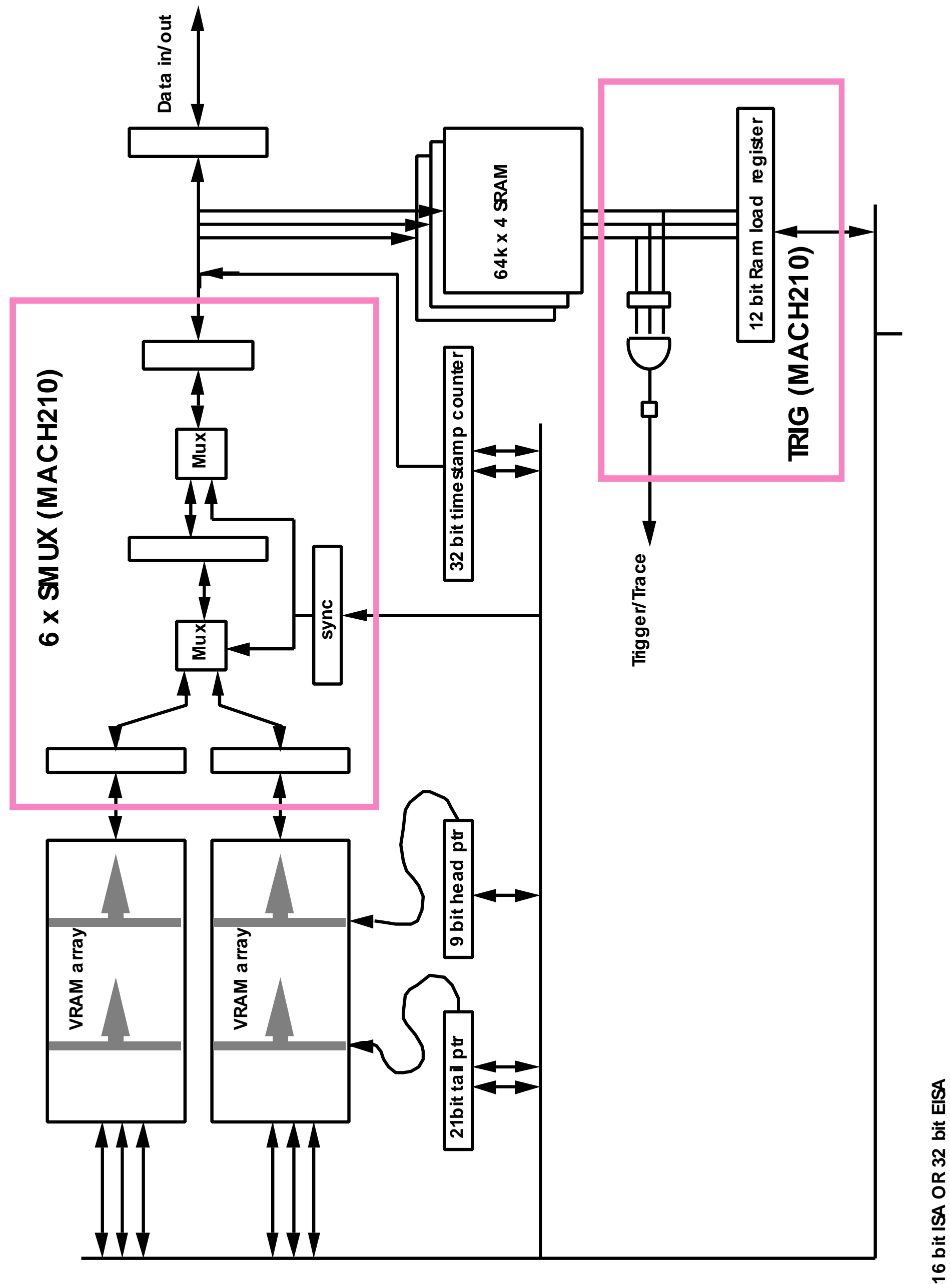
Row 511(Bank 0) FF8000-FFBFFF
Row 511(Bank 1) FFC000-FFFFFF

The serial data is connected from the SMUX pals to the VRAMs as follows :

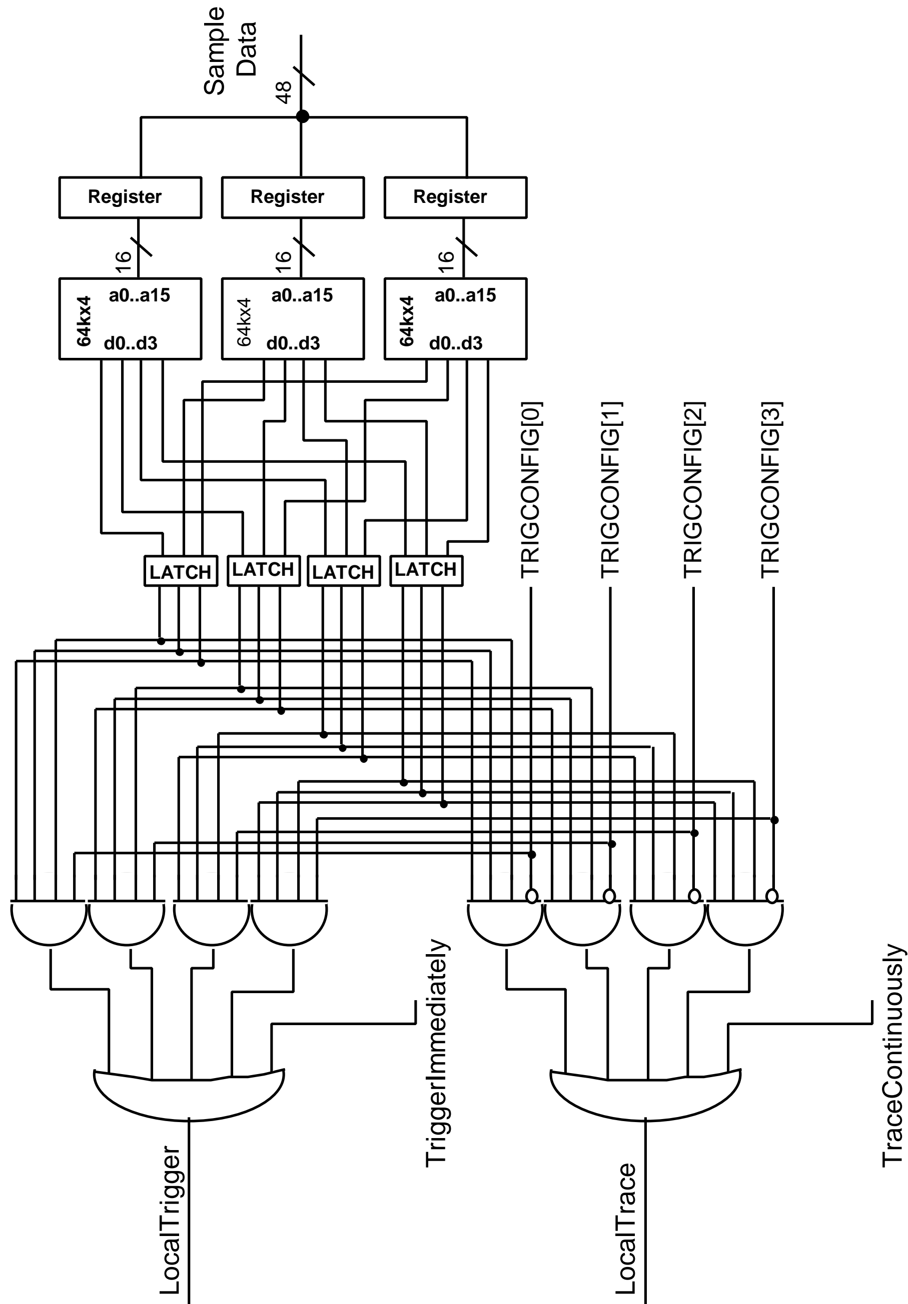


Trigger/Trace and Timestamp Logic

The sample data signals at the SCONN connector at the right edge of the board are buffered via SN74AS646N buffers SBUF0-5, whence they connect to the SMUX pals and the VRAMs as described directly above. The trigger/trace patterns are stored in SRAMs; the address inputs of these are fed with the buffered sample data as shown in the diagram below. The diagram also shows how the timestamp counter can be substituted for the buffered data, one byte at a time; if any of SBUFENA[5..0] are at a logical 1 then the appropriate sample data buffer is enabled, otherwise the appropriate timestamp output buffer is enabled, the lower three bytes via the TIME0 pal and the upper three bytes via TIME1.



The trigger and trace logic of the TRIG pal is shown below.



The sample data is connected to the address lines of SRAM0, SRAM1 and SRAM2. A triplet of data bits from each SRAM (e.g. d0 on SRAM0 & SRAM1 & SRAM2) is called a *trace channel* and is individually selectable to be a *trigger* (if trigger value is matched, stop sampling when STOPCOUNT register reaches zero) or a *trace* (store sample only if trace value is matched).

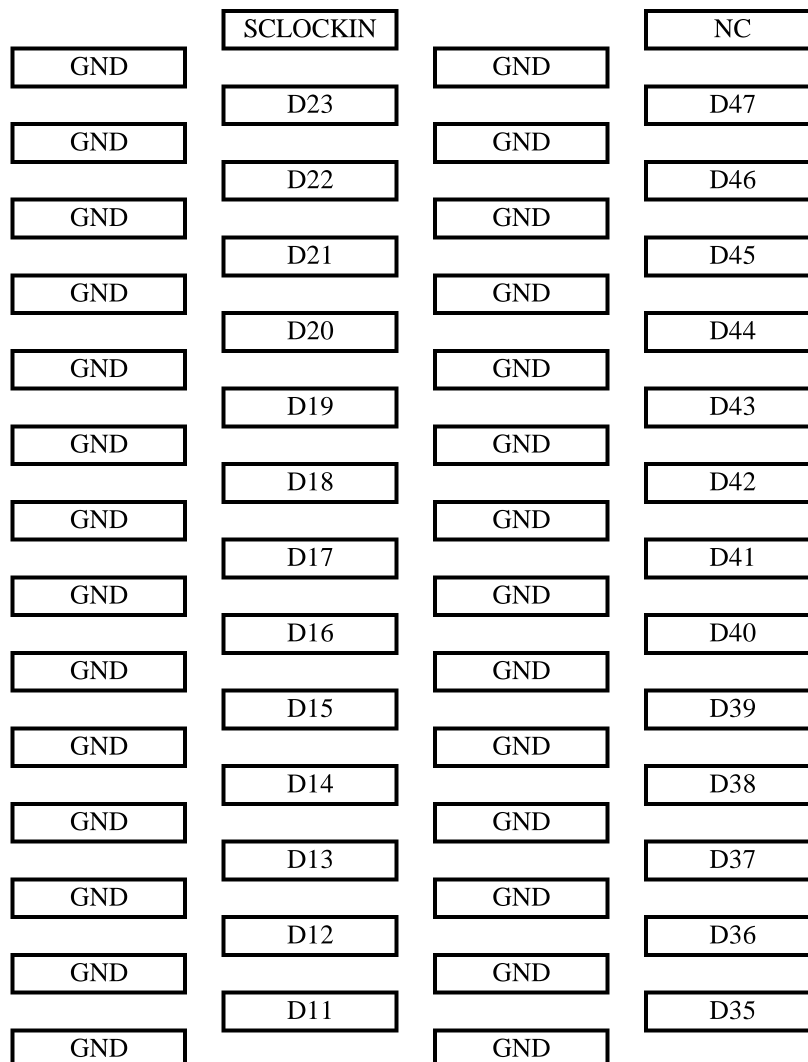
Channels are selected as trace or trigger by writing the TRIGCONFIG register. Bits [3..0] of this register individually select the four trace channels as trigger (if set to one) or trace (if reset to zero). The TRACE CONTINOUSLY bit forces acceptance of every sample. The TRIGGER IMMEDIATELY bit forces the STOPCOUNT to begin counting down immediately.

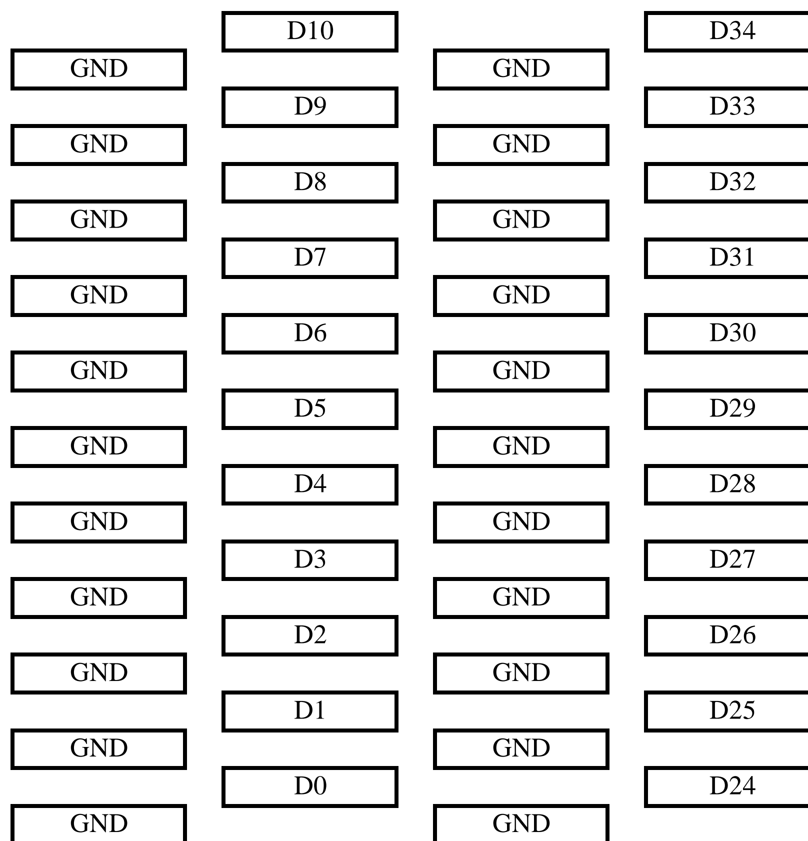
To read or write the SRAM, first the SRAMLD bit of the MODE register must be set, and the RECORD bit must be reset. This disables sampling and maps the SRAM into the lower 128k of the EISA memory address space. Usually PAUSE would also be set. All three SRAMs are read/written together. The EISA address is buffered in triplicate onto the sample data path (via the SMUX pals) and the SRAM0 data is buffered to/from EISA D[3..0], SRAM1 to/from EISA D[7..4] and SRAM2 to/from EISA D[11..9], all via the TRIG pal.

The CLOCK pal accepts the LOCALTRACE and LOCALTRIGGER signals from the TRIG pal, combines these with EXTRA2-0 and EXTRIG2-0, and generates appropriate clocks for the CNTCMP pal. It also contains 9bit counters that track the progress of the VRAM sequential access memory (SAM) pointer, and issue a request to STATM for a VRAM transfer cycle when they roll over from `111111111` to `000000000`.

Sample Data Input

The pinout for the SCONN connector at the right edge of the board (from the component side) is as follows :





towards EISA connector

All signals are terminated via 22? series resistors.

When sampling data, it is imperative that there are no false clock transitions, so the sample clock SCLOCKIN is timed in a latch that inhibits spurious clock transitions for a small interval of time after each incident clock transition, i.e. it enables the DT200.1 to respond only to the incident wave of the SCLOCKIN input signal. This latch circuit is implemented in the CLKDR pal.

Width Expansion

The pinout for the EXTCON connector at the top of the board is as follows (from the component side) :

towards VRAM

/SYNC	1	20	GND
/PAUSEOC	2	19	GND
/CSYNC	3	18	GND
/EXTRAC2	4	17	GND
/EXTRAC1	5	16	GND
/EXTRAC0	6	15	GND
/RESET	7	14	GND
/EXTRIG2	8	13	GND
/EXTRIG1	9	12	GND
/EXTRIG0	10	11	GND

towards SCONN

/EXTRIG0-2, /RESET, /EXTRAC0-2 and /CSYNC are terminated via 22? series resistors, while /PAUSEOC and /SYNC are terminated to 4.7k ? pullup resistors. The internal end of the 22? series resistor via which /RESET terminates is also connected to a 4.7k ? pullup resistor.

Pin /PAUSEOC is wired to pin /CSYNC. This appears to be an error.

The triggering, etc. may be extended across up to four boards by connecting a 20-way flat cable across each EXTCON connector :

	board 0	board 1	board 2	board 3
/SYNC	/SYNC	/SYNC	/SYNC	/SYNC
/PAUSEOC	/PAUSEOC	/PAUSEOC	/PAUSEOC	/PAUSEOC
/CSYNC	/CSYNC	/CSYNC	/CSYNC	/CSYNC
/EXTRAC2	/TRAC2	/TRAC2	/TRAC2	/TRAC2
/EXTRAC1	/TRAC2	/TRAC1	/TRAC1	/TRAC1
/EXTRAC0	/TRAC0	/TRAC0	/TRAC0	/TRAC0
/RESET	/RESET	/RESET	/RESET	/RESET
/EXTRIG2	/TRIG2	/TRIG2	/TRIG2	/TRIG2
/EXTRIG1	/TRIG1	/TRIG1	/TRIG1	/TRIG1
/EXTRIG0	/TRIG0	/TRIG0	/TRIG0	/TRIG0

where : /TRAC0 == /LOCALTRACE from board 0
/TRIG0 == /LOCALTRIGGER from board 0

Thus at all but one board, /EXTRAC0-2 and /EXTRIG0-2 are used to output the / LOCALTRACE and /LOCALTRIGGER from that board, whereas on one board / EXTRA2-0 and /EXTRIG2-0 are used to input the equivalent signals from the other boards.

The ADDRMAP register must be set to a unique base address for each board. The triggering requirements, etc., must then be distributed across the boards.

Depth Expansion

The DT200.1 is designed to allow interleaving of boards to increase the buffer depth in multiples of two million, and to proportionately increase the sample rate. This requires external preprocessing of the sample clock to stagger individual SCLOCKIN signals across the boards.

The ADDRMAP register must be set to a unique base address for each board. The other register and SRAM settings must then be duplicated across the boards.

Electrical Characteristics

minimum logical 1 input voltage	2.0 Volts
maximum logical 1 input voltage	5.5 Volts
minimum logical 0 input voltage	-0.5 Volts
maximum logical 0 input voltage	0.8 Volts

capacitive loading per input under test TBD

minimum SCLOCKIN frequency	0 Hz
maximum SCLOCKIN frequency	50 MHz
minimum SCLOCKIN logical 0 duration	5 nS
minimum SCLOCKIN logical 1 duration	5 nS
SD[0..47] setup time relative to SCLOCKIN positive transition	5 nS
SD[0..47] hold time relative to SCLOCKIN positive transition	1 nS
supply current	TBD

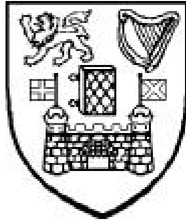
Parts List

The parts list is :

Part No	Supplier	Description	Quantity
DT2001	Exacta	PCB	1
DP8431V-33	STC 873795E	DRAM Controller	1
TC524256AZ-10	STC	VRAM	96
MCM6208C	Motorola	64k x4 SRAM	3
SN74AS646N	STC 405697F	8bit bidirectional registers/buffers	10
Lattice GAL16V8-7	STC 403938C	PAL	2
AmPAL22CEV10H/4	STC 403521R	PAL	2
MACH210-12JC	STC 403544F	GAL	14
HM050H00	STC 015265C	50MHz Crystal oscillator	1
Bourns 4308R-102-22R	STC 001467B	22? 8pin SIP separate resistor pack	14
Bourns 4309R-101-4K7	STC 045827C	4k7 9pin SIP common resistor pack	11
Philips M-RS16-22K	STC 022296E	22k? 400mW resistor	3
Philips M-RS16-3K3	STC 022276B	3k3? 400mW resistor	4
Philips M-RS16-22R	STC 022312D	22? 400mW resistor	4
749076-9	AMP	100pin Cable socket	1
749111-8	AMP	100pin Cable plug	1
AVX Kyocera TAP10	STC 032904A	4.7µF tantalum capacitor	3
AVX Kyocera SR20 2F4 100nF	STC 016910E	0.1µF Reset timer capacitor	1
1PGAUAQ05Z	Rogers Mektron	Transmission line capacitor	19
Philips CZ15A103K	STC 407207R	0.01µF monolithic capacitor	40
Kemet C410X7R-10000	STC 091798F	0.01µF axial bypass capacitor	28
037752R	STC 037752R	20 Pin DIL sockets	2
037753G	STC 037753G	24 Pin DIL sockets	15
067628R	STC 067628R	44pin PLCC socket	14
067630X	STC 067630X	68pin PLCC socket	1

Note that boards 1 and 2 use Mitsubishi M5M442256AL-8 (STC 400999C) VRAMs rather than Toshiba TC524256AZ-10 VRAMs. Micron MT5C2564 SRAMs are a suitable alternative to the Motorola MCM6208C.

SCI Europe



Deep Trace DT200.1

Patch List

1. Surface Wire Patches

<REFER TO PATCH LOCATIONS DIAGRAM>

1.1 Connect the long trace at the top of each array of 48 VRAMs to Ground (Patch A) . This is necessary to ensure that Transfer cycless occur correctly. This line is marked NC in most VRAMs but the Mitsubishi M5M442256AL VRAMs use this line for special functions.

1.2 Connect the /ECAS lines on the DRAM controller to the / RAS lines on the DRAM controller (Patch B) It is only necessary to do this for / ECAS[3..1]. /ECAS[0] is connected by an internal trace. This patch also requires an internal trace to be cut (see Internal Trace Patches).

1.3 Connect CAB to SAB on each of the SN74AS646N latching buffers (Patch C).

1.4 Connect pins 56 (CLK) and 57 (DELCLK) on the DRAM controller (Patch D) to pin 13 (SYSCLK) on the EISAIF pal.

1.5 Connect the D1 line between VRAM no. XXX and VRAM no. YYY as per Patch E. This is a missing net in the file VRAM96.NET.

1.6 Terminate the clock line input on pin 1 of the CLKDR pal using 220? pullup and 330? pulldown (Patch F).

1.7 Cut the trace on the solder side of the board as shown for Patch G. This accidentally connects a data line to an IRQ line on the EISA bus.

1.8 Connect pin 11 to pin 12 on each of the SRAMs (Patch H). This i s a missing net for the SRAM /E input in the file CORE.NET, which needs to be taken to logic 0.

1.9 Connect CBA to SBA on each of the SN74AS646N latching buffers (Patch I). This is needed to enable transparent operation of the buffers during writes so that the VRAM /DTWBC signal latches valid write data early in the memory cycle.

1.10 Connect the /ADDLATCH signal from the DECODE pal to the CNTCMP pal (Patch J).

1.11 Connect the H14 signal from the CLOCK pal to the STATM pal (Patch K).

1.12 Connect a 4.7k? resistor between pins 60 (/WAITIN) and 55 (VCC) on the DRAM controller (Patch L).

1.13 Connect pin 23 (A3) of any VRAM to pin 83 (Q3) on the DRAM controller (Patch M).

1.14 Connect the /INIT signal from the CLOCK pal to pin 11 of the IOTR pal.

2. Internal Trace Patches

This is easier than it sounds. You just have to dig a tiny hole on the surface of the board and break the internal trace. It is most easily accomplished using a Stanley knife with a new blade. Be careful not to apply too much pressure or cut through to the power planes. The locations of the internal trace cuts are shown in two diagrams.

2.1 Solder-side cuts (see Solder-side-cut Diagram). These cuts are necessary to shorten the length of WB/WE and DT/OE traces on the VRAM array.

2.2 Component-side cuts (see Component-side-cut Diagram). These cuts are necessary to break traces connecting the ECAS[3..0] lines to ground. Because these cuts are situated underneath the DRAM controller they have to be accomplished before the PLCC sockets are attached to the board.

2.3 Component-side cuts (see Component-side-cut Diagram). This cut is necessary to break a trace connecting the EXTCON./PAUSEOC pin to the EXTCON./CSYNC pin.

3. Remaining Problems

3.1 New power- ON reset initializes the DRAM controller, but has not been checked thoroughly.

3.2 The DRAM R1 input appears to remain at logic 0 during DRAM initialization, even though it should be logic 1. This should be investigated. Note that during the initialization, the /CS, /ADS and /AREQ signals are not deliberately disabled - this implies that the reset is " /CS-programmed".

3.3 The GLOBAL reset may need to be changed in the STATM pal to ensure the power- ON reset is long enough. At the present the flip-flop that generates it is asynchronously reset by it, so only a very short pulse will be generated.

3.4 For some reason, /EISAIOCS is activated for one 25MHz cycle during EISA refresh, even though the equation for is deliberately equated to logic 0. There may be crosstalk problems, or an internal wiring short.

3.5 EISA bursts are disabled in the STATM pal.

3.6 The MODE bits DMAENA and IRQENA are not implemented yet, nor is any of the DMA logic.

3.7 One of the pullup packages is still missing from board 2.

3.8 There may be a wiring error within the boards, since EXTER1.R3B is shown in the netlist (which is probably out of date) as connected to both /PAUSEOC and /SYNC. The wiring diagrams should be checked.

3.9 It is still unclear whether there is a **SERIOUS** problem with turn-around of the VRAM SAM buffers during serial transfers. When /RECORD is toggled, the STATM pal responds by doing a pseudo-write cycle to turn around the buffers, but this only works for the currently selected VRAMs. What about the other VRAMs that will become selected as the sampling proceeds ?

4. Major Recent Fixes

4.1 3-APR-1998 : added comments to the pal source files, which were completely without comments.

4.2 3-APR-1998 : fixed irregular failure to boot due to lack of power- ON reset of the DRAM controller, by changing the EISADR pal to generate an abnormal pattern on /WR and /ADDLATCH whenever RESDRV=1, and changing the DECODE pal to recognize this and activate /BOARDRESET (which drives /RESET, which drives the DRAM input /ML). Probably the boot was very system-sensitive before this, since some systems may scan all of memory during boot and on finding the trace board would get no EXRDY and then wait forever.

4.3 3-APR-1998 : /EISAIOCS remained active after /CMD was deactivated, due to a latching "bridge" term DELCS in the /EISAIOCS equation in the EISAIF pal - this term bridges any gap between deactivation of /START and activation of /CMD. This was fixed by adding the NOST1 and NOST2 to recognize the end of /START, then limiting DELCS to 2 cycles after that.

4.4 9-APR-1998 : fixed system-sensitivity of write data latching on the cd[31..0] local bus, caused by a CBA positive transition before the EISA data became valid - this only happened for systems that generate valid EISA write data later than usual. Fixed by delaying the CBA positive transition by adding "+s2+s3" to the equation (in the STATM pal) for OUTCLK, which drives CBA.

4.5 9-APR-1998 : connected SBA to CBA on all the EISA buffers, so that when CBA=0, the buffers pass data transparently from the EISA bus to cd[31..0], instead of only generating the latched write data after the CBA positive transition. This ensures the write data is passed to the VRAMs as early as possible.

4.6 28-APR-1998 : properly implemented the ADDRMAP register.

4.7 29-APR-1998 : changed the STATM state machine conditions to qualify them by the select inputs, so that a spurious select would cause a default branch back to SWITCH. This dramatically improved the stability of the state machine. Also qualified other signals accordingly.

4.8 17-MAY-1998 : changed the timing of /DTWBC for writes by delaying its activation in the STATM pal from st2 to st3, since its negative transition (which clocks write data into the VRAMs) was occuring before the write data was valid.

4.9 19-MAY-1998 : connected the SRAM /E inputs to GND (logic 0) to allow them to be accessed.

4.10 27-MAY-1998 : fixed up the triggering and tracing equations in the TRIG pal.

4.11 27-MAY-1998 : added the CONTINUOUS TRACE and IMMEDIATE TRIGGER functions to the TRIG pal.

4.12 28-MAY-1998 : added the STATUS register by decoding both /TRREGCS and /HIDCS for address `0xZ020`, and activating DONE on bit 15 for a read, where DONE is the STOPCOUNT signal TEND.

4.13 31-MAY-1998 : added the EISA identifier `'DTR200.1'` to EISAIF, CNTCMP and TRIG pals.

4.14 2-JUN-1998 : added the EISA configuration file !DTR2001.CFG.

4.15 5-JUN-1998 : fixed the timing of /DS for all the I/O cycle types.

4.16 5-JUN-1998 : added wait states for I/O cycles that are requested by the EISA bus while the state machine is busy (probably with refresh) to the HSTB equations in the STATM pal.

4.17 5-JUN-1998 : fixed the EXRDY equations for the EISADR pal. They were deactivating EXRDY for all non-memory accesses !

4.18 13-JUL-1998 : fixed the equations for /WBWEB0LWR32, /WBWEB0UPR16, /WBWEB1LWR32 and /WBWEB1UPR16 in the IOTR pal by substituting `*(/trfbank+/init)` for `*/trfbank` and `*(/trfbank+/init)` for `*trfbank`. This required the assignment of a new /INIT input to pin 11 of the IOTR pal, plus a wire from CLOCK./INIT to IOTR./INIT.

4.19 22-JUL-1998 : included `*/hidcs` in the equations for D[0..11] in the TRIG pal, since otherwise there is nothing to prevent the SRAM contents from being read when the EISA ID is being accessed.

4.20 20-AUG-1998 : added outputs /SCLOCKC and /SCLKDLY to the CLKDR pal to form the SCLOCKIN input latch. This inhibits spurious transitions on the /SCLOCKA and /SCLOCKB outputs for small period of time after each valid transition, thereby tolerating ringing on the SCLOCKIN input. It requires a resistor from /SCLOCKC (pin 17 of CLKDR) to /SCLKDLY (pin 11 of CLKDR) and a capacitor from /SCLKDLY (pin 11 of CLKDR) to GND (pin 10 of CLKDR).

4.21 21-AUG-1998 : terminated the SC inputs to the VRAMs with 100 ? to GND, since otherwise substantial ringing occurs on negative transitions, causing extra unwanted clocking.

4.22 13-OCT-1998 : fixed the TRFREQ equation for the STATM pal. It was clearing TRFREQ in state (sE + sF) of a REFRESH as well as a TRANSFER cycle. This resulted in lost transfer cycles.

4.23 23-OCT-1998 : qualified the ACC[0..1] equations for the STATM pal with a new ARBITRATE signal that is itself qualified with a new ARBHOLD signal. ARBITRATE and ARBHOLD are only activated during the SWITCH state; ARBITRATE is activated for one cycle then ARBHOLD is activated for the next. The latter gives the state machine a cycle in which to respond without fear of arbitration changes (prior to this fix the arbitration could change at the end of the cycle in which the state machine was responding).

4.24 23-OCT-1998 : qualified the ACC[0..1] decoding into cycle types in the STATM pal with a new CYCLE_END signal that prevents mis-activation of other signals when the state machine has just re-entered the SWITCH state at the end of a cycle.

4.25 23-OCT-98 : removed the TRFREQ term from the /CS equation in the STATM pal, since otherwise /CS is deactivated too early by the deactivation of TRFREQ at state sE of the state machine.

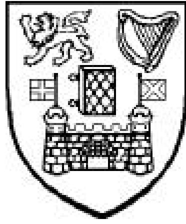
5. Still to be Done

5.1 Tracing and triggering from the timestamp counter still hasn't been successfully tested, despite many attempts.

5.2 Tracing and triggering from the timestamp counter needs to be tested from an external source, perhaps a logic tutor generating a pseudo random number.

5.3 A program needs to be written to gener ate a plot of node layouts for MACH210 (and MACH435/445) pals from the .PLC files. This would greatly simplify the manual placement that becomes necessary when the pal utilization is greater than, say 80%.

SCI Europe



Deep Trace DT200.1

Document History

Document Sources

1. This document uses design information from the archive of Philip O'Carroll's directory from Inma Kinsella's Macintosh (see Philip O'Carroll 5-NOV-96 in archive VOLUME3), and information in the file TOLSYS.ZIP in the same archive (VOLUME3), which is duplicated in the PC Archive VOLUME20.

2. The files DEEPTRA.DOC, DT200E.DOC and PATCHES.DOC appear to have been created by Word6 for Windows.

3. The file IDAPROP reads into Word, but contains what appears to be formatting information.

4. It has not yet been possible to establish what application was used to create the files DT-TRIG and LAYOUT. There are also versions of these files without a resource fork.

5. The file TOLSYS.ZIP contains the most comprehensive collection of original TraceBoard info.

6. The original design and manufacturing files have not yet been located. The manufacturer, Exacta Circuits Limited, Selkirk, Scotland, no longer have copies of the manufacturing files.

Revision History

Revision 1 : 6th June, 1998

Revision 2 : 28th October, 1998