efficēon

Development Tools

Version 1.2

Transmeta™ Efficeon™ TM8x00 1.x/2.x
Code Morphing™ Software 6.x

Transmeta PROPRIETARY Information
Provided Under Nondisclosure Agreement

Preliminary Information—SUBJECT TO CHANGE

August 26, 2004

Transmeta
CORPORATION

Property of:

Transmeta Corporation
3990 Freedom Circle
Santa Clara, CA 95054
USA
(408) 919-3000
http://www.transmeta.com

Directory of Contents

Chapter 1

Configuration Tools Overview

The following chapters provide information about tools provided by Transmeta that are used to implement, configure, and debug a system based on a Transmeta Efficeon™ processor. Tools include the following:

- **ROM Creation Utilities and Templates (page 9)**

  Utilities for assembling and examining a 2MB ROM image. Includes **makerom**, **print-oct**, **print-ver**, **disasm-rom**, and **rom-format**.

- **Bringup Tools (page 21)**

  Console tool for viewing results of Power-On Self Test codes while booting (**POSTCodeTool**), in-system programming of LPC flash memory (**FlashTool**), system reset (**ResetTool**), processor state listing (**scandump**), and debug module detection (**TDM2Detect and defaultTDM**). The **DOSFlash** suite of utilities enables ROM management from DOS.

- **Debugging Hardware Interfaces and Tools (page 29)**

  The LPC interface, system debug interface, and physical debugging devices.

- **Virtual x86 In-Circuit Emulator (VICE) (page 35)**

  In-circuit emulator and debugger.

1.1 ROM Image Creation

This section describes the process of creating a system ROM image, and then programming the image into an LPC flash ROM using the Efficeon™ TM8x00 ROM tools.

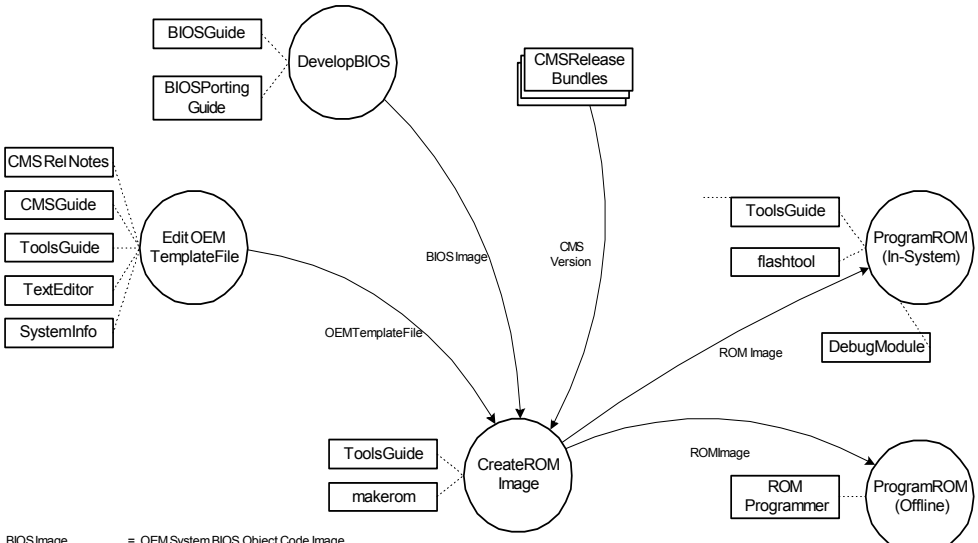
For detailed information about ROM images, see Chapter 2 *Code Morphing Software Image* in the *Efficeon System Configuration Guide*.

An Efficeon system ROM contains Code Morphing software code, boot code, OEM system-specific configuration information in the OEM configuration table, and the system BIOS code. The components of the system ROM image are combined into a single programming image using utilities described in *ROM Creation*

Utilities and Templates on page 9. ROM programming images created by this process are programmed into the target system LPC flash ROM using a Transmeta flash ROM programming tool (see *flashtool* on page 22). When the system is rebooted, power-on self-test (POST) codes can be viewed with *POSTCodeTool* (see page 21).

The figure below graphically shows the Efficeon TM8x00 LPC flash ROM image creation and programming process. The associated Efficeon TM8x00 tools and documents required for each step in the process are shown in the rectangular boxes connected to either side of the activity bubbles in the figure.

Efficeon TM8x00 ROM Image Creation and LPC Flash ROM Programming



BIOS Image = OEM System BIOS Object Code Image
BIOS Guide = Efficeon TM8x00 BIOS Programmer's Guide
BIOS Porting Guide = Efficeon TM8x00 BIOS Porting Guide
CMS Guide = Efficeon TM8x00 Code Morphing Software Guide
CMS Image = Efficeon TM8x00 Code Morphing Software Object Code Image
CMS Release Bundles = Efficeon TM8x00 Code Morphing Software Release Bundle File(s)
CMS Rel Notes = Efficeon TM8x00 Code Morphing Software Release Notes
OEM Template = Efficeon TM8x00 CMS OEM Configuration Table Definition File
ROM Image = OEM System LPC Flash ROM Image
ROM Programmer = Off-Line Flash ROM Programmer
System Info = Efficeon TM8x00 System Components
Debug Module = Macraigor Raven Debug Module
Text Editor = Standard Text File Editor
Tools Guide = Efficeon TM8x00 Bringup/Debug Tools Guide

makerom = Efficeon TM8x00 ROM creation utility
flashtool = Efficeon TM8x00 Flash Programmer Tool

1.2 Installation

See the *Development Tools Release Notes and Errata* for system requirements.

The Development Tools installer requires the Microsoft .NET framework version 1.1 or later. This can be obtained by using "Windows Update" to install the latest system software.

Note that if you update your system regularly, it is very likely that you already have installed the latest .NET framework.

There are three components required in order to produce Code Morphing software ROM images:

- TM8x00 Development Toolset
- TM8x00 System Components
- TM8x00 Code Morphing Software

Installing the Development Toolset

Install the TM8x00 Development Toolset first. This installer provides a default path where the tools should be installed. The other two installers install in the same path as the TM8x00 Development Toolset directory without prompting.

Please note that Transmeta provides ".msi" files for installation. "MSI" is a short form for Microsoft Installer. These installers are supported only under Windows XP or Windows 2000/SP3.

The default installation path is:

```
C:\Program Files\Transmeta\TM8x00 Development Tool Set\
```

This path is recommended by Transmeta—all relevant documentation refers to that default path. It is highly recommended that this path be used to install the tools.

Note that the installer installs mac_tdmd.exe as a Windows service. After installing the DevTools package, you must perform a few steps to start mac_tdmd.exe in order to use the Macraigor Raven debugger to communicate with your target system. For details, see *Connecting and Configuring the Macraigor Raven* on page 33.

To install the Development Toolset, double-click on the installer and follow all instructions.

Installing the System Components

The tools are now accessible from a command prompt. However, a ROM image can not be created until the System Components and Code Morphing software are installed.

Note that these installers use the installation path provided by the Development Toolset.

Run the System Components installer. Click Next to continue, and then click Next at the confirmation page to continue installation.

Installing Code Morphing Software

After installation finishes, run the Code Morphing software installer. Click Next to continue, and then click Next at the confirmation page to continue installation.

The tools, components, and Code Morphing software are now installed and ready to program.

1.3 Example: Creating a ROM Image Programming File

There are several steps to create a ROM image and upload it to an Efficeon system, as shown in the following example. These examples assume that the tools software, components software, and Code Morphing software have been installed as described in the previous section.

1. Copy the OEM-template.rcl file and edit.

   The file OEM-template.rcl contains a basic template for the OEM configuration table. Fields in the table are described in *OEM-template.rcl* on page 13.

   a. Copy this file to working directory from the sub-folder `TM8x00 System Components` in the tools installation folder.

   b. Edit the file using any text editor.

2. Create a ROM image using makerom.

   Once you have a system BIOS, completed OEM-template.rcl file, and Code Morphing software, you can assemble a system ROM image file using makerom, as in this example:

   System BIOS image file: MyBIOS (binary)
   OEM configuration table: OEM-template.rcl
   Code Morphing software: version 6.0.0#9

   ```
   makerom MyImage OEM-template.rcl 6.0.0#9 MyBIOS
   ```

   The resulting binary file MyImage contains the new system ROM.

3. Connect debug device to your system, and select it using the instructions in *Debugging Interfaces* on page 30.

4. Flash the ROM image into your system.

   Use flashtool to flash the image onto the system ROM using the LPC bus:

   ```
   flashtool MyImage
   ```

5. Restart the system:

   ```
   resettool -r
   ```

6. Start POSTCodeTool, which also resets the system when using a TDM2 device.

   ```
   postcodetool
   ```

Chapter 2

ROM Creation Utilities and Templates

2.1 makerom

Create a 2MB ROM image with customer's BIOS. Requires a Code Morphing Software revision in the format x.y.z#a (for example, 6.0.0#9) and an OEM definitions file. If the BIOS image file is omitted, a 1344kb CMS image is created, without an embedded BIOS.

Note that OEMTemplate.rcl *must* be edited prior to using makerom. See *OEM-template.rcl* on page 13.

makerom Usage

```
makerom [options/switches] <output> <OEM> <CMS> [BIOS]
  -V       | --version          Print version information for this utility
  -h       | --help             Print usage information and exit
  -v       | --verbose          Print additional information
  <output>                      (REQUIRED) output image file (binary format)
  <OEM>                         (REQUIRED) input OEM definitions file
  <CMS>                         (REQUIRED) input CMS revision
  [BIOS]                        Optional input BIOS image file
```

For information about the OEM definitions file, see *OEM-template.rcl* on page 13.

2.2 print-ver

"print-ver" extracts and prints version information from a 2MB ROM image. It is useful when the origin of a particular image is unknown. It can be used on images created by makerom (see *makerom* on page 9), or on images extracted from the LPC ROM using flashtool (see *flashtool* on page 22).

print-ver Usage

```
usage: print-ver [options/switches] <image>
  -V        | --version              Print version information for this utility
  -h        | --help                 Print usage information and exit
  -o<value> | --output=<value>       output file containing version information
  -v        | --verbose              Print additional information
  <image>                           (REQUIRED) binary image file to extract
                                    version information from
```

print-ver Example

```
C:\>print-ver image.bin
Version Information from Image: "image.bin"
  Primary  Boot:       "20030926 00:47 official release 6.0.0#9"
  Primary  CMS:        "20030926 00:47 official release 6.0.0#9"
  Primary  OEM-Config: "4.2" @ 2003-10-03 10:37:27
  Recovery Boot:       "20030926 00:47 official release 6.0.0#9"
  Recovery CMS:        "20030926 00:47 official release 6.0.0#9"
  Recovery OEM-Config: "4.2" @ 2003-10-03 10:37:27
```

2.3 print-oct

print-oct extracts the OEM-specific section of the OEM configuration table from a 2MB ROM image and prints it in a human-readable format. It is useful when the exact settings in the table are unknown. It can be used on images created by makerom (see *makerom* on page 9), or on images extracted from the LPC ROM using flashtool (see *flashtool* on page 22) with "flashtool --dump".

For more information about the OEM configuration table, see *OEM-template.rcl* on page 13. :

print-oct Usage

```
print-oct [options/switches] <image> <CMS>
  -V         | --version               Print version information for this utility
  -b<num>    | --BIOSsize=<num>       size of BIOS (in KBytes)
                                       Default value = 704
  -h         | --help                  Print usage information and exit
  -o<value>  | --output=<value>       output file containing OEM Config Table
  -r         | --recovery              print recovery OEM Config table
  -v         | --verbose               Print additional information
  <image>                             (REQUIRED) binary image file containing
                                      OEM Config table
  <CMS>                               (REQUIRED) input CMS revision
```

print-oct Example

```
C:\>print-oct image.bin 6.0.0#9
vr_100mV_ramp_time      =>   0x0023; # @  256 (`oem)
vr_voltage[0]           =>   0x06d6; # @  260 (`oem)
vr_voltage[1]           =>   0x06a4; # @  262 (`oem)
vr_voltage[2]           =>   0x0672; # @  264 (`oem)
. . .
longrun_frequencies[6]  =>   0x0000; # @  648 (`oem)
longrun_frequencies[7]  =>   0x0000; # @  650 (`oem)
```

2.4 disasm-rom

disasm-rom breaks 2MB ROM images down into their components. It can be used on images created by makerom (see *makerom* on page 9), or on images extracted from the LPC ROM using flashtool (see *flashtool* on page 22).

disasm-rom takes an image file (image.bin) as input, and creates several files as a result. See below for an example. images.bin is created from either "flashtool --dump" or "makerom".

To use disasm-rom, you need to know which version of CMS was used to create the image. Use print-ver to identify the CMS version (see *print-ver* on page 9).

disasm-rom Usage

```
disasm-rom [flags/switches] <image> <CMS>
  -V         --version               Print version information for this utility
                                     and exit
  -b<num>    --BIOSsize=<num>        size of BIOS (in KBytes)
                                     Default value = 704
  -h         --help                  Print usage information and exit
  -o<value>  --output=<value>        output folder/file for extracted images
                                     Default value = "."
  -r<value>  --region=<value>        name of region to extract, default is to
                                     extract ALL regions.
                                     Valid choices are:
                                       BIOS
                                       primaryBoot
                                       primaryCMS
                                       primaryOEM
                                       recoveryBoot
                                       recoveryCMS
                                       recoveryOEM
  -v         --verbose               Print additional information during
                                     execution
  <image>                            (REQUIRED) binary image file to extract
                                     all components from
  <CMS>                              (REQUIRED) input CMS revision
```

Examples

```
C:\>disasm-rom image.bin 6.0.0#9
```

. . . the following files are created in the same directory:

```
    720,896 image-BIOS.bin      <--  704k BIOS image
     28,672 image-Boot-P.bin    <--  28k primary boot code
     28,672 image-Boot-R.bin    <--  28k recovery boot code
    655,360 image-CMS-P.bin     <--  640k primary CMS code
    655,360 image-CMS-R.bin     <--  640k recovery CMS code
      4,096 image-OEM-P.bin     <--  4k primary OEM config table
      4,096 image-OEM-R.bin     <--  4k recovery OEM config table
```

. . . the following file is also located in the directory:

```
  2,097,152 image.bin           <--  2MB LPC ROM input image
```

```
C:\>disasm-rom -o MyPrimaryBoot.bin -r primaryBoot TwoMegImage.bin 6.0.1#5

Extracting Primary Boot Image
...
```

Creates a single output file called "MyPrimaryBoot.bin" in the current folder

2.5 rom-format

rom-format is a ROM file format converter. It converts from any format to any format. The input file format is determined automatically. Output format is selected on the command line and can be one of "compressed", "binary" or "s-record". The "compressed" format utilizes a proprietary Transmeta compression algorithm and is not compatible with any other compression standard. The in-system ROM programming utility, "DOSFlash" uses files in the TMTA compressed format. (See *DOSFlash* on page 26.)

rom-format Usage

```
usage: rom-format [flags/switches] <input> <output
 -V        | --version          Print version information and exit
 -f<value> | --format=<value>   Desired output file format. Valid choices are:
                                    compressed, binary, s-record
 -h        | --help              Print usage information and exit
 -v        | --verbose           Print additional information during execution
 <input>                        (REQUIRED) input image file to convert (any
                                    format)
 <output>                       (REQUIRED) converted output image file
```

2.6 make-SPD

make-SPD enables customers to easily create and modify SPD data. SPD stands for "Serial Presence Detect", a 64-byte table containing information about a memory module. Typically this information is available from a serial ROM on the memory module. Systems which do not incorporate memory modules (for board space and cost considerations) must specify this information directly to Code Morphing software for memory configuration.

make-SPD creates an S-record file containing 64 bytes of SPD information. This file may be used inside the OEM-template.rcl file (see page 13) for those systems using soldered down memory (these systems typically do not have an SPD ROM containing SPD data).

To use this utility, copy the file `SPD-template.rcl` from the following directory to your project directory:

```
C:\Program Files\Transmeta\TM8x00 Development Tool Set\TM8x00 System Components
```

Rename the template to accurately reflect the memory device. Then edit the file, changing every instance of `??` to the actual SPD value for the device. Any remaining `??` causes a syntax error from make-SPD. SPD information is available from the memory manufacturer.

Note: change only the text ??; altering any other text in the file can cause a syntax error. For more information about SPD fields, see the SPD specification, and contact the memory manufacturer.

Finally, run make-SPD according to the usage below. This creates a file containing SPD data suitable for use with makerom (see *makerom* on page 9).

make-SPD Usage

```
make-SPD [flags/switches] <SPD>
  -V          --version             Print version info for this utility and exit
  -h          --help                Print usage information and exit
  -o<value>   --output=<value>      Output SPD file name
  -v          --verbose             Print additional information during execution
  <SPD>                             (REQUIRED) input SPD setting file
```

2.7 OEM-template.rcl

This file is a template for OEM-related options. It is used as input to makerom (see *makerom* on page 9). This section describes the template file and how to change it to meet your project needs.

Be sure to use the OEM-template.rcl file that came with your system. Older configuration files may not have the settings you need to properly configure your system. The revision number for the file can be found on the in the fifth line of the file with the tag "Revision:". Refer to this revision when contacting your Transmeta representative. To see an example template file see Appendix B, *OEM-template.rcl* on page 63.

Note
The examples shown in this section are not indicative of correct values—they are for example purposes only. Setting your file to these values will not result in an operable system.

2.7.1 System Specific Information

CPU SKU

In section 1, the CPU SKU defines valid frequency and voltage combinations for your CPU. Normally there is one conservative "debug" or "bring-up" SKU, and one or more "performance" SKUs for production. The SKU definitions in the template file describe the maxiumum frequency and voltage for that SKU.

Uncomment the line in section 1 that matches the SKU for your situation. Transmeta recommends using the bring-up SKU while in development, and then switching to your specific production SKU for final testing and production.

For example, the following SKU definition is for a TM8600 running at 1GHz on a maximum of 7.5 watts:

```
# SKU = "860012.BPLAAA1000"; # TM8600; 1000MHz/7.5W/CR70
```

Change the line to:

```
SKU = "860012.BPLAAA1000"; # TM8600; 1000MHz/7.5W/CR70
```

Voltage Regulator Table

Each voltage regulator has a unique Voltage Regulator Digital to Analog (VRDA) table associated with it. Currently, there is only one voltage regulator approved for use with the Efficeon microprocessor—the Maxim 1718.

Uncomment the line in section 2 that matches the voltage regulator in your system.

For example:

```
# VRDA = "MAX1718";# Currently only MAXIM 1718 is supported
```

Change the line to:

```
VRDA = "MAX1718";# Currently only MAXIM 1718 is supported
```

ROM Selection

The ROM which stores Code Morphing Software determines the capacity for future upgrades. Currently, there are three 2MB ROMs approved for use with the Efficeon processor—the Sharp LH28F016, the SST 49LF080A, and the STMicro M50LPW116. Contact your Transmeta representative to discuss which ROM is most appropriate for your system.

After a ROM has been selected, a virtual ROM model must be entered in the template file. Efficeon processors use a virtual ROM model to perform in-system and secure upgrades. There is a virtual ROM model for each of the supported physical LPC ROMs.

Uncomment the line in section 3 that corresponds to the ROM in your system. For example:

```
# VIRTUAL_ROM = "SHARP_LHF00L01";  # Was called LH28F016 in pre-production
# VIRTUAL_ROM = "SST_49LF080A";
# VIRTUAL_ROM = "STMICRO_M50LPW116";
```

Change the line to:

```
# VIRTUAL_ROM = "SHARP_LHF00L01";  # Was called LH28F016 in pre-production
# VIRTUAL_ROM = "SST_49LF080A";
VIRTUAL_ROM = "STMICRO_M50LPW116";
```

2.7.2 Platform Identifier Fields

Each configuration must include a unique signature consisting of: a customer ID supplied by Transmeta, a number indicating the platform variant, and additional optional information provided by the OEM. This signature is used to ensure that secure upgrades can be safely performed on the OEM system, and that different systems can be uniquely identified.

Transmeta-Provided Customer Identifier and Platform Identifier Fields

Transmeta assignes each customer an identification number in hexadecimal format. Contact your Transmeta representative for this number.

In section 4.1, uncomment the `CustomerID` line and enter your customer ID in place of `<customer-platform-id>`.

For example:

```
# CustomerID        = <customer-platform-id>; # Hexidecimal value in
                                              # the range of 1..0xFFF
```

Change the line to:

```
CustomerID        = 0xD42; # Hexidecimal value in
                                              # the range of 1..0xFFF
```

The platform variant differentiates systems produced by the same OEM. This number is assigned based on how many distinct Efficeon systems are developed by one OEM. Contact your Transmeta representative for this number, which should stay the same for a given project.

Uncomment the `PlatformVariation` number and enter your platform variation in place of `<platform-variation>`. For example:

```
# PlatformVariation = <platform-variation>;    # Hexidecimal value in
                                               # the range of 1..0xFF
```

Change the line to:

```
PlatformVariation = 0x01;    # Hexidecimal value in
                                               # the range of 1..0xFF
```

Customer Platform Identification Field

This optional section enables OEMs to specify unique identifiers. The use of this field is unspecified by Transmeta, and may be used by the OEM in any manner. This field is visible in the PCI configuration space of the nortbridge as a DWORD identifed as bus=0, device=0, func=0, reg=0xC4.

To set a value for this field, in section 4.2 uncomment the line for `upgrade_oem_id1` and replace the 0 with your desired hexadecimal value. For example:

```
# upgrade_oem_id1 => 0;
```

Change the line to:

```
upgrade_oem_id1 => 1234;
```

2.7.3 Upgrade Security Options

Upgrade security options comprise section 5 of the template file.

The LPC flash ROM is protected by x86-visible Northbridge registers (see the section entitled *ROM Write Protection Control* in *Chapter 2 Buses and Connections* in the *Efficeon BIOS Programmer's Guide*). The default state of these registers is to write-protect the Flash ROM.

Efficeon processors support both *secure* and *insecure* upgrades. A secure upgrade is possible only with a signed upgrade image generated by Transmeta. An insecure upgrade can be performed with any ROM image—it is not necessary to obtain Transmeta's assistance to perform insecure upgrades. This mechanism is designed to enable OEMs to easily perform and test upgrades during development, but to turn this capability off for production units entering the field.

This template file provides two fields to control upgrade security, as described below. Transmeta strongly recommends also reading the technical bulletin entitled *ROM Security and BIOS ROM Protection* for further information.

global_write_security

The field `global_write_security` controls write access to the Northbridge field `global_write_enable`. Potential values include:

Value	Description
0	Secure (default value)
1	Hardware dongle required
2	Insecure

bios_write_security

The field `global_write_security` controls controls permission to decrease the protected ROM size (in Northbridge field `rom_protected_size`). Potential values include:

Value	Description
0	BIOS writing is controlled (i.e. BIOS_WE is enabled) (default value)
1	BIOS writing is insecure

Update Security Settings

During development and debugging, provide settings as necessary for your security needs.

Transmeta recomends one of the following configurations for production systems, since these ensure that the Code Morphing software image in the Flash ROM cannot be overwritten:

	bios_write_security	
global_write_security	**0x0 (BIOS_WRITE_CONTROLLED)**	**0x1 (BIOS_INSECURE)**
(ROM_WRITE_SECURE) 0x00	RECOMMENDED	
(ROM_WRITE_HW_DONGLE) 0x01	RECOMMENDED	
(ROM_WRITE_INSECURE) 0x02	UNSAFE	UNSAFE

For example, for a development system:

```
upgrade_options.global_write_security    => 0;  # Default is "rom_write_secure"
upgrade_options.bios_write_security      => 0;  # Default is
                                                # "bios_write_controlled"
```

Change the lines to:

```
upgrade_options.global_write_security    => 2;  # Default is "rom_write_secure"
upgrade_options.bios_write_security      => 1;  # Default is
                                                # "bios_write_controlled"
```

For a production system, change the lines to:

```
upgrade_options.global_write_security    => 0;  # Default is "rom_write_secure"
upgrade_options.bios_write_security      => 0;  # Default is
                                                # "bios_write_controlled"
```

For more information about this field, see the technical bulletin entitled *ROM Security and BIOS ROM Protection.*

Signed Upgrades

Transmeta provides a secure upgrade mechanism (with signed Code Morphing software images) that operates independently of the above protection mechanisms. For more information, see the document *Transmeta Code Morphing Software Upgrade API*.

2.7.4 LongRun Frequencies

Section 6 enables OEMs to specify custom LongRun settings in a given SKU. It is strongly recommended that this field be left at its defaults—the LongRun frequencies provided by Transmeta have been optimized for each SKU. Consultation with a Transmeta representative is required for changing these values.

2.7.5 CPU Features

Various CPU features are available in section 7.

Processor Serial Number

Set `cpu_feature.psn_disable` to 1 to permanently disable the processor serial number. The default is 0, i.e. PSN is enabled. For example:

```
cpu_feature.psn_disable                    => 0;
```

To disable PSN, change the line to:

```
cpu_feature.disable_ecc                    => 1;
```

To leave PSN enabled, do not change the line.

Disable ECC

Set `cpu_feature.disable_ecc` to 1 to treat ECC memory as non-ECC memory. The default is 1, indicating that ECC is disabled. For example:

```
cpu_feature.disable_ecc                    => 1;
```

To enable ECC, change the line to:

```
cpu_feature.disable_ecc                    => 0;
```

To leave ECC disabled, do not change the line. Note that enbling ECC results in a small performance penalty.

Enable SSTL-2 Termination

Set `cpu_feature.SSTL2_termination` to 1 to indicate that SSTL_2 termination is in use for the DRAM bus. The default is 0, indicating that termination is not in use. For example:

```
cpu_feature.SSTL2_termination              => 0;
```

To indicate SSTL_2 termination in the DRAM bus, change the line to:

```
cpu_feature.SSTL2_termination              => 1;
```

Leave the line as it is to indicate that termination is not in use.

NX Support

Set FIXME to 1 to enable support for the No-Execute (NX) command. The default is FIXME. For example:

```

```

Change the line to:

```
```

2.7.6 Memory Configuration

Section 8 provides memory configuration parameters. For DIMMs, the SPD chip on the memory module itself provides all necessary configuration information. For soldered memory, SPD information can be provided by creating a configuration file based on a template.

Other memory parameters can be set individually with the other fields shown in this section. See below for details.

Memory SPD Data for Soldered Down Memory

For memory that is soldered down, OEMs normally need only specify the SPD data using an SPD configuration file. Modifying the provided template file SPD-template.rcl with information provided by the memory vendor, and then use the utility make-SPD (see page 12) to create an SPD s-record file.

Specify this file in OEM-template.rcl by uncommenting the appropriate line(s) that contain the fields MemSPDfile_0 and/or MemSPDfile_1, and specify the absolute pathname to the valid SPD file inside the single quotes on that line.

For example:

```
# MemSPDfile_0 = 'FULL PATH TO VALID SPD FILE (S-RECORD FORMAT)';
```

Change the line to:

```
MemSPDfile_0 = 'C:\my_dev_folder\SPD\my_mem_spd_file.srec';
```

SPD ROM Address

Hexadecimal address of first SPD ROM on SMBUS. Contact your Transmeta representative if you need to change this field. The field is shown below:

```
mem_smbus_spd_base_addr                    => 0x50;
```

Clock Assignments

Table mapping DIMM slot number to clocks for that DIMM. Contact your Transmeta representative if you need to change this field. The field is shown below:

```
mem_slot_to_clocks                         => {0x07, 0x38, 0x00, 0x00};
```

Lowest Memory Frequency

Lower limit of memory frequency in increments of 100/6 MHz. Contact your Transmeta representative if you need to change this field. The field is shown below:

```
mem_freq_min                               => 5;  # 83.33 MHz
```

Highest Memory Frequency

Upper limit of memory frequency in increments of 100/6 MHz. Contact your Transmeta representative if you need to change this field. The field is shown below:

```
mem_freq_max                              => 10; # 166.67 MHz
```

2.7.7 Code Morphing Software Memory Size

This field indicates the size of the block of memory (in MB) reserved for Code Morphing software. Valid values range from 8MB to 128MB, with a recommended absolute minimum of 24MB. The default is 32MB. Contact your Transmeta representative if you need to change this field.

```
cms_memory_size                           =>  32;
```

2.7.8 Debugging Options

Several options are avilable for debugging purposes. Unless you have a board issue you should not need to adjust the defaults. Boot code sets the drive strengths based on the loading it detects. Unless the board layout deviates from the norm as detailed in the *Efficeon System Design Guide* in trace lengths and loading, the settings should not need adjustment. Transmeta highly recommends allowing boot code to do the tuning, and only manually adjusting for known layout or loading deviations.

Debug Port I/O Address

This field enables developers to specify an I/O port address to be used for BIOS (and other) POST codes. Set to a port number, or to 0 to disable.

```
io_port_debug_led                         => 0x0000;  # Default is disabled
```

S-CLK Delays

This field provides a table which maps sclkdly values to maximum memory frequency (in 16.67MHz increments). this table depends on the length of the DDR traces on the board. Longer traces mean that lower values of sclkdly top out at lower frequencies.

```
sclkdly_to_mem_frequency                  => {7, 10, 255, 255};
```

The sclkdly_to_mem_frequency is used to accommodate the board round trip time. In general, the default value of {7,10,255,255} sets the sclkdly to 1 at 167Mhz and should be used for short trace layout. {7,8,255,255} sets the sclkdly to 2 at 167Mhz and should be used for long trace layout.

These only affect the drive strength of the CPU driving the memory. Memory drive strength back into the CPU receiver is not programmable in the same way. There is a DRAM MRS setting for weak drivers, but that is not applicable for any socketed memory where the load may vary.

Minimum and Maximum Loads

These fields provide tables which specifies how many chip loads in each signal group can be driven with the drive strength set to 0 (the minimum) or 7 (the maximum). The minimum load table depends strongly on board capacitance. The maximum load table depends weakly on board capacitance.

`group_to_min_loads`	`=> {0, 0, 0, -1};`
`group_to_max_loads`	`=> {18, 18, 18, 9};`

These 2 group settings are used to control the drive strength for 4 different signal groups, namely CLK, CTRL (CS#, CKE), Address/RAS/CAS/WE, and DQ/DQS in that order. For the above default settings, the 1st three groups are driven with the maximum 18 drivers when the total load is 11 or higher. The last group DQ/DQS is driven with the maximum drivers when the total load is 5 or higher.

It is possible to use linear interpretation to pick the driver vs. load pair. For example, for a given memory module of 1GB on both slots, assume parameters of 32x8-2B per module. Is this correct? If so, the first group (CLK) will have 8 loads, so the number of drivers will be 12. The second and third groups (CTRL and Address/RAS/CAS/WE) will have 32 loads, so the number of driver will be the maximum. The last group (DQ/DQS) will have 4 loads, so the number or driver will be 12—this is because the line has shifted down by 1 and slope change to 9, roughly speaking.

There is normally no need to change these numbers unless developers see problems with DIMMs due to loading. In that case, the memory must be well-characterized to make the right adjustment.

Chapter 3

Bringup Tools

3.1 POSTCodeTool

NOTE: POSTCodeTool is supported only with Code Morphing software 6.0.2 or greater.

A console application used to view Power On Self Test (POST) codes output during the boot process.

postcodetool Usage

```
postcodetool [options] [state...]
  -d<file>   --desc=<file>         select a postcode description file
  -n<value>  --target=<value>       override default target with an
                                    alternate target hostname
  -s<num>    --seconds=<num>        total seconds to check codes
  -x<num>    --exit=<num>          exit code (exit when POST equals this value)
  -V         --verbose              verbose mode---print additional
                                    information when displaying codes
  -h         --help                 Print usage information and exit
  -v         --version              Print version information and exit
```

The -d option maps POST codes to descriptions. Note that many POST codes are vendor-specific—for this reason, you must specify a file that contains a mapping of vendor codes to descriptions. Code Morphing software POST codes are described in Appendix A, *POST Codes* on page 61. Contact your Transmeta representative for further information.

postcodetool Example

```
C:\> postcodetool --seconds=15 --desc=vendor-provided-post-codes
CMSB   0x6802: Completed reading SPD data
CMSB   0x6900: Memory Configuration complete
CMSB   0x5830: Memory Configuration Complete
CMSB   0x5865: Begin XBOOT initialzation
CMSB   0x5880: Begin CMS decompression
CMSB   0x5890: Main CMS successfully decompressed
CMSB   0xACED: CMS loaded, jumping to CMS
CMSN   0xAC04: Begin CMS nucleus initialization for regular cold boot
CMSN   0xAC14: CMS component initialization complete for cold boot
CMSN   0xAC18: x86 install state complete
CMSN   0xAC1C: About to execute first x86 instruction
. . .
```

3.2 TDM2Detect and defaultTDM

TDM2Detect is a Windows application to display all the Transmeta Debug Module (TDM) devices that can be accessed through the local network. defaultTDM.exe is a simple console command which has no parameters.

TDM2Detect does not detect Macraigor Raven devices. For the Macraigor, simply run defaultTDM, or use TDM2Detect to set the default host as “localhost. Also, note that the Macraigor can not communicate with the target unless mac_tdmd is running. For details on the Macraigor device, see *Macraigor Raven* on page 33.

TDM2Detect Usage: Select the TDM-2 in the listbox and click "Select As Default" to set that TDM-2 as the default. This TDM-2 is now the default device for all the tools. Note that in order for TDM2Detect to function on Efficeon processors, the default device must be set to 3 (TCP/IP).

defaultTDM Usage: `defaultTDM` *`target`*

When used with no parameters, defaultTDM sets the default Transmeta Debug Module (Macraigor in this case) to the local machine. In other words, if you connect the Macraigor device to the same machine that you run the tools from, use "defaultTDM" with no parameters. If your Macraigor device is connected to another machine, include the name of that machine on the command line. In this case, the Development Tools must be installed on both the machine connected to the Macraigor device and the machine where the tools will run.

3.3 flashtool

A console application to flash, erase, peek, or poke the ROM using the Transmeta Debug Cable.

flashtool Usage

```
flashtool [OPTIONS] [file]
   -h              | --help                  Print this message.
   -v              | --version               Display version information.
   -e              | --erase                 Erase the entire flash device.
   -p<addr>        | --peek=<addr>           Read the flash location 'addr'.
   -o<addr>,<val>  | --poke=<addr>,<val>     Write the byte value 'val' to flash
                                             location 'addr'.
   -r<addr>        | --peek32=<addr>         Read 32-bit flash loaction 'addr'.
   -s<addr>,<val>  | --poke32=<addr>,<val>   Write the 32-bit 'val' to flash
                                             location 'addr'.
   -i              | --info                  Display information about the flash
                                             device(s).
   -n<name>        | --target=<name>         Override TARGET_TDM environmnt var-
                                             iable and specify alternate target.
   -d[filename]    | --dump[=filename]       Dump flash contents to file.  If
                                             <filename> ends in '.srec', an SREC
                                             formatted file is generated. Other-
                                             wise, a binary file is written. If
                                             <filename> is omitted, the SREC
                                             contents are written to stdout.
   -c              | --checksum              Generate a 32-bit checksum of ROM
                                             contents. Checksum is computed
                                             after all write operations (if any)
                                             are complete.
```

`file` is the name of a ROM image file to be written to the flash. Binary and S-Record formats are supported. S-Record files must use the .srec extension.

Multiple S-Record files can be specified on a single command line, and they will be combined into one image. They are combined by overlaying at the offset addresses embedded in each file.

Note: all flash access commands will reset the target.

flashtool supports flashing through JTAG and the following ROM types:

- SST 49LF080
- Sharp LHF00L01
- ST Micro M50LPW116

3.4 scandump

This tool extracts critical internal state information from the TM8x00 device via JTAG scan chains, and writes this information to a text file which can be used by Transmeta engineers to analyze the state of the processor at the time the scan was performed. This file does not contain any information which is directly usable by a customer—it must be forwarded to TMTA for analysis.

scandump Usage

```
scandump [OPTIONS]
   -h              | --help                Print this message.
   -v              | --version             Display version info.
   -o<filename>    | --output=<filename>   Select output file
   -c<comment>     | --comment=<comment>   Add comment to output file
   -n<name>        | --target=<name>       Override TARGET_TDM environment
                                           variable and specify an alternate
                                            target debug device.
```

When to Use scandump

scandump should be used only when it is necessary to perform a post-mortem analysis with a hung system.

scandump should *not* be used when there is a BSOD, or the mouse is still moving, or the keyboard is still active. The scanning process turns off the processor's internal clocks, hence it is not possible to resume normal operation following a scandump. Since it is only used post-mortem, a hard reset is required anyway.

How to Use scandump

- The --output=*filename* option provides a filename for the results of the JTAG scan. Use descriptive file names, especially when taking multiple scandumps. The –o option can be anywhere in the command line. If you do not use the –o option, the results of the scandump appear on the host console.
- The --comment=*comment* option adds a text comment to the scandump filename (used only in conjunction with the -o option). There can be no spaces in the text comment. Typically this is used to include the Code Morphing software version number in the output. The –c option can be anywhere in the command line.
- scandump --target=*name* overrides the TARGET_TDM variable and selects an alternate target. This is normally not required with the Macraigor Raven debugging device, but it is useful when doing remote scandumps with the TDM-2 device over a network.

Note that the Macraigor device requires that either defaultTDM.exe has been run, or the target must be manually set to “localhost,” as the Macraigor device is not detected automatically with TDM2Detect. For details on the Macraigor device, see *Macraigor Raven* on page 33. For details on the TDM-2 device, see *TDM-2* on page 31.

- Before sending the output file to Transmeta, examine it carefully. The file should contain random characters (not all one character, such as all Fs). If it is all (or mostly) a repetition of one character, then the scandump was not successful.
- When sending a scandump to Transmeta for analysis, you must specify the version of Code Morphing software being used on the platform. The results from scandump cannot be completely analyzed without knowing the Code Morphing software version.
- When sending the file, include any other pertinent information, i.e.
  - Type of system (name of customer platform or reference design)
  - How did the system hang (during boot, BSOD, no response to keyboard and mouse, etc.)?
  - The application being executed (if any)
  - The operating system being used
  - Abnormal thermal activity, i.e. CPU appeared to be very hot
  - Does the hang occur on all systems, or on a subset of all systems?

scandump Examples

Command Line	Result
`scandump`	The target is examined using the default target debugging device. Results are displayed on the host console and not saved to a file.
`scandump –o wizbang0302.txt`	The target is examined and the results are stored in a file named `wizbang0302.txt`
`scandump –o wizbang0302.txt –c cms6.0.0#12`	The target is examined and the results are stored in a file named `wizbang0302.txt`, with the text “cms6.0.0#12” added to the file as a comment
`scandump –n localhost` *`options...`*	The target is examined using the Macraigor debugging device. (This is not necessary as long as you have already run defaultTDM.exe as explained above.)

3.5 TMTASysInfo

This utility extracts system information from the system under debug via the JTAG connector. By default it extracts the electronic serial number from the TM8x00 processor as well as the silicon revision. The "raw" option includes the unprocessed ID scan chains from the processor, the "verbose" option included additional

manufacturing information. If the "cmsversion" option is selected, tmtaSysInfo also attempts to extract version information from the LPC ROM in a fashion similar to print-ver (see *print-ver* on page 9).

TMTASysInfo Usage

```
tmtaSysInfo [OPTIONS]
   -h                  --help                       Print this message.
   -c                  --cmsversion                 Display CMS version
information from ROM
   -r                  --raw                        Display raw scan chains
   -V                  --verbose                    Display more information
   -v                  --version                    Display version info.
   -o<filename>        --output=<filename>          Select output file
   -n<name>            --target=<name>              Override TARGET_TDM
                                                    environment variable and
                                                    specify an alternate
                                                   target TDM-2.
```

TMTASysInfo Examples

```
C:\> tmtaSysInfo
Silicon Revision: 1.2
Electronic Serial Number: G44124-23-14-07
```

```
C:\> tmtaSysInfo --raw
Unique ID: 2000039C17220922
ID Code: 000C24C01189
Silicon Revision: 1.2
Electronic Serial Number: G44124-23-14-07
```

```
C:\> tmtaSysInfo --raw --verbose
System Info: TDM = transmeta-tdm
Unique ID: 2000039C17220922
ID Code: 000C24C01189
Silicon Revision: 1.2
Lot ID: 44124
Wafer ID: 23
Die Coordinates: X=14; Y=7
Fab: G
Foundry: 3
Process: 0
Process Rev: 0
Electronic Serial Number: G44124-23-14-07
```

```
C:\> tmtaSysInfo --cmsversion
Silicon Revision: 1.2
Electronic Serial Number: G44124-23-14-07
Primary Boot Version: "20031217 22:31 official release 6.0.1#5"
Primary CMS Version: "20031217 22:31 official release 6.0.1#5"
Recovery Boot Version: "20031217 22:31 official release 6.0.1#5"
Recovery CMS Version: "20031217 22:31 official release 6.0.1#5"
Primary OEM Configuation Table Version: 4.6; Configured: 2004-01-21 15:58:17
Recovery OEM Configuration Table Version: 4.6; Configured: 2004-01-21 15:58:17
```

3.6 resettool

A console application to perform system reset, CPU reset, jtag reset, or power reset.

resettool Usage

```
resettool [OPTIONS]
OPTIONS are:
  -h           | --help                  Print this message
  -v           | --version               Display version information.
  -n <name>    | --target=<name>         Use TDM named <name>.  Default specified
                                         by $TARGET_TDM environment variable.
```

Note: the remaining options are processed in the order they appear on the command line

```
  -r[<state>] | --sys_reset[=<state>]  Control system-wide reset on target.
                                       <state> is either 0, 1, 3, or 4
                                       (default 4 if omitted).
  -R[<state>] | --cpu_reset[=<state>]  Control CPU reset on target.  <state> is
                                       0, 1, 2, or 3 (default 3 if omitted).
  -j[<state>] | --jtag_reset[=<state>] Control JTAG reset on target. <state> is
                                       0, 1, 2, or 3 (default 3 if omitted).
  -s<action>  | --soft_power=<action>  Press the target system's soft power
                                       button.  <action> is a number with the
                                       following meaning:
                                         0  == turn off
                                         !0 == turn on
  -i[<num>]   | --ignore_pwr[=<num>]   If <num> is 1, ignore whether the target
                                     is powered off when attempting a reset. If
                                       <num> is 0, issue an error if reset is
                                    attempted with a powered off target. (<num>
                                     default is 1 if omitted.) The options list
                                       is treated as though it starts with -i0.
```

The `<state>` numbers mean the following:

0 == release (i.e. tri-state)
1 == assert (i.e. drive low)
2 == deassert (i.e. drive high)
3 == pulse (i.e. assert then release)
4 == release all three reset signals and then pulse system reset

Note that the target system may not always respond as expected to resettool commands involving the soft power button ("-s" option) due to the way the southbridge interprets certain hardware signals from the debugger. Specifically, issuing a "resettool -s0" command immediately after a "resettool -r1" command does not power off the system; the "-r1" command causes the southbridge to think the system is already off.

To return the system to a known state, issue a "resettool -r" command followed by a "resettool -s1" command

3.7 DOSFlash

DOSFlash contains several utilities, as shown in the following sections.

DOSFlash Utility

DOSFlash can read and write the LPC system ROM from DOS in a TM8x00 system. Execution from a Windows command prompt is not supported. However, DOSFlash runs under MS-DOS, PC-DOS, FreeDOS, a Windows 9x bootable floppy disk, or a "Safe mode command prompt only" boot of Windows 9x.

DOSFlash cannot override the ROM security features in the OEM configuration table. Therefore, the field global_write_security must have either the value 01b or the value 10b. If the value is 01b, then the CPU pin L_SECURE must be low when DOSFlash is run. Note that the value 01b is not recommended for production systems. If DOSFlash is unable to run, it prints an error message explaining why.

DOSFlash Usage

```
DOSFlash.exe [OPTIONS]
 -r<value>  | --read-rom=<value>   Dump the ROM contents to the specified file
 -w<value>  | --write-rom=<value>  Flash the ROM using the specified image file
 -p<value>  | --params=<value>     Use the ROM device model in the provided file
                                    instead of the one specified in the existing
                                    OEM config table
 -h         | --help               Print usage information and exit
 -v         | --version            Print version information and exit
```

Either the `--read-rom` or `--write-rom` option must be specified. If both are specified, DOSFlash reads the current ROM contents before writing the new image file. The ROM image files read and written by DOSFlash are compressed using a proprietary Transmeta algorithm.

When writing the ROM, DOSFlash uses the ROM device model specified in the upgrade_virtual_rom_model field of the OEM configuration table. If that ROM device model is incorrect, then the flash operation fails. This indicates a problem with the OEM configuration table that will prevent secure field upgrades from working. The table should be corrected at the earliest opportunity. DOSFlash can work around this problem by using a separate file that contains a complete ROM device model. That file is specified on the command line with the `--params` option.

make-flash-floppy Utility

The utility `make-flash-floppy` creates a bootable DOS floppy disk containing DOSFlash and ROM image file. The ROM image file must be provided on the command line.

make-flash-floppy Usage

```
usage: make-flash-floppy [options/switches] <image>
  <image>          (REQUIRED) ROM image file to place on floppy disk.
  -h | --help      Print usage information and exit.
  -v | --version   Print version information and exit.
```

The floppy disk is created using a separate utility called `rawrite` and a floppy image file. The floppy image file, `rawrite`, and `DOSFlash` must all be present in the same directory as the `make-flash-floppy` utility.

Chapter 4

Debugging Hardware Interfaces and Tools

Note
The next chapter describes VICE, the Transmeta in-circuit emulator/debugger. See *Virtual x86 In-Circuit Emulator (VICE)* on page 35.

4.1 System Debug Interface

The *Efficeon™ TM8x00 System Design Guide* describes the recommended JTAG interface debug connector for TM8x00 processor-based system designs.

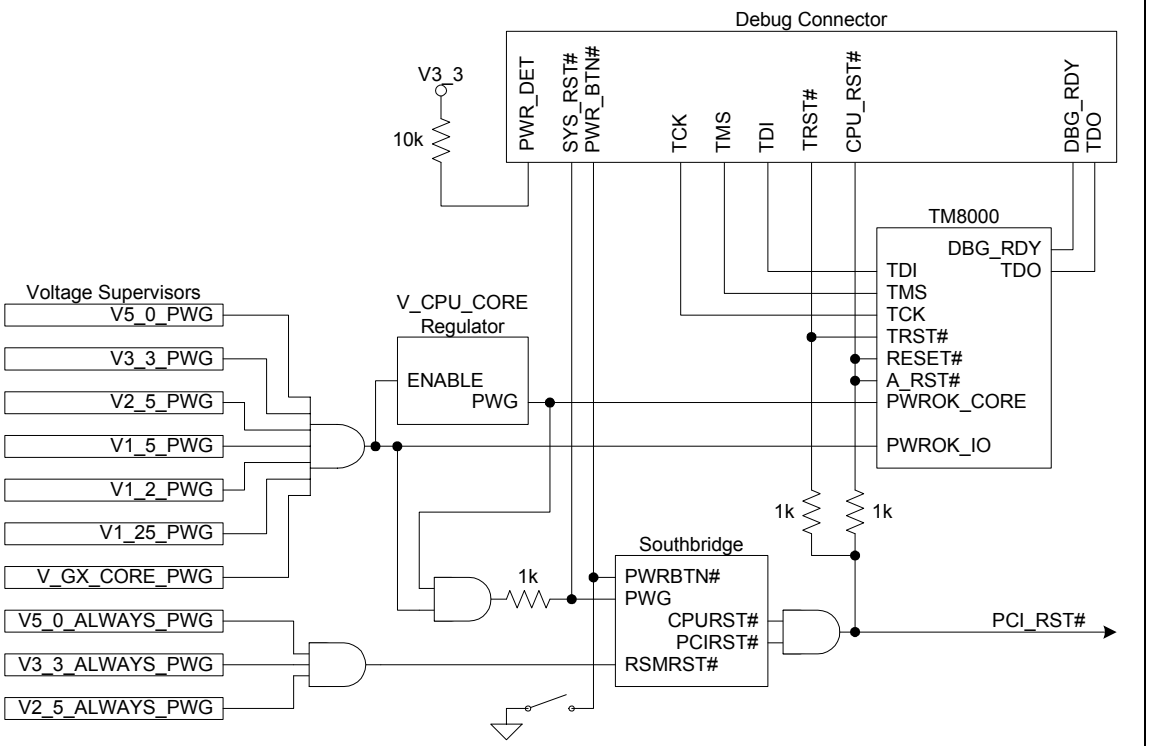
The system debug interface uses the TM8x00 processor 16-pin JTAG interface, of which a subset is the standard 5-pin JTAG interface. This interface supports IEEE 1149.1, EXTEST, SAMPLE/PRELOAD, BYPASS, and HiZ instructions. The processor JTAG interface operates up to 100 MHz. The JTAG interface debug connector pinout is described in the table below.

TM8x00 System Debug Connector		
Pin Number	**Signal Name**	**Description**
1	SYS_RST#	System reset signal, active low.
2	CPU_RST#	TM8x00 processor RESET# signal, active low.
3	GND	System ground.
4	DBG_RDY	TM8x00 processor DBG_RDY (debug ready) signal.
5	GND	System ground.
6	TDO	TM8x00 processor JTAG interface TDO (data output) signal.
7	GND	System ground.
8	TCK	TM8x00 processor JTAG interface TCK (clock input) signal.
9	GND	System ground.
10	TMS	TM8x00 processor JTAG interface TMS (test mode select) signal.
11	GND	System ground.
12	TDI	TM8x00 processor JTAG interface TDI (data input) signal.

TM8x00 System Debug Connector (Continued)		
Pin Number	**Signal Name**	**Description**
13	GND	System ground.
14	TRST#	TM8x00 processor JTAG interface TRST# (reset) signal, active low.
15	PWR_BTN#	System power button signal, active low.
16	PWR_DET	System power detect signal.

The figure below shows an example TM8x00 system debug interconnection diagram. The debug connector is shown routed to the required signals on the system motherboard.

Example TM8x00 System Debug Interconnection Diagram



See the *Efficeon™ TM8x00 Data Book* and *Efficeon™ TM8x00 System Design Guide* for more details on the TM8x00 processor JTAG interface and the recommended system debug connector.

4.2 Debugging Interfaces

The minimum system requirement for the Efficeon Configuration Tools is a 500MHz Pentium-III or equivalent, with at least 512 MB RAM. Windows 2000/SP3 and XP are supported; Windows 9x and Windows Me are not supported. If your system has no parallel port, a PCMCIA parallel port adapter such as the Qualtech SPP-100 may be used.

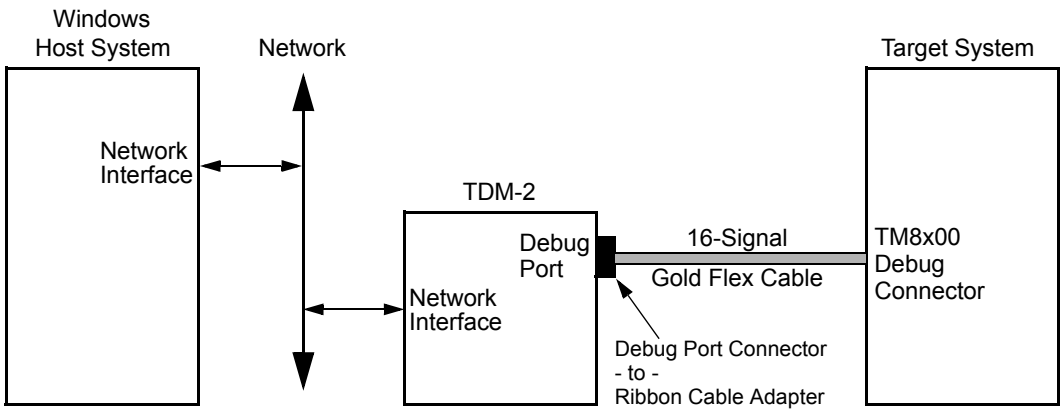
There are two supported debugging interfaces:

- TDM-2, the Transmeta Debug Module version 2
- Macraigor Raven

TDM-2

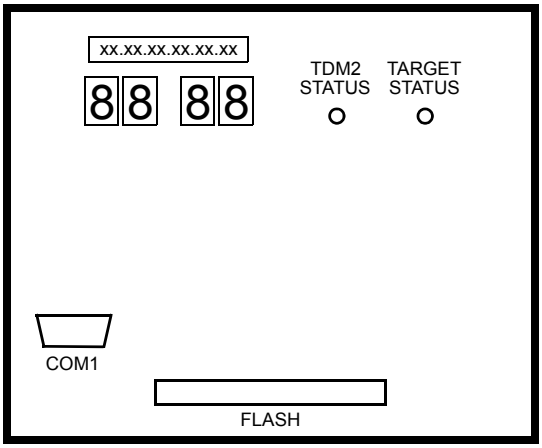
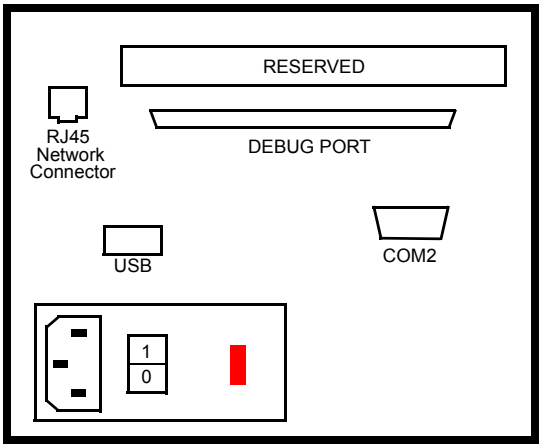
The Transmeta debug module-2 (TDM-2) provides the host system with access to the target system through the TM8x00 debug interface. The TM8x00 software tools require either the TDM-2 or Macraigor Raven (see later section on Macraigor Raven) in order to access and control the target system. The TDM-2 is a hardware device that connects between the network and target system as shown in the diagram below. The host uses the TDM-2 through the network connection to access the target system.

Connecting the TDM-2 to the Host and Target Systems



The TDM-2 front and rear panel displays and connectors are shown in the figures below.

Transmeta TDM-2 Displays and Connectors

Front Panel	Rear Panel
	
xx.xx.xx.xx.xx.xx (12 character label)	RJ45 network connector
TDM-2 network adapter MAC address	Reserved connector

Transmeta TDM-2 Displays and Connectors (Continued)	
Front Panel	**Rear Panel**
LED POST code displays	Debug port connector
TDM2 status LED	USB port connector
Target status LED	COM2 serial port connector
COM1 serial port connector	AC power cable connector
Compact flash memory socket	AC power switch and red LED power indicator

Connecting and Configuring the TDM-2

The procedure below describes how to connect the TDM-2 to the network and the target system, and how to set up the host system to access the target system and TDM-2 through the TM8x00 tools.

NOTE

The network to which the host computer and TDM-2 are connected must have a DHCP server (which automatically assigns an IP address to the TDM-2 and maps the TDM-2 MAC address to this assigned IP address) to use the configuration procedure described here. If there is no DHCP server on the network, another method must be used for assigning and determining the IP address of the TDM-2.

If `tdm2detect` cannot be used and no default TDM-2 is assigned for the host system, the TM8x00 tools require the target TDM-2 IP address to be explicitly called out on the command line.

1. Connect the power cord to the TDM-2 (rear panel) and plug the other end of the power cord into a standard AC outlet.

2. Plug a standard RJ45 network cable into the TDM-2 network connector socket (rear panel) and plug the other end of the network cable into a live network connection.

3. Turn on the TDM-2 using the power switch on the rear panel. Wait until the front panel display changes from ‘88.88’ and the ‘TDM-2 Status’ light comes on. The TDM-2 requires less than a minute to complete it’s power-on diagnostics, boot-up, and network configuration.

4. Make sure the host computer is connected to the same network as the TDM-2. Then run the Transmeta `tdm2detect` utility on the host computer:

   a. Open the `tdm2detect` Windows application from the host computer desktop.

   b. `tdm2detect` will display a list of TDM-2 devices it detects on the network.

   c. In the list of TDM-2 devices displayed by `tdm2detect`, look for the TDM-2 that has the same MAC address as the TDM-2 you want to use. Note that the TDM-2 MAC address can be found on a label on the front panel of the TDM-2 (see front panel figure above).

   d. In the `tdm2detect` window, highlight the TDM-2 you wish to use (based on the MAC address), and click ‘Select As Default’ to set the selected TDM-2 as the default TDM-2 for the host system. This TDM-2 is now the default TDM-2 for all the TM8x00 tools. `tdm2detect` also shows the IP address assigned to the TDM-2.

5. Connect the TDM-2 flex cable to the Transmeta TDM-2 debug port connector-to-flex cable adapter provided with the TDM-2.

6. Plug the debug port connector-to-flex cable adapter into the TDM-2 rear-panel debug port connector.

7. Connect the free end of the TDM-2 flex cable to the target system TM8x00 debug connector.

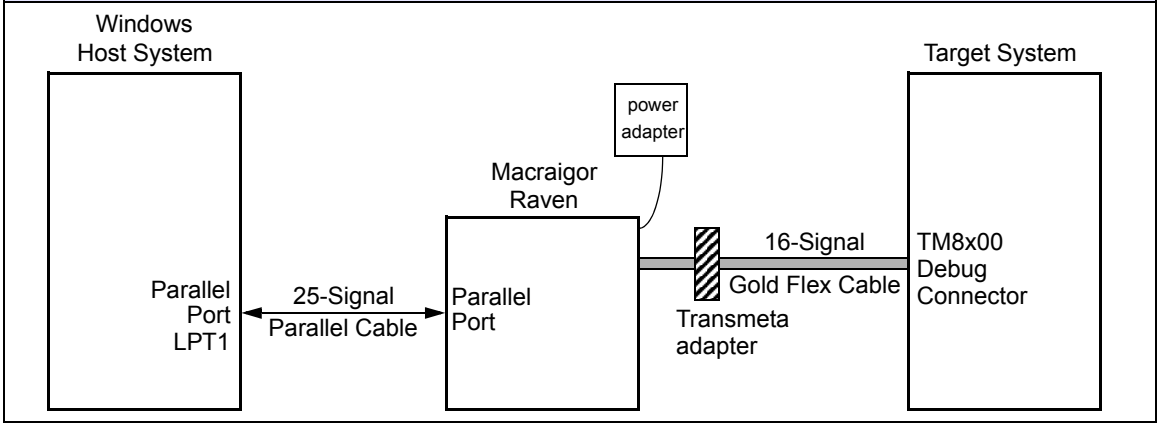
8. Power on the target system. The host, target, and TDM-2 are now ready for use.

Macraigor Raven

The Macraigor Raven provides the host system with access to the target system through the TM8x00 debug interface. The TM8x00 software tools require either the Macraigor Raven or TDM-2 (see prior section on TDM-2) in order to access and control the target system.The Macraigor Raven is a lower-cost (than TDM-2) hardware device that connects between the host and target systems, as shown in the diagram below. The host uses the Macraigor Raven through a parallel port connection to access the target system.

Connecting and Configuring the Macraigor Raven

Connecting the Macraigor Raven to the Host and Target Systems



1. Ensure that the host system parallel port is set to EPP mode.

   Failure to set the parallel port to “EPP” mode may result in JTAG scan failures. The parallel port is configured from your BIOS setup screen. If the BIOS allows you to configure more than one parallel port, configure LPT1.

2. Connect all Raven hardware as follows:

   a. Power down your target TM8x00 system.
   b. Connect the Transmeta adapter to the Macraigor Raven.
   c. Connect the other end of the adapter to the target system using the gold flex cable provided.
   d. Connect the parallel port cable to the host computer. If your system has multiple parallel ports, you must connect to LPT1.
   e. Connect the other end of the parallel port cable to the Macraigor Raven.
   f. Connect the Macraigor power adapter to the Raven.
   g. Plug in the Macraigor power adapter.

3. Install the TM8x00 Development Tool Kit.

   Follow all instructions carefully. Note, the Efficeon Tools installer installs the mac_tdmd server as a Windows service. mac_tdmd must be running in order for any of the tools to operate correctly. The installer will ask you if you'd like to start the mac_tdmd service automatically. If you select "Yes", you are done. If you select "No", you must manually start the service as described in the next step.

   Reboot the host computer when complete.

4. If you chose not to start the mac_tdmd service automatically in the previous step, you must start it manually. To manually start the mac_tdmd service, please perform the following steps:

   a. Go to Control Panels->Administrative Tools and double-click on "Services"
   b. Double-click on "Transmeta DevTools Server (for Macraigor)"
   c. From the "General" tab, click on the Start button to start, Stop button to stop.
   d. If you want mac_tdmd.exe to automatically restart if it crashes, go to the Recovery tab and select "Restart the Service" in the appropriate drop down windows.

   To view any error/warning output:

   a. Go to Control Panels->Administrative Tools and double-click on "Event Viewer"
   b. Under "Event Viewer (Local)" click on Application
   c. In the "Application" window, if there are any output message, you can get more details by double-clicking on any message.

This completes the installation procedure

Note for TDM-2 Users
If you have previously installed and used the TM8x00 Development Tools with a TDM2 unit supplied by Transmeta, you may need to reconfigure your default TDM settings. The default TDM is set using the TDM2Detect utility. Start the TDM2Detect program and manually set the default target to “localhost”. TDM2Detect does NOT detect Macraigor Raven devices, so all entries must be done manually. If you have never run TDM2Detect, no reconfiguration is necessary. The installer automatically sets the default target TDM to “localhost”. For more information on TDM2Detect, see *TDM2Detect and defaultTDM* on page 22.

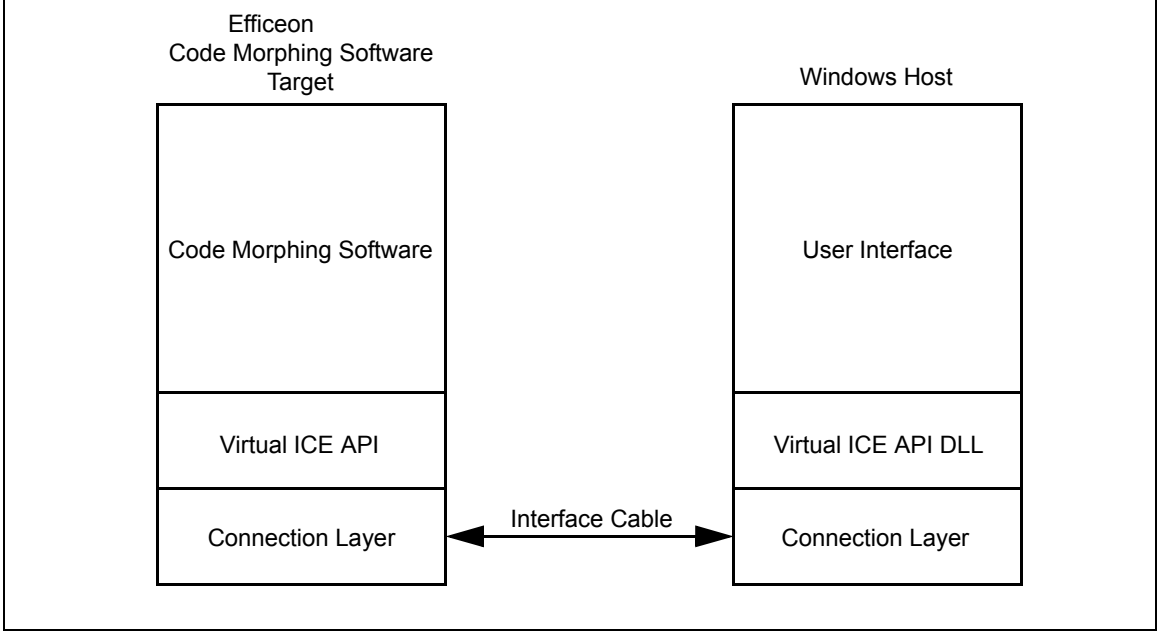
Chapter 5

Virtual x86 In-Circuit Emulator (VICE)

`vice` is a Windows-based virtual x86 in-circuit emulator for the Efficeon processor. `vice` is a powerful debugging tool used during bringup to assist in debugging system designs and BIOS development. `vice` uses a traditional host/target emulator model, the target being the Code Morphing software component of the Efficeon processor environment, which provides a set of debug operations and a virtual in-circuit emulator software interface (API). The Windows host system provides a user interface on a machine independent from the target system. By utilizing a separate host system, instabilities in the target system being debugged do not interfere with the debugger software running on the host.

Code Morphing software includes integrated standard in-circuit emulator functionality, together with some powerful extended capabilities. The `vice` host user interface Windows application communicates with `vice` through a Windows DLL. The `vice` API DLL allows direct program control of `vice`. The connection layer provides the hardware mechanism (JTAG interface) for the host to communicate with the target. The figure below shows the software architecture of the Efficeon virtual in-circuit emulator.

Efficeon Virtual In-Circuit Emulator Software Architecture



5.1 Feature Overview

General

Debugging can begin before the first x86 instruction is executed.

A hard CPU reset can be issued to reset/reboot the system without losing debugging context.

The host can halt the target at any time and provide a consistent x86 state.

At any time, the host can force a non-maskable system reboot as though the reset button had been pressed.

Registers

Supports reading and writing all x86 registers, including MSRs, MMXs, XMMs, and MTRRs, and CPUID values.

Memory

There are certain necessary features related to memory contents. Memory can be examined, set, filled, tested, dumped. Memory access can be done in bytes, words, or dwords. Memory addressing can be physical, linear, or logical, with a SMM enable/disable qualifier.

Memory addresses can be described in a *selector:offset* form where the offset is an expression that can include labels, registers, and constants. The ICE performs paging translations if enabled, or accepts the address in the physical space.

I/O Ports

The ICE supports reading and writing arbitrary data to/from the I/O during a debug session. This is done without perturbing the x86 processor state. I/O references can be bytes, words, or dwords.

Configuration Registers

A "scan" of the HT bus can be done by the ICE to inquire the device configuration on the bus. This is done without any x86 instructions executed.

All device configuration registers can be read/written, including registers in the virtual northbridge (VNB).

Breakpoints

Stopping the processor can be done with breakpoints or by use of the **halt** command. Breakpoints can be set in the following circumstances:

- The x86 IP hits a particular valid address. Read and/or write to a particular memory address, I/O port, or interrupt. A maximum of four instruction breakpoints can be active at one time.
- The processor is reset. When enabled, every processor reset or power-up generates this breakpoint. This breakpoint is enabled by default when the ICE is started.

After hitting a breakpoint or halting the processor, the user can:.

- Step an instructions (or 1 repetition through a LOOPx or REP prefixed instruction), a source line, into a routine call, or into an interrupt service routine.
- Continue execution.
- Issue other ICE commands

System Management Mode

This ICE is operational during SMM, and can access the SMM memory, when not in SMM by adding a type cast to the memory address. Breakpoints can be set on the SMM handler.

5.1.1 Operational Notes

The Virtual x86 ICE, like all debuggers and ICE's, may change the behavior of the system, or have side effects. The following is a list of things to keep in mind when using the Virtual ICE, but this not necessarily a complete list.

- The target is halted at every reset to allow the host to down load the list of current breakpoints, even if the reset breakpoint interrupt is disabled.
- Normally DMA requests are serviced when halted in the ICE, and during host to target communications. The exception to this is at the reset, prior to the executing the first x86 instruction DMA requests are ignored.
- Interrupt requests are ignored when halted in the ICE, and during host to target communications.
- The ICE uses the debug registers for x86 breakpoints.
- There are real time affects to the ICE being attached.
- The parallel port interface to the ICE can affect power management, and be affected by x86 programs writing to the assigned port.
- The PCI commands perform I/O cycles to the PCI configuration registers. There is the side effect of changing the PCI configuration address register when issuing these commands.
- Setting invalid combinations of CR0 and other registers cause the ICE and processor to hang.

5.1.2 Known Issues

The ICE uses the debug registers for execution breakpoints, but currently doesn't protect them from x86 accesses. Programs can actually disable the ICE breakpoints without the ICE knowing about it.

5.2 Installation

5.2.1 System Requirements

Minimum host system requirements are at least a 500MHz Pentium III or equivalent with at least 512MB of memory. Windows 2000/SP3 and XP are supported; Windows 9x and Windows Me are not supported. If your system has no parallel port, a PCMCIA parallel port adapter such as the Qualtech SPP-100 may be used.

5.2.2 Software Installation

The VICE software is installed as part of the Efficeon™ Development Tool Set.

The default installation location is: C:\Program Files\Transmeta\TM8x00 Development Tool Set

When installation is complete, you can start VICE using the desktop icon or by selecting VICE from the Transmeta sub-folder in the Programs folder under your Start menu.

5.2.3 Host to Target Connection

The debugging device is connected to the host via a network cable. VICE uses Windows socket APIs to communicate with the debugging device.

TDM-2 devices can be connected to the same LAN as the host system, or it can be directly connected to the host via a cross-over cable or a stand-alone hub. In all cases, the operating system MUST be set to "obtain an IP address automatically". For Macraigor Raven devices to connect, mac_tdmd must be running, and no cross-over cable is used. For connection details, see *Debugging Interfaces* on page 30.

TDM2detect can be used to detect all debugging devices accessible from the host system—see *TDM2Detect and defaultTDM* on page 22). One of the features of the TDM2detect is to set the default debugging device to which VICE should connect. For Efficeon processors, this default value must be set to 3 (TCP/IP). Upon starting up, VICE uses the following algorithm to locate the device to which it should connect:

- If VICE can only detect one debugging device, VICE connects to it.
- If VICE detects more than one debugging device, the default device (set by TDM2detect) is used.
- Otherwise, an error is displayed.

5.3 Text Command Interface

The ICE is started by executing the program `vice.exe`. Optionally, a file containing VICE commands to execute prior to prompting the user can be provided on the command line.

Users can also optionally specify the Code Morphing software version on the command line as shown below. This setting overrides the default provided with VICE.

```
usage: vice [options/switches] command_file
```

Option		Description
`-c`*value*	`--cms=`*value*	Specify the Code Morphing software version on the command line. This setting overrides the default provided with VICE. For example, if you have two target systems with different versions of Code Morphing software, you can use the same installation of VICE by providing the Code Morphing software version. It is normally recommended to simply use the version of VICE that is provided with Code Morphing software. *Example:* `vice -cms 6.0.0#12`
`-n`*value*	`--target=`*value*	Specify the target TDM. This setting overrides the default setting.
`-h`	`--help`	Display VICE help message.
`-v`	`--version`	Display VICE version.

Upon exit, the return code from vice is dependent upon the value specified on the `quit` command. If no value is given with the `quit` command, the return code is 0.

5.3.1 Command Prompt

The prompt displayed shows the last known status of the CPU:

Prompt	Meaning
`ICE:`	The CPU is currently executing instructions. All commands are still active, but CPU execution is halted when a command requires interaction with the target to perform the command. After the command completes CPU execution is resumed, except for the **halt** command.
`ICE>`	The CPU is stopped waiting for user input.
`ICE?`	The CPU run/stop state is unknown by the ICE. This happens when the host and target are not physically or logically connected. (Cable disconnected, or the CPU is in a power-management suspend/sleep state).

Entering a command or just hitting the return key at the command prompt causes the ICE to query the CPU for its current running state.

5.3.2 Commands

The commands are loosely based on the other common DOS debugger command sets, with additions provide to give a richer debugging control. Commands are entered at the prompt, and executed when

RETURN is entered. Multiple commands can be placed on one line if separated by semi-colons. Commands can be grouped together into a compound command by using braces {}.

See *Numeric Values* on page 53 and *Addresses* on page 56 for input and display formats.

5.3.2.1 Assembly/Disassembly

a—Assembly

Assemble code into memory. ICE displays the address and waits for the user to enter assembly mnemonics.

Syntax	`a [address_range]`
Example	`ICE> a cs:ip`

u—Unassemble

Disassemble instructions at the specified address range, using the current CPU state. If no address range is given, disassembly continues from the previous command that generated disassembly output (this includes a register dump). If the length is given as part of the address range, it specifies the number of instructions to be disassembled. If no length or ending address is provided, eight instructions are disassembled.

Syntax	`u [address_range]`

Example	`ICE> u cs:ip`
	`07C0:0000  4A                    dec         dx`
	`07C0:0001  6F                    outsw`
	`07C0:0002  72 64                 jc          0068h`
	`07C0:0004  61                    popa`
	`07C0:0005  6E                    outsb`

The current setting of the D/B flag in the CS attributes is used to determine whether the code should be disassembled as 16-bit or 32-bit instructions. This can be overridden by casting the starting address with a `use16` or `use32` (see *Type Casts* on page 55.)

5.3.2.2 Breakpoints

The breakpoint commands allow the user to create and control breakpoints. As breakpoints are created they are assigned a monotonically increasing breakpoint numbers starting at one, with a maximum of 32,767 breakpoint numbers. The numbering is reset to one only if the command `bc *` is issued.

bpx—Set Instruction Breakpoint

Create a new breakpoint at the instruction address. The supplied address must be the first byte of the instruction. Any number of instruction breakpoints can be created, but only four instruction breakpoints can be enabled at one time.

Syntax	`bpx address [if (conditional_expression)]`
Example	`ICE> bp 7c00:0` `ICE> bp f000:e05b`
Abbreviation	`bp`

Note: The instruction breakpoints use the CPU debug registers, and are subject to the limitations of instruction breakpoints imposed by those registers (i.e. Internally the address is converted to a linear address).

bpr, bpw, bprw—Set Memory Read/Write Breakpoint

Create a new breakpoint at the read and/or write of the memory address.

Syntax	`bpr address [if (conditional_expression)]`
	`bpw address [if (conditional_expression)]`
	`bprw address [if (conditional_expression)]`

Example	`ICE > bpr 2345:0`

bpio—Set Breakpoint on I/O Location

Create a breakpoint on an I/O location.

Syntax	`bpio port [if (conditional_expression)]`
Example	`ICE> bpio 5`

bpint—Set Interrupt Breakpoint (reset, icebp, int3, num)

Create a breakpoint on one of the following conditions: CPU reset, init, or shutdown; the `icebp` instruction; intercept the `int3` instruction; or on a specific interrupt number. The `reset` breakpoint is enabled by default when the ICE is started, but can be disabled or cleared like any other breakpoint.

Syntax	`bpint reset`
	`bpint icebp [if (conditional_expression)]`
	`bpint int3 [if (conditional_expression)]`
	`bpint num [if (conditional_expression)]`

Example	`ICE> bpint reset` `ICE> bpint int3 if(edx == 5)`

bl—List Breakpoints

List one or more breakpoints. The breakpoint number, type, address and if they are currently disabled is displayed. A `*` next to the breakpoint number denotes the most recently hit breakpoint. Breakpoint types are `exe` for instruction execution, and `reset` for CPU reset/init. Reset breakpoints have no associated address.

Syntax	`bl [list|*]`

Example	`ICE> bl`
	`01 - reset`
	`02*- exe 7C00:0000`
	`03 - int3 if(…) hit 0 times`
	`03 - exe F000:E05B disabled`

Note: A list specifier of `*` indicates all breakpoints.

bd—Disable Breakpoints

Disable one or more breakpoints. This allows you to have the CPU not trigger a given breakpoint, but maintain the breakpoint information for later use. The `be` command can restore the breakpoint functionality.

Syntax	`bd` *list*`|*`

Example	`ICE> bd 1,3`
	`ICE> bl`
	`01 - reset disabled`
	`02 - exe 7C00:0000`
	`03 - exe F000:E05B disabled`

Note: A list specifier of `*` indicates all breakpoints.

be—Enable Breakpoints

Make one or more breakpoints functional, allowing the CPU to trigger the beakpoint if hit again.

Syntax	`be` *list*`|*`

Example	`ICE> be 1`
	`ICE> bl`
	`01 - reset`
	`02 - exe 7C00:0000`
	`03 - exe F000:E05B disabled`

Note: A list specifier of `*` indicates all breakpoints.

bc—Clear Breakpoints

Disable and delete one or more breakpoints. If `*` is used as a breakpoint number list, the breakpoint numbering sequence is reset to start at 1. Once a breakpoint is cleared it cannot be re-enabled, but its breakpoint number is not reused unless the command `bc *` is issued, clearing all breakpoints.

Syntax	`bc` *list*`|*`

Example	`ICE> bc 2`
	`ICE> bl`
	`01 - reset`
	`03 - exe F000:E05B disabled`

Note: A list specifier of `*` indicates all breakpoints.

5.3.2.3 Memory/Data Operations

These commands allow the user to display and modify sections of memory, I/O or PCI space. The default address type is `memory`, but can be overridden with `IO` or `PCI` type casts (see *Type Casts* on page 55).

dump, db, dw, dd—Dump Data

Display the contents of a range of addresses in the specified address space. If no address range is supplied the defaults to a continuation of the previous dump command. If an address is supplied, but no length or ending address is supplied as part of the range, the range is assumed to be 128 bytes.

Syntax	`dump [address_range]` `d[b|w|d] [address_range]`

Example	`ICE> dw 40:0 l 30`
	`07C0:0000 : 6F4A 6472 6E61 2F20 4D20 6369 6168 6C65`
	`07C0:0010 : 2D20 4120 7475 6F68 7372 0000 0000 0000`
	`07C0:0020 : 0000 0000 0000 0000 0000 0000 0000 0000`
	
	`ICE> d (io) 20`
	`0020: 00 B8 FF FF 00 08 FF FF : 00 08 FF FF 00 08 FF FF`

Abbreviation	`d`

The `b`, `w`, or `d` suffix on the command sets the default data reference size of byte, word, or dword, respectively. Bytes are assumed if no suffix is given, or the address does not contain a size cast.

The address is displayed as address type that was specified (`logical`, `linear` or `physical`), with a trailing `l` or `p` for linear and physical (respectively). Data is displayed as a string of digits with separators to make them more readable. For byte-wide memory data, the ASCII characters are displayed at the end of each line with periods for non-printable characters.

enter, eb, ew, ed—Enter Data

Write a list of data values, starting at the given address. The data values are entered into the target addresses from left to right.

Syntax	`enter address [=] expression_list` `e[b|w|d] address [=] expression_list`
Example	`ICE> ew 40:0 03F8 02F8`
Abbreviation	`e`

The `b`, `w`, or `d` suffix on the command sets the default data reference size of byte, word, or dword, respectively. Bytes are assumed if no suffix is given, or the address does not contain a size cast.

fill, fb, fw, fd—Fill Data

Fill an address range with a value or a repeating set of values. The expression list is used in a left to right order repeating until the address range is completely filled.

Syntax	`fill address_range [=] expression_list` `f[b|w|d] address_range [=] expression_list`
Example	`ICE> fw b800:0 l 1000 = 0720`
Abbreviation	`f`

The `b`, `w`, or `d` suffix on the command sets the default data reference size of byte, word, or dword, respectively. Bytes are assumed if no suffix is given, or the address does not contain a size cast.

5.3.2.4 TCP/IP Support

attach—Connect to a host

Connect to host *host_name*.

Syntax	`attach` *host_name*

Example	`ICE> attach`
	`ICE> attach mike_tdm2`
	`ICE> attach 23.98.98.99`

connect—Connect to a host and issue hard reset

Connect to host *host_name* and issue a “reset hard” command (see *reset—Reset the CPU* on page 49).

Syntax	`connect` *host_name*

Example	`ICE> connect`
	`ICE> connect mike_tdm2`
	`ICE> connect 23.98.98.99`

disconnect—Disconnect from a host

Disconnect from the currently connected host. If `continue` is specified, a "go" command (see *go—Go Command* on page 47) is issued before disconnecting.

Syntax	`disconnect [continue]`
Example	`ICE> disconnect` `ICE> disconnect continue`

5.3.2.5 I/O Logging Support

iolog—I/O Logging Support

Add, delete, list, or dump contents of the I/O log.

Syntax	`iolog add` *port*
	`iolog delete` *port*
	`iolog list`
	`iolog dump`

Example	`ICE> iolog add 4` `ICE> iolog add 1`

The I/O log is stored in a limited buffer inside the processor. A processor reset clears the buffer contents. There are commands to start logging, stop, remove, list the logged ports, and dump the buffer contents.

5.3.2.6 I/O Operations

in, ib, iw, id—Input Command

Read from a device I/O port and display the value.

Syntax	`in` *address* `i[b|w|d]` *address*
Example	`ICE> i 21` `B8`
Abbreviation	`i`

The `b`, `w`, or `d` suffix on the command sets the default data reference size of byte, word, or dword, respectively. Bytes are assumed if no suffix is given, or the address does not contain a size cast.

out, ob, ow, od—Output Command

Write to a device I/O port and display the value.

Syntax	`out` *address* `[=]` *value* `o[b|w|d]` *address* `[=]` *value*
Example	`ICE> o 21 B9`
Abbreviation	`o`

The `b`, `w`, or `d` suffix on the command sets the default data reference size of byte, word, or dword, respectively. Bytes are assumed if no suffix is given, or the address does not contain a size cast.

5.3.2.7 Register Command

register—Register Read/Write

Read CPU registers, or write a single CPU register.

Syntax	`register [`*reg*`] [[=]` *expression*`]`

Example	`ICE> r`
	`EAX=F00004E3  EBX=00126080  ECX=12BC51C8  EDX=6CD60040`
	`ESP=001231C8  EBP=0012325C  ESI=E0ED0000  EDI=000E3264`
	`EIP=0001515A  Flags=3A96`
	`CS=00A7  DS=00AF  SS=00AF  ES=00AF  FS=00CF  GS=00CF`
	`00A7:1515A  25 FF 00  and  ax, 00FFh`
	
	`ICE> r ax`
	`AX=04E3`

Abbreviation	`r`

If the register is not specified, the set of common integer registers, flags and a disassembly of the instruction at the current CS:EIP is displayed. If a register is specified, that register is displayed (if no expression is supplied) or modified (if an expression value is supplied).

5.3.2.8 MSR Command

msr—MSR Read/Write

Read or write the CPU model specific register. See the CPU specification for a list of valid registers and their definition. Referencing invalid MSRs returns an error, but does not cause the CPU to fault.

Syntax	`msr` *reg* `[[=]` *value*`]`
Example	`ICE> msr 80860000` `0000000000000542`

5.3.2.9 CPUID Command

cpuid—CPUID View

Display the CPUID information for the specified index. Referencing invalid CPUID indexes returns an error, but does not cause the CPU to fault.

Syntax	`cpuid` *index*

Example	`ICE> cpuid 0`
	`EAX=00000003 ....`
	`EBX=756E6547 Genu`
	`ECX=3638784D Mx86`
	`EDX=54656E69 ineT`
	`ICE> cpuid 3`
	`EAX=00000000 ....`
	`EBX=00129189 ëæ..`
	`ECX=00000000 ....`
	`EDX=00000000 ....`

5.3.2.10 PCI Commands

pci scan—Scan PCI Bus for Devices

Scan the PCI bus and display the PCI bus, device number, vendor id and device id for all the devices found in the system.

Syntax	`pci scan`

Example	`ICE> pci scan`
	`Bus 00 Device 00 - 1279 0295`
	`Bus 00 Device 07 - 8086 7110`
	`Bus 00 Device 0A - 5333 8A01`

pci, pcib, pciw, pcid—Display or Modify PCI Configuration Registers

Display or modify PCI configuration registers in the specified PCI bus/device/function.

Syntax	`pci[b|w|d] [[[[`*bus*`:]`*device*`:]`*function*`:]`*reg* `[`*length number*`]]` `pci[b|w|d] [[[[`*bus*`:]`*device*`:]`*function*`:]`*reg* `[[=]` *expression_list* `]]`

Example	`ICE> pci 7:0:0 l 20`
	`00:07:00:00  86 80 10 71 0F 00 80 02-01 00 01 06 00 00 80 00`
	`00:07:00:10  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00`

For fields not specified the previously specified values are used. The address is displayed a bus, device, function and register separated by colons. The data is displayed in the same format as the `dump` command (see *dump, db, dw, dd—Dump Data* on page 43).

The `b`, `w`, or `d` suffix on the command sets the default data reference size of byte, word, or dword, respectively. Bytes are assumed if no suffix is given, or the address does not contain a size cast.

pci info—Display ICE PCI Information

Display the current default PCI bus, device and function used by the `pci` command (see *pci, pcib, pciw, pcid—Display or Modify PCI Configuration Registers* on page 47).

Syntax	`pci info`
Example	`ICE> pci info` `Current Bus=00 Device=07 Function=00`

pci dump—Scan and Display PCI Bus

Display the current PCI bus, device and function used by the `pci` command. This command is a combination of **pci scan** (see *pci scan—Scan PCI Bus for Devices* on page 46) and **pcid** (see *pci, pcib, pciw, pcid—Display or Modify PCI Configuration Registers* on page 47).

Syntax	`pci dump`

5.3.2.11 Execution Control Commands

go—Go Command

Continue execution of the processor until the next breakpoint condition (if one exists), or until the user issues the `halt` command.

Syntax	`go`
Example	`ICE> g`
Abbreviation	`g`

halt—Halt Execution

Halt execution of the processor.

Syntax	`halt`
Example	`ICE> halt`

trace—Single Step

Single-step execution on the processor. *`count`* is the number of times to single-step; if it is not provided, one step is implied. Also see *strace—Single Step Without Register Display* on page 49.

Syntax	`trace [`*`count`*`]`

Example	`ICE> t`
	`EAX=F00004E3  EBX=00126080  ECX=12BC51C8  EDX=6CD60040`
	`ESP=001231C8  EBP=0012325C  ESI=E0ED0000  EDI=000E3264`
	`EIP=0001515A  Flags=3A96`
	`CS=00A7  DS=00AF  SS=00AF  ES=00AF  FS=00CF  GS=00CF`
	`00A7:1515A  25 FF 00               and        ax, 00FFh`
	
	`ICE> t`
	`EAX=F00000FF  EBX=00126080  ECX=12BC51C8  EDX=6CD60040`
	`ESP=001231C8  EBP=0012325C  ESI=E0ED0000  EDI=000E3264`
	`EIP=0001515A  Flags=3A96`
	`CS=00A7  DS=00AF  SS=00AF  ES=00AF  FS=00CF  GS=00CF`
	`00A7:1515D  26 FF 00               and        bx, 00FFh`

Abbreviation	`t`

proc—Step Over the Call

Single-step execution on the processor. This is the same as `trace` except CALL instruction targets are completed as one instruction rather than stepping into the CALLed routine. (See *trace—Single Step* on page 48.)

Syntax	`proc [`*`count`*`]`
Example	`ICE> proc 4`

reset hard—Reset the Debug Connect

Instruct the TDM to assert the system reset line on the debug connector.

Syntax	`reset hard`
Example	`ICE> reset hard`

reset—Reset the CPU

The x86 processor is reset and execution is started.

Syntax	`reset`

Example	`ICE> reset`
	`…`
	`ICE> r`
	`EAX=00000000  EBX=00000000  ECX=00000000  EDX=00000542`
	`ESP=00000000  EBP=00000000  ESI=00000000  EDI=00000000`
	`EIP=0000FFF0  Flags=0002`
	`CS=F000  DS=0000  SS=0000  ES=0000  FS=0000  GS=0000`
	`F000:FFF0  EA 5B E0 00 F0  jmp  F000h:E05Bh`

strace—Single Step Without Register Display

Same as trace (see *trace—Single Step* on page 48), but without register display.

Syntax	`strace [`*`count`*`]`

Example	`ICE> strace`
	`CS=00A7  DS=00AF  SS=00AF  ES=00AF  FS=00CF  GS=00CF`
	`00A7:1515A  25 FF 00  and  ax, 00FFh`

5.3.2.12 File Operations

load—Load File Into Memory

Load a file into memory. There are two file formats supported. The input file format is binary if no `srecord` parameter is supplied. Binary files are loaded at the address specified or, if no address is given, the binary image is loaded starting at address zero.

Syntax	`load `*`filename`*` [`*`destination_address`*`] [`*`binary`*`]` `load `*`filename`*` srecord`
Example	`ICE> load image.bin 1000:100` `ICE> load test.hex srec`

File types are assumed to be binary if no type is specified. The `binary` parameter can be supplied for clarity.

If the `srecord` parameter is use the file is read as a S-record file, and the addresses specified in the file are use for the data.

Note: $0 contains the starting address at the end of the execution of the code if available; if not, it contains the starting load address.

<—Pipe Input

Read source command from the specified input file until end of the file, or a pipe input command is given without a file name. The pipe input command supports nested input files, so input files can reference other input files. If no file name is provided, the current input file is closed.

Syntax	`< [filename]`
Example	`ICE> < cmd_file.ice`

<<—Close All Input Files

Close all the nested input files, and return to accepting user input.

Syntax	`<<`
Example	`ICE> <<`

>, >>—Pipe Output

Direct the output to the specified file. Output redirection stops when a pipe output command is given without a file name. If no file name is provided, the output file is closed.

Syntax	`> [filename]  // Overwrite output file.` `>> [filename]  // Append to output file.`
Example	`ICE> > logfile.txt`

5.3.2.13 Conditional Execution

if—Conditionally Execute Command

Evaluate the expression, and perform the command if the expression yields true. Note that the command can be a compound command using `{}`, but it can not extend beyond the end of the line.

Syntax	`if ( expression ) command [ else command ]`
Example	`ICE> if (bx == ax) print "Current AX/BX is  ", bx` `Current AX/BX is E05B`

5.3.2.14 Miscellaneous Commands

wait—Wait for Breakpoint

Wait until the next breakpoint is hit, or until an optional timeout. This command is useful in scripts for automatically pausing the script until a breakpoint is hit. `wait` can also pause until a key is pressed, if the `keyboard` argument is specified.

Syntax	`wait [timeout_in_seconds] [keyboard]`
Example	`ICE> g ; wait 5 ; halt`

gowait—Go and Wait for Breakpoint

Go and wait for a breakpoint. Same as using `go` without arguments and then executing a `wait` for a particular amount of time.

Syntax	`gowait [timeout_in_seconds] [keyboard]`
Example	`ICE> gowait 5`

help—Help Command

Command-line help. This feature is only partially implemented as of this writing.

Syntax	`help [command]` `?`

cls—Clear Screen

Clear the console window.

Syntax	`cls`

?—Evaluate Expression

Evaluate the expression or expressions and display the results.

Syntax	`? expression_list`
Example	`ICE> ? ( cs << 4 ) + ip` `FE05B`

quit—Exit the ICE

Exit the ICE. The expression value is the return code value when the ICE exits. If no expression is supplied, the return value is zero.

Syntax	`quit [expression]`
Abbreviation	`q`

version—Version Information

Display the version information for CMS, the ICE DLL, and the ICE user interface.

Syntax	`version`
Abbreviation	`ver`

//—Comment

Comment to end of line. The rest of the input line after the `//` is ignored. This command enables comments to be placed in command files.

Syntax	`// comment`

;—Command separator

Separate multiple commands on the same line.

Syntax	*command* ; *command* ; …
Example	ICE: r ax ; wait 5 ; halt ; r ax AX=046C AX=2624

{ }—Multiple Command Grouping

Group commands together to be treated as one command.

Syntax	{ *commands* }
Example	ICE> if (ax != 3) { r ax ; r bx } AX=04E3 BX=6080

print—Print a List of Expressions

Print a list of comma-delimited expressions that can consist of any combination of valid expression, including strings.

Syntax	print *expression_list*
Example	ICE> print "Current BX is ", bx Current BX is 42

set—Set/Show Feature

Set or show VICE environment features. This feature is only partially implemented as of this writing.

Syntax	set // Show all the features. set [*feature*] // Show a single feature. set [*feature* [= *value*]] // Set a single feature.

Valid features are as follows:

VICE Features

Feature	Description
target	The target interface connection. FIXME 0 = TDM 1 = MI2 2 = Parallel port
lptbase	The parallel port I/O base address for the target connection.
input	A flag to determine if the input characters and prompt are displayed.
output	A flag to set if the output data is displayed to the console.
bundlepath	The path where the bundle is located.
lines	Number of lines in the ICE text command box.

!—Execute OS Console Command

Execute an OS console command.

Syntax	`! command`

Example	`ICE> ! dir ..\debug\*.ice`
	` Volume in drive T is ICE`
	` Directory of T:\working.r4.ice\src\ice\debug`
	
	`test ice 263 06-17-99 6:12p test.ice`
	`exp ice 778 06-15-99 8:47a exp.ice`
	`reg ice 999 06-26-99 6:40a reg.ice`
	`qqq ice 999 06-26-99 6:40a qqq.ice`
	`4 file(s) 3,039 bytes`
	`0 dir(s) 2,876,590,080 bytes free`

5.3.3 Syntax Notes

5.3.3.1 Numeric Values

All numeric values entered or displayed are expressed as a sequence of digits valid for the selected base. The default base for all numbers is 16. The default base for user entered values can be overridden by providing a prefix or suffix to the number.

Base	Prefix	Suffix	Examples
Binary	0y	y	0y101, 110y
Octal	0o	q *or* o	0o377, 177q, 322o
Decimal	0n	t	0n255, 128t
Hex	0x	h	0xFE, 0A3h

5.3.3.2 Supported Registers

Supported registers are as follows:

EAX	AX	AH	AL	EBX	BX	BH	BL
ECX	CX	CH	CL	EDX	DX	DH	DL
ESI	SI	EDI	DI	ESP	SP	EBP	BP
EIP	IP	EFLAGS	FLAGS				
CS	DS	ES	FS	GS	SS		
MM0	MM1	MM2	MM3	MM4	MM5	MM6	MM7
FP0	FP1	FP2	FP3	FP4	FP5	FP6	FP7
ST0	ST1	ST2	ST3	ST4	ST5	ST6	ST7
CR0	CR2	CR3	CR4				
DR0	DR1	DR2	DR3	DR4	DR5	DR6	DR7
CS.BASE		CS.LIMIT		CS.ATTR or CS.AR			
DS.BASE		DS.LIMIT		DS.ATTR or DS.AR			
ES.BASE		ES.LIMIT		ES.ATTR or ES.AR			
FS.BASE		FS.LIMIT		FS.ATTR or FS.AR			
GS.BASE		GS.LIMIT		GS.ATTR or GS.AR			
SS.BASE		SS.LIMIT		SS.ATTR or SS.AR			
GDT.BASE		GDT.LIMIT		GDT.ATTR or GDT.AR			
LDT.BASE		LDT.LIMIT		LDT.ATTR or LDT.AR			
IDT.BASE		IDT.LIMIT		IDT.ATTR or IDT.AR			
TR	LDTR						
TSS.BASE		TSS.LIMIT		TSS.ATTR or TSS.AR			
SMBASE *(read only)*							

Notes: DR0-DR7 are read only.
FP0-7 are an alias for ST0-7
Setting invalid combinations of CR0 causes the ICE and processor to hang.

5.3.3.3 Expressions

The ICE supports expression for most address and data values. These expressions are a subset of C expressions. Expressions can include type casting, constants, registers and symbols.

Supported expression operators are as follows:

+	Addition	!	Logical NOT
-	Subtraction	&&	Logical AND
*	Multiplication	||	Logical OR
/	Division	<	Less than
%	Modulus	<=	Less than or equal to
&	Bitwise AND	>	Greater than
^	Bitwise-exclusive-OR	>=	Greater than or equal to

`|`	Bitwise OR	`==`	Equal
`<<`	Left shift	`!=`	Not equal
`>>`	Right shift	`*`	Pointing to
`~`	One's complement	`(type)`	Type cast

There are some command formats where *full expressions* are not supported, i.e., you can only use a *limited expression*. This means that you can use a variable, constant or a parenthesized full expression. In other words, `(2*4)` is valid, but `2*3` is not. Often if the command supports an optional equal sign (**=**), the version without the equal sign only supports a limited expression, and with the equal sign supports a full expression.

Expression Lists

Expression lists are groups of expressions delimited by either spaces or commas. Delimiting expressions in the list by using spaces limits the format of the expression to a *limited expression*. Using commas as the list delimiter allows use of *full expressions*. For example:

1 3 4

is a *limited* expression list, but:

1+2, -4, -5

is a *full* expression list because commas are shown as separators for the expressions.

Type Casts

Type casting follows the C syntax for forcing a conversion of one type to another. In the ICE, casting can modify the address format, data type, or access width. The casting is done by placing the type in parentheses prior to the expression to be cast. For example the command:

```
d (io word) 43
```

casts the value 43 to be an I/O port address with a 16 bit access to the port.

Supported type casts are as follows:

Cast	Description
`byte`	Makes the references with 8 bit accesses.
`word`	Makes the references with 16 bit accesses.
`dword`	Makes the references with 32 bit accesses.
`io`	Convert to a I/O address.
`pci`	Convert to a PCI address.
`real`	Assume a real mode address.
`linear`	Convert to a linear address.
`physical`	Convert to a physical address.
`use16`	Override the default disassembly output to be 16 bit code.
`use32`	Override the default disassembly output to be 32 bit code.
`smm`	Make the memory reference as if the CPU was in SMM.
`nonsmm`	Make the memory reference as if the CPU was not in SMM.

5.3.3.4 Addresses

An address refers to a memory space, I/O space or PCI configuration space. The assumed address space varies depending on the command. For memory, the format of an address is *selector*:*offset*. Both selector and offset are expression, with the selector being optional. For I/O, the format of an address is a 16-bit port number. For PCI, the format of an address is *bus*:*device*:*function*:*register*, where everything but *register* is optional.

Sometimes, addresses are displayed in the form "*selector*.base+*offset*". For instance, CS.base+0xf000. This means that the processor is in protected mode, but the hidden descriptor registers do not represent the current descriptor entry in the descriptor table. This usually happens between the time the processor enters protected mode and the selector is reloaded. In such a case, ICE does what the processor does: it uses the hidden selector register contents.

There are four basic memory addressing modes: real, logical, linear and physical. The default assumption about the address mode depends upon the current CPU addressing mode.

Address mode type casting is the method for using addressing modes and address spaces different from the default for a given command. Casting is done by prefixing the address with (*type*). Possible address type casts are: real, linear, physical, smm, nonsmm, io and pci. For example:

```
(smm linear) BFEFC
```

Addresses are displayed in the same format entered, except that linear and physical address are displayed with a l or p suffix, respectively.

Address Ranges

Address ranges can be specified in one of two forms: a start and end address, or a start address and length. If the length is used, it is specified as the number of units being referenced (bytes, words, dwords, or instructions). The syntax for the ranges are:

```
address … address
address length length
```

Address Type Conversion

This feature is only partially implemented as of this writing.

From / **To**

	Numeric Value	**Logical**	**Linear**	**Physical**
Numeric Value	-	offset = value	adrs = value	adrs = value
Logical	value = offset	-	adrs = selector base+offset	adrs = selector base+offset, page table walk
Linear	value = adrs	offset = adrs	-	adrs = page table walk
Physical	value = adrs	offset = adrs	adrs = adrs	-

For value, adrs and offset assignments there is no lookup, just a direct use of the number.

`Selector base+offset`: The logical address is used to look up the linear address.

`Page table walk` = The linear address is used to find the physical address in the page tables.

Examples:

Expression	Address
`(lin) 40`	`40`*`l`*
`(phy) 40`	`40`*`l`*
`(lin) 40:17`	*`GDT`*`(40).Base+17`
`(phy) 30:12`	*`TLB`*`( `*`GDB`*`(30).Base+12 )`
`(lin phy) 35`	`35`*`p`*
`(lin) 400p`	`400`*`l`*

Accessing SMM Memory

The ICE views SMM as an attribute on the memory address. The default view of SMM is the CPU's current SMM state. The SMM state can be overridden with the **smm** and **nonsmm** type casts. This allows the user to look at the contents of hidden SMM memory without having the CPU in SMM. For example, in order to examine the current contents of the SMM memory state save map you would do the following:

```
ICE> dw (smm lin) BFEF8
000BFEF8l  0000 000B 0002 0003 0000 0000 01F0 0000
000BFF08l  0000 0000 C502 0000 8C29 0000 87BE 0000
000BFF18l  8000 0012 FF3C 0000 FF34 0000 8600 01FF
000BFF28l  0000 0000 0001 0000 FFFF FFFF 0000 0004
000BFF38l  4093 0000 FFFF 0000 0000 0005 0093 0000
000BFF48l  FFFF 0000 0000 0005 0093 0000 FFFF FFFF
000BFF58l  0000 0004 4093 0000 FFFF 0000 0000 0000
000BFF68l  0093 0000 FFFF 0000 0000 0000 0093 0000
```

The following table summarizes the interaction between SMM memory accesses and paging in the ICE.

Paging	SMM		
		OFF	**ON**
	OFF	Linear used as actual	Linear with SMM modification
	ON	Linear-to-physical lookup	Linear used as actual

5.3.4 Command Summary

Following is a summary of all available commands, their usage and valid abbreviations (if any), and the page in this document where the full documentation entry can be found:

Command	Usage	Abbrev	Page
a—Assembly	`a [`*`address_range`*`]`		40
u—Unassemble	`u [`*`address_range`*`]`		40
bpx—Set Instruction Breakpoint	`bpx `*`address`*` [if (`*`conditional_expression`*`)]`	`bp`	40

Command	Usage	Abbrev	Page

bpr, bpw, bprw—Set Memory Read/Write Breakpoint	`bpr address [if (conditional_expression)]`
	`bpw address [if (conditional_expression)]`
	`bprw address [if (conditional_expression)]`
			41

bpio—Set Breakpoint on I/O Location	`bpio port [if (conditional_expression)]`		41

bpint—Set Interrupt Breakpoint (reset, icebp, int3, num)	`bpint reset`
	`bpint icebp [if (conditional_expression)]`
	`bpint int3 [if (conditional_expression)]`
	`bpint num [if (conditional_expression)]`
			41

bl—List Breakpoints	`bl [list|*]`		41
bd—Disable Breakpoints	`bd list|*`		42
be—Enable Breakpoints	`be list|*`		42
bc—Clear Breakpoints	`bc list|*`		42
dump, db, dw, dd—Dump Data	`dump [address_range]` `d[b|w|d] [address_range]`	d	43
enter, eb, ew, ed—Enter Data	`enter address [=] expression_list` `e[b|w|d] address [=] expression_list`	e	43
fill, fb, fw, fd—Fill Data	`fill address_range [=] expression_list` `f[b|w|d] address_range [=] expression_list`	f	43
attach—Connect to a host	`attach host_name`		44
connect—Connect to a host and issue hard reset	`connect host_name`		44
disconnect—Disconnect from a host	`disconnect [continue]`		44

iolog—I/O Logging Support	`iolog add port_range`
	`iolog delete port_range`
	`iolog list`
	`iolog dump`
			44

in, ib, iw, id—Input Command	`in address` `i[b|w|d] address`	i	45
out, ob, ow, od—Output Command	`out address [=] value` `o[b|w|d] address [=] value`	o	45
register—Register Read/Write	`register [reg] [[=] expression]`	r	45
msr—MSR Read/Write	`msr reg [[=] value]`		46
cpuid—CPUID View	`cpuid index`		46
pci scan—Scan PCI Bus for Devices	`pci scan`		46
pci, pcib, pciw, pcid—Display or Modify PCI Configuration Registers	`pci[b|w|d] [[[[bus:]dev:]func:]reg [length num]]` `pci[b|w|d] [[[[bus:]dev:]func:]reg [[=]expr_list]]`		47
pci info—Display ICE PCI Information	`pci info`		47
go—Go Command	`go`	g	47
trace—Single Step	`trace [count]`	t	48
proc—Step Over the Call	`proc [count]`		48
halt—Halt Execution	`halt`		48

Command	Usage	Abbrev	Page
reset hard—Reset the Debug Connect	`reset hard`		48
reset—Reset the CPU	`reset`		49
load—Load File Into Memory	`load filename [destination_address] [binary]` `load filename srecord`		49
<—Pipe Input	`< [filename]`		50
<<—Close All Input Files	`<<`		50
>, >>—Pipe Output	`> [filename]  // Overwrite output file` `>> [filename]  // Append to output file`		50
if—Conditionally Execute Command	`if ( expression ) command [ else command ]`		50
wait—Wait for Breakpoint	`wait [timeout_in_seconds] [keyboard]`		50
gowait—Go and Wait for Breakpoint	`gowait [timeout_in_seconds] [keyboard]`		51
help—Help Command	`help [command]` `?`		51
cls—Clear Screen	`cls`		51
?—Evaluate Expression	`? expression_list`		51
quit—Exit the ICE	`quit [expression]` `q`	`q`	51
version—Version Information	`version` `ver 25`		51
//—Comment	`// comment`		52
;—Command separator	`command ; command ; …`		52
{ }—Multiple Command Grouping	`{ commands }`		52
print—Print a List of Expressions	`print expression_list`		52

set—Set/Show Feature	`set  // Show all the features`
	`set [feature]  // Show a single feature`
	`set [feature [= value]] // Set a single feature`
			52

!—Execute OS Console Command	`! command`		53

Much of the command syntax can be abbreviated, by shorting the command or one of its parameters. The following list shows the shortest valid abbreviations:

Abbreviation	Meaning
l	length
phy	physical
lin	linear
u	unassemble
d	dump
e	enter
f	fill
i	in
o	out

Abbreviation	Meaning
r	register
t	trace
q	quit
ver	version
key	keyboard
bp	bpx
srec	srecord
bin	binary
a	assemble

Appendix A

POST Codes

This section describes the Code Morphing software POST and error codes for Transmeta Efficeon™ processors.

A **POST code** is a “progress marker”, and if displayed indicates the executed code stream has successfully moved past that point. Code Morphing software boot code typically issues a POST code prior to some new functional section of code being executed, and if the next POST code never appears, the last POST code issued indicates the last known working section of code executed before a problem arose.

An **error code** is a status descriptor giving some information about a problem encountered. Code Morphing software typically issues an error code after doing something and finding a problem with it.

POST and error codes are two bytes in length. *xx* in the second byte position indicates a variable value, that for most codes is 00. The table below lists the Efficeon TM8x00 Code Morphing software (CMS) POST and error codes.

Table 1: Efficeon POST Codes in Receipt Order

Code	Description
04 00	Hello world, posts within 10 instructions of boot
58 20	Checksums match, caches initialized, late boot code decompressed and running—before memory configuration
9x xx	Error computing chip capacity, where xxx = calculated size
a0 xx	Error in configuring and sizing memory (xx = nonzero)
68 02	Finished reading SPD data
69 xx	Memory configured, xx = result (00 = success)
6a xx	Memory verified, xx = memory verification return val (00 = success)
68 03	Finished configuring memory (Note, sometimes this code doesn't show up because it is emitted in quick succession with 5830, too fast for the debugger to recognize)
58 30	Memory successfully configured
58 65	(if xboot present) before xboot init
58 80	Before CMS decompression
58 90	Main CMS successfully decompressed
58 91	Main CMS decompression failed, decompressed recovery CMS
AC ED	CMS decompressed, instructions begin
AC 04	Begin CMS nucleus initialization for regular cold boot

Table 1: Efficeon POST Codes in Receipt Order (Continued)

Code	Description
AC 06	Begin CMS nucleus initialization for resume path
AC 14	CMS component initialization complete for cold boot
AC 16	CMS component initialization complete for resume path
AC 18	CMS x86 install state complete
AC 1C	CMS about to execute first x86 instruction

Appendix B

OEM-template.rcl

Following is an example template file. Note that this file is provided as an example only—the file supplied with your version of Code Morphing software will not be identical to the example shown here. Be sure to use the template associated with your version of Code Morphing software.

Example 1: Example OEM-template.rcl

```
################################################################################
##
## Default OEM Settings file
##
## $Revision: 1.13 $
##
################################################################################

################################################################################
##
## 1.) SKU Selection
##
## Each configuration can support exactly one TM8x00 SKU. The SKU setting
## selected below must match exactly the particular SKU for this platform.
##
## Uncomment exactly one of the following SKUs:
##
## Rev 1.2 silicon
# SKU = "r1.2_debug";         # TM8600; Conservative "bring up" SKU, 533MHz at 1.0 volts
# SKU = "860012.BPLAAA1000"; # TM8600; 1000MHz/7.5W/CR70;  SKU ID: 860012.BPLAAA1000
#
################################################################################
##
## 2.) Core Voltage Regulator Selection
```

Example 1: Example OEM-template.rcl (Continued)

```
##
## Each voltage regulator has a unique Voltage Regulator Digital to
## Analog table (VRDA) associated with it.
##
## To select your regulator, uncomment exactly one of the following VRDAs:

# VRDA = "MAX1718";# Currently only MAXIM 1718 is supported

################################################################################
##
## 3.) Virtual ROM Model
##
## The TM8x00 uses the following virtual ROM model information to perform in-system
## and secure upgrades. There is a virtual ROM model for each LPC ROM support by
## the TM8x00.
##
## Uncomment the line containing the LPC ROM for your system:

# VIRTUAL_ROM = "SHARP_LHF00L01";  # Was called LH28F016 in pre-production
# VIRTUAL_ROM = "SST_49LF080A";
# VIRTUAL_ROM = "STMICRO_M50LPW116";

################################################################################
##
## 4.) Platform Identifier Fields
##
## Your configuration must include a "unique signature" consisting of a
## Customer ID and a platform variation number and possibly additional
## information (if desired). This information is used to ensure that
## secure upgrades can safely performed on your system and that different
## systems can be uniquely identified.
##
## 4.1) Customer ID and Platform Variation
##
## The Customer ID is assigned to you by Transmeta.  Transmeta assigns
## one Customer ID per company in hexidecimal format.  If you do not
## know your Customer ID, please contact your Transmeta representitive.
##
## The platform variation number is used to differentiate distictly
## different systems produced by the same company.  The platform variation
## number is assigned based on how many separate TM8x00 based designs
```

Example 1: Example OEM-template.rcl (Continued)

```
## there are at your company. For example, if this is the fifth TM8x00
## design at your company, assign the value 5 for the platform variation.
##
## Uncomment each of the following lines and replace the angle bracket
## sequences with the appropriate hexidecimal numeric value.

# CustomerID        = <customer-platform-id>; # Hexidecimal value in
                                              # the range of 1..0xFFF
# PlatformVariation = <platform-variation>;   # Hexidecimal value in
                                              # the range of 1..0xFF

## Note:
## The "CustomerID" and "PlatformVariation" are combined for form a
## single composite value (upgrade_oem_id0) in the OEM Configuration
## table as follows:
##   upgrade_oem_id0 = CustomerID << 8 | PlatformVariation

## 4.2) Additional Platform Identification
##
## If desired, an additional 32 bits of platform identifcation may be
## specified. The use of this field is unspecified by Transmeta and may
## be used by the OEM in any way they see fit.  This field is visible
## in the PCI configuration space of the northbridge as the DWORD
## identified as bus=0,device=0,func=0,reg=0xc4. As such, the OEM may
## find this field useful for assisting in software distribution that
## is platform specific.

# upgrade_oem_id1 => 0;

################################################################################
##
## 5.) Upgrade Options
##
## 5.1) Security Options
##
## The TM8x00 supports both "secure" and "insecure" upgrades. A "secure"
## upgrade is only possible with a special signed upgrade image generated
## by Transmeta. An "insecure" upgrade can be performed with any ROM
## image. It is not necessary to obtain Transmeta's assistance to perform
## insecure upgrades. It is generally desirable for fielded systems to
## be secure but may be desireable for systems to remain insecure during
```

Example 1: Example OEM-template.rcl (Continued)

```
## the development and debug phases of development.
##

## The bitfield global_write_security is 2 bits wide, and it controls write
## access to the VNB field global_write_enable.  Here are the meanings of
## the possible values:
##
##      0 = rom_write_secure
##      1 = rom_write_hw_dongle
##      2 = rom_write_insecure
##      3 = invalid
upgrade_options.global_write_security    => 0;  # Default is "rom_write_secure"

## The bitfield bios_write_security controls permission to decrease the VNB
## field rom_protected_size.  Here are the meanings of the possible values:
##
##      0 = bios_write_controlled (GPI: BIOS_WE enable)
##      1 = bios_write_insecure
upgrade_options.bios_write_security      => 0;  # Default is
                                                # "bios_write_controlled"

################################################################################
##
## 6.) Longrun Frequencies
##
## The longrun frequencies provided by Transmeta have been optimized for
## each SKU.  It is possible that special circumstances would require a
## special set of Longrun frequencies. Please consult with your Transmeta
## representative before overriding the default Longrun settings table.
## Frequencies must be specified in ascending order (lowest to highest)

## If a non-standard longrun table is required, fill in the table below
## with desired longrun frequencies in ascending order. A table of all
## "0" indicates that the Transmeta supplied SKU defaults should be used.
longrun_frequencies                      => {0, 0, 0, 0, 0, 0, 0, 0};

################################################################################
##
## 7.) CPU Features
##
## 7.1) Processor Serial Number
```

Example 1: Example OEM-template.rcl (Continued)

```
## Set to "1" to permanently disable the processor serial number (PSN).

cpu_feature.psn_disable                  => 0;

## 7.2) ECC Memory Disable
## Set to "1" to treat ECC memory as non-ECC memory.

cpu_feature.disable_ecc                  => 1;

## 7.3)
## Set to "1" to indicate that SSTL_2 termination is in use for the DRAM bus.

cpu_feature.SSTL2_termination            => 0;

###############################################################################
##
## 8.) Memory Configuration
##

## 8.1) SPD Data For Soldered Down Memory -- PLEASE READ CAREFULLY
##
## To enable the use of soldered down memory, uncomment
## one or more of the following lines. The full path to
## the appropriate SPD s-record file should be included
## within the single quotes.
##
## Use the Transmeta utility "make-SPD" to create the appropriate SPD
## file based on information provided by your memory vendor.  Modify the
## file "SPD-template.rcl" with this information. See the Tools Guide
## for complete directions on how to use this utility.

# MemSPDfile_0 = 'FULL PATH TO VALID SPD FILE (S-RECORD FORMAT)';
# MemSPDfile_1 = 'FULL PATH TO VALID SPD FILE (S-RECORD FORMAT)';


## 8.2) SPD ROM Address
## The address (in hexidecimal) on SMBUS of the first SPD ROM

mem_smbus_spd_base_addr                  => 0x50;

## 8.3) Clock Assignements
```

Example 1: Example OEM-template.rcl (Continued)

```
## This table maps DIMM slot numbers to the clocks needed for that
## DIMM.

mem_slot_to_clocks                      => {0x07, 0x38, 0x00, 0x00};

## 8.4) Lowest Memory Frequency
## Specify the lower limit of memory frequency in 100/6 MHz increments.

mem_freq_min                            => 5;  # 83.33 MHz

## 8.5) Highest Memory Frequency
## Specify the upper limit of memory frequency in 100/6 MHz increments.

mem_freq_max                            => 10; # 166.67 MHz

## 8.6) CMS Memory Allocation
## Size of the block of memory (in MB) to reserve for CMS usage. Legal
## values range from 8 to 128 with a recommended absolute minimum of
## 24 MB.
cms_memory_size                         =>  32;

################################################################################
##
## 9.) Debug Information
##
## 9.1) Debug port I/O address
## Specify the I/O port address used for BIOS (and other application)
## POST codes (0 to disable)

io_port_debug_led                       => 0x0000;  # Default is disabled

## 9.2) S-Clk Delays
## Table mapping sclkdly values to max. memory freq. (in 16.67 MHz
## incr.). This table depends on the length of the DDR traces on the
## board. Longer traces mean that lower values of sclkdly top out at
## lower frequencies.

sclkdly_to_mem_frequency                => {7, 10, 255, 255};

## 9.3) Minimum Loads
## Table specifying how many chip loads in each signal group can be
```

Example 1: Example OEM-template.rcl (Continued)

```
## driven with drive strength set to 0 (min).  It depends strongly on
## board capacitance.

group_to_min_loads                        => {0, 0, 0, -1};

## 9.4) Maximum Loads
## Table specifying how many chip loads in each signal group canbe
## driven with drive strength set to 7 (max).  It depends weakly on
## board capacitance.

group_to_max_loads                        => {18, 18, 18, 9};

################################################################################
```

Appendix C

Recommended Reading

The following documents should be used in conjunction with this guide. Due to the preliminary nature of this document, not all reference documents may be available.

- *Efficeon™ Data Book*
- *Efficeon™ BIOS Programmer's Guide*
- *Efficeon™ Functional and Specification Errata*
- *Efficeon™ System Design Guide*
- *Efficeon™ Code Morphing Software Release Notes*
- *Macraigor Raven User's Guide*

Glossary

A

ABI—Application Binary Interface. The detailed low-level conventions that applications use to communicate with an operating system or a language/environment library. An ABI is the low-level realization of an API. It consists of the register conventions, parameter passing and return conventions, invocation of the O/S conventions (e.g. through an exception, etc.), etc.

ACPI—Advanced Configuration and Power Interface. An operating-system-centric power management scheme based on a description of a set of hardware interfaces that the operating system can use to both change the active and suspend power states of the CPU and to determine the relative advantages and implement an appropriate policy. BIOS sets up the ACPI tables, and the OS uses the tables to enter and exit states and decide when it is profitable to do it.

AGP—Advanced Graphics Port. An interface added to modern northbridge chips used by graphics accelerators. It is similar to the PCI bus, but wider and running at a faster frequency. It is logically a PCI bus on its own right, but running at a higher frequency.

API—Application Programming Interface. The high-level (often C-language) interface that applications use to communicate with an operating system or a language/environment library. By extension, the high-level interface presented by any subsystem (e.g. a telephony API).

APIC—Advanced Programmable Interrupt Controller. An extension to the x86 architecture which allows finer granularity of control over interrupts and more interrupt sources without sharing an IRQ. The APIC architecture is composed of two components, the local APIC in each CPU, and the I/O APICs in the southbridge and elsewhere. The APICs communicate either by a dedicated protocol (dedicated bus in the P6 class of Intel CPUs) to arbitrate interrupt delivery.

atom—An individually-encoded operation in a VLIW instruction. A VLIW instruction, also called a molecule, consists of a collection of operations called atoms. Atoms are similar to the instructions in RISC architectures.

B

BIOS—Basic Input Output System. A collection of software utilities that virtualizes some of the platform for the benefit of the operating system. Typically it gains control after reset (power up), and after configuring some platform-specific hardware, loads the operating system into memory and transfer control to it. It remains resident as a set of subroutines to virtualize the keyboard, the mouse, the screen, the disk, etc. Modern operating systems bypass the BIOS for most services, and the BIOS has become almost exclusively a boot loader.

bridge—A device connected to two or more buses that forwards requests or transactions from one to another. The northbridge, among oth er things, is a bridge logically connecting the processor local (or front-side) bus to the PCI (and AGP) bus. Some southbridges, among other things, are bridges connecting the PCI bus to an ISA bus.

C

CMS—Code-Morphing™ Software. A dynamic translator and interpreter system that emulates some well-established architecture (e.g. x86) on top of Transmeta's unique VLIW hardware.

CPU—Central Processing Unit. The processor in a computer system. It executes the SW instructions which advance state and may in turn access peripheral devices. Generally excludes memory and I/O devices, unless embedded in the CPU.

CPX—Cycles Per X-tick. A measure of x86 machine efficiency/performance. At the same frequency and running the same application or benchmark, a lower CPX generally indicates better performance.

D

DDR—Double Data Rate. A bus protocol technique in which data is transferred on both rising and falling edges of the bus clock, effectively achieving twice the data throughput. It is the counterpart to SDR on which data is transferred only on rising or falling edges of the clock. Examples of DDR interfaces are the TM8000 MI (memory interface) unit, and the HyperTransport link from the TM8000 Virtual Northbridge to the southbridge.

DIMM—Dual In-line Memory Module. A memory (DRAM) module with a particular organization.

DMA—Direct Memory Access. Traditionally, the ability of devices on a bus to access main memory without the processor's intervention—a form of doing I/O where the CPU is not directly involved in the transfers, unlike programmed I/O (PIO). In modern computer systems the DMA facility is implemented in the memory controller, called the northbridge in PC terminology. In Transmeta products, the processor is involved to maintain T-cache coherence.

DRAM—Dynamic Random Access Memory. RAM implemented by capacitor circuits (single transistor cells, typically). Very dense, but requires frequent refreshing since the capacitors slowly discharge (leak), eventually losing their data contents. In addition, reads typically require rewriting because driving the output lines also discharges the capacitors. Refresh is sometimes implemented internally by parallel read/write.

E

ECC—Error Correcting Code. A system in which data words (esp. memory) are extended with a few extra bits such that some number of bit errors can be detected and some (smaller) number of bit errors can be corrected. Typically done on 64-bit words with 1 bit error correction and 2 bit error detection because a simple scheme requires only 8 additional bits, thereby taking no more storage than byte-parity and enabling the use of the same memory DIMMs.

EIP—Extended Instruction Pointer. The architectural 32-bit register in an x86 (a.k.a. IA-32) processor that holds the address of the currently executing instruction. The lower 16 bits are called the IP or `instruction pointer'.

EISA—Extended Industry Standard Architecture. A 32-bit peripheral (I/O) bus defined by several companies in the PC industry as a counterpart to IBM's proprietary MCA bus. Never dominant, it was superseded by the PCI bus.

F

FPR—Floating-Point Register. A register in the floating-point unit. It can contain floating-point data, integer data, or packed media data. It cannot be used to address memory directly.

FPU—Floating-Point Unit. A part of a modern CPU that contains floating-point registers and implements floating-point and media operations.

front-side bus—A bus connecting a CPU (or CPU core) to the memory controller/northbridge. Also called the processor local bus. It is typically CPU-core specific, and may be only conceptual when the northbridge is physically in the same chip as the CPU core, as in Transmeta products.

G

GART—Graphics Address Relocation Table. A facility in modern AGP-enhanced northbridge chips used to give SW and the graphics accelerator the illusion that they are dealing with contiguous physical memory which is in fact sparsely allocated out of a page pool by the operating system without regard to contiguity.

GR—General Register Same as GPR.

GPR—General Purpose Register. An integer register that can be used to contain integer data or memory addresses. As opposed to dedicated or special-purpose registers such as floating-point registers or special registers that control the memory subsystem, etc.

H

HT(1)—HyperTransport. See LDT.

HT(2)—HyperTransport controller. The unit in the TM8000 Virtual Northbridge that implements the HyperTransport host bridge.

HW—Hardware

HyperTransport—A new name for LDT.

I

ICE—In-Circuit Emulator. A hardware device that allows very low-level debugging of a target system when controlled by software from a debugging host. It typically allows single stepping of system instructions and low level operations before and while the operating system and/or BIOS are running.

I/O, IO—Input Output

IOIO—I/O-port I/O. On the x86 architecture, I/O performed by using IO ports, i.e. in and out instructions to ports. These ports are not memory-mapped. They are an altogether different address space.

IP(1)—Internet Protocol. A low-level network protocol that covers routing and delivery over large area networks. UDP and TCP are protocols built on top of IP, which is often built on top of Ethernet and other such protocols.

IP(2)—Intellectual Property. Knowledge protected by trade secret, copyright, or patents. Sometimes used to describe some synthesizable logic (e.g. Verilog) that provides some modular functionality and that companies license to other companies for the use in their chips, e.g. an IDE controller.

IRQ—Interrupt request line. Each interrupt request line results in a distinct interrupt vector to the CPU. Interrupt request lines can be uniquely assigned to devices or shared between devices. When unique, the interrupt handler can know exactly which device needs service. When shared, the interrupt handler must poll the devices that share the line to figure out which need service.

ISA(1)—Instruction Set Architecture. The software-visible portion of a CPU's organization. It typically consists of the resources (e.g. registers), operations (e.g. instructions), and encodings of both that software needs to use to make use of the computer. On traditional CPU families, what is common between the members of the family and allows SW to run across the family.

ISA(2)—Industry Standard Architecture. The 16-data-bit, 24-address-bit memory and peripheral (I/O) bus for the IBM PC-AT—the first with an Intel 286 processor. Until recently, all PCs had one. The trend these days, especially in laptops, but even on desktops, is to abandon it after moving all legacy devices to the southbridge.

L

L1—Level-1. First level of the cache hierarchy. On the TM8000 there is an L1 data cache and an L1 instruction cache.

L2—Level-2. Second level of the cache hierarchy. On the TM8000 there is a combined data and instruction L2 cache.

LDT—Lightning Data Transport. A peripheral interconnect fabric designed by AMD to replace the PCI bus and possibly others (AGP). Logically it is very similar to PCI. Physically it is not a bus at all, but instead a point-to-point connection. LDT devices can be daisy chained and additional chains can be split from the

primary chain. It is the TM8000's primary external interface. During 2001, LDT has been renamed 'HyperTransport'.

LongRun—A Transmeta technology that allows the x86 processor to change operating clock frequency and supply voltage on the fly to consume as little power as possible while satisfying the computational needs of the running OS and applications. It is implemented by a combination of HW (SW-controlled frequency and voltage) and SW (mechanisms and policies for frequency and voltage ramping).

LongRun Advanced Thermal Management—A Transmeta technology

l-value—Left value. A compiler term describing a location written by an assignment statement. Opposed to r-value which is a value that can be stored into such a location. Thus an assignment consists of an l-value and an r-value.

M

MCA(1)—Machine Check Architecture. An x86 extension that allows diagnostic software to inspect the state of some internal processor registers when an MCE is raised. It is processor-specific.

MCA(2)—Micro-Channel Architecture. A 32-bit peripheral (I/O) bus defined by IBM to supersede the ISA bus in the PS2 PC architecture. It was proprietary, and hence other companies came up with a different standard, EISA, that ultimately proved more popular. The fiasco over MCA marked the end of IBM's dominance of the PC system architecture.

MCE—Machine Check Exception. An x86 exception raised by an x86 processor when it encounters some irrecoverable internal condition. It indicates a failure in the processor, not the running program at the time the fault is raised.

MI—Memory Interface. The unit in the Virtual Northbridge that controls and communicates with the DRAM DIMMs. It supports only DDR DIMMs.

MMIO—Memory-mapped I/O. Input/output to devices by using accesses to the regular "memory" address space. Device control registers and buffers are mapped into the "memory" address space and respond to ordinary reads and writes on the bus by effecting the appropriate action.

MMX—Multi Media eXtensions. SIMD integer extensions to the x86 architecture.

molecule —A VLIW instruction encoding multiple operations that logically occur simultaneously. Each individual operation is called an atom.

MTRR—Memory Type Range Register. An x86 architectural extension by which the memory attributes of an access (cacheability) can be described at the physical level -- the address space can be divided into multiple regions with different cacheability properties for reads and writes. It was introduced by Intel to handle several problems in earlier systems where the cacheability of an access was known to the northbridge (PAB bits) but not the processor. It is in effect a processor-local "copy" of the PAB bits and memory top register.

N

NaN—Not A Number. An IEEE 754 format floating-point "value" that does not have a numeric value. They are computed as the masked response to invalid operations (e.g. divide 0 by 0), and also sometimes used for uninitialized memory. They come in two flavors. Signalling NaNs (SNaN) raise the invalid numeric exception unconditionally. Quiet NaNs (QNaN) only raise it in certain circumstances. Typical masked response is to produce quiet NaNs that will propagate through the computation contaminating the result (only comparisons can eliminate NaNs -- arithmetic produces NaN outputs when given any NaN inputs).

Natural Alignment—A power-of-two-sized memory operand is naturally aligned if the address of the operand is an integer multiple of the size of the operand. For example a 1-byte-sized operand is always naturally aligned. An 8-byte-sized operand is naturally aligned only at addresses that are multiples of 8, etc.

NC—Non-cached or Non-cacheable. A memory (load/store) operation that does not go through the cache subsystem. It may be an operation to an IO port (IOIO), memory mapped IO (MMIO), or a memory-bound

operation that CMS has decided must follow the NC path in order to guarantee cache coherence in the system as a whole or for performance reasons.

NMI—Non-Maskable Interrupt. An interrupt that cannot be blocked by software. The x86 NMI is only blocked by the execution of the SMI handler. Transmeta HW allows blocking x86 NMI, but CMS makes it appear that it is not blocked except in this circumstance.

normal memory—An x86 address space region where reads and writes (or x86 instruction fetches) can be optimized and reordered because it corresponds to cacheable DRAM on the memory controller in the northbridge.

Northbridge—A part of a PC's chip set that implements the memory controller, the DMA engine, and the PCI bus master. Modern ones also implement AGP. It is connected to the CPU via a non-standard local (or front-side) bus, and is connected to the southbridge via the PCI bus and a few additional signals.

O

OOL(1)—Out-Of-Line subroutine. A subroutine in CMS that implements the functionality of some complex x86 instruction. Rather than inlining the code in translations and the interpreter, these out of line handlers are used instead. The OOL doesn't have to implement the complete instruction -- part of it (e.g. loading from the stack) can be done in line.

OOL(2)—Object Oriented Language. Sometimes used abbreviation for a high-level computer language whose principal data items are organized as objects with lifetimes and properties independent of the programs that manipulate them. Often used for languages with message-based invocation semantics and modularity and encapsulation organized around data instead of code.

OS—Operating System. The main control program of a computer system. It typically multiplexes the computer's resources (including execution time) among competing application programs and provides several services to them such as uniform I/O APIs that abstract the details of the individual devices in use.

P

PAB—Programmable Attribute Bits. The set of bits in a traditional northbridge that control the access to the legacy region (640K - 1M), especially in regards to ROM shadowing. The PAB bits allow reads and writes to independently be routed to memory or the PCI bus.

PAE—Physical Address Extension. An extension to the x86 architecture by which physical addresses can be 36-bits wide, instead of the usual 32-bits wide. Each PTE/PDE becomes 8 bytes long instead of the usual 4 bytes long, the page tables are 3 levels deep instead of the usual 2 levels, and the large pages (as in PSE) are 2MB instead of 4MB.

PAT—Physical Attribute Table. An x86 architectural extension by which the memory attributes of an access (cacheability) can be described at a virtual level instead of physical level. Each virtual to physical mapping has a memory type index in the relevant PTE that indexes the PAT to produce a memory type that is then combined with the type from the MTRRs to produce the actual memory type for the access.

PC(1)—Personal Computer. A computer intended to be used by a single person, i.e. traditionally running a single-user operating system. More commonly, an x86-compatible computer, i.e. what used to be called an IBM-compatible computer, irrelevant of whether the operating system running on it makes it multi-user or not.

PC(2)—Program Counter. The conceptual register in a processor that holds the address of executing instructions. In pipelined processors there may not be such a register -- each pipeline stage may have its own copy. PCs are usually not visible as registers -- they are reified on interrupts/exceptions, and reflected on resume from exception/interrupt return. In the x86 architecture, the PC is called EIP.

PCI—Peripheral Component Interconnect. The peripheral (I/O) bus of a modern PC. It is a 32-bit bus with multiple address spaces: The I/O port (IOIO) address space, the ordinary memory-mapped (MMIO) address space, and the configuration address space. It was first introduced in the PC architecture by Intel with the Pentium processor, to rationalize and standardize the multiple peripheral buses then extant (VESA local bus, EISA, MCA, ISA). Although memory-like devices can be added to the bus, this is unusual, and memory is

typically elsewhere on a system. The system's northbridge typically connects to both memory and the PCI bus, but the memory is not on the PCI bus, although accessible from it by using DMA.

PDE—Page Directory Entry. A data structure in the page tables that aggregates a collection of virtual to physical mappings. It does not describe any virtual to physical mapping itself, but allows the page table walker to find the individual mappings. In the usual two-level x86 page table structure, the PDEs are the level between the root and the PTEs.

PGE—Page Global Extension. An extension to the x86 architecture by which specially-marked 'global' virtual to physical mappings survive TLB invalidations and can only be invalidated explicitly.

PIC—Programmable Interrupt Controller. Traditionally, a device on a PC's motherboard that multiplexed interrupts to the CPU. These days the PIC is implemented as part of modern southbridges, and not as separate components.

PIO—Programmed I/O (input/output). A way of doing I/O where the CPU actively executes instructions to perform the actual transfer of data, as opposed to DMA, where the CPU is passive: the transfer of data is done by an asynchronous device without any CPU instructions needed except to set up the transfer.

PSE—Page Size Extensions. An extension to the x86 architecture by which virtual pages can be either 4MB (large pages) or the usual 4KB. Virtual to physical mappings for large pages take the place of PDEs in the page tables.

PSE-36—Page Size Extensions - 36 bits. An extension to the x86 architecture that combines features of PSE and PAE. The PTEs and PDEs remain as in the normal 2-level page tables, but the 4MB pages can be mapped to a full 36-bit address space, instead of the usual 4KB address space.

PTE—Page Table Entry. A data structure in the page tables that describes the virtual to physical mapping for a single virtual page. In the usual multi-level x86 page table structure, the PTEs are the level farthest from the root.

Q

QNaN—Quiet NaN. See NaN.

R

RAM—Random Access Memory. The main memory of a computer system, addressable in no particular order.

ROM—Read-Only Memory. A memory-like device that can only be read and not written. Most modern ROMs are not really ROMs at all, but Flash ROMs or EEPROMs. EEPROMs are electrically-erasable programmable read-only memories, i.e. memory-like devices that are easy to read and difficult -- but possible -- to write. Flash ROMs are similar to EEPROMs that cannot be erased at the individual byte or word level but can be erased in units of moderate to large sectors. These EEPROMs and Flash ROMs are treated largely like true ROMs, except when their contents are upgraded, which is a very infrequent operation. Programming them (i.e. writing to them), is usually called "flashing", perhaps because early EPROMS (erasable programmable read-only memories) could only be erased by exposing them to ultra violet light.

r-value—Right value. See l-value.

S

SDR—Single Data Rate. A bus protocol technique in which data is transferred once per clock cycle, typically on rising edges of the clock. It is the counterpart to DDR in which data is trasferred twice per cycle.

serialization —The property that prevents two operations from being either reordered or carried out simultaneously. There are many forms of serialization in the x86 architecture, including but not restricted to code serialization and data serialization.

serializing instruction—An instruction that has some serialization property with respect to other instructions. x86 serializing instructions can be strong or weak.

SIMD—Single Instruction Multiple Data. A form of obtaining parallelism that consists on having single instructions operate on multiple data items simultaneously. The operations are usually uniform. The Intel variants of this, called MMX, SSE, and SSE2 divide large (64-bit or 128-bit) registers into multiple same-sized fields, and the operations operate on all the fields independently.

SMC—Self-Modifying Code. Code that either modifies its own instructions (the proper usage), or has its instructions modified by other pieces of code (the general usage). Particular common patterns are Bit-blit generators, which frequently generate code on the fly in a common buffer, dynamic compilers (such as CMS itself), and runtime patching of immediates.

SMI—System Management Interrupt. A special interrupt in an x86 CPU that causes the CPU to enter SMM.

SMM—System Management Mode. An extension to the x86 architecture that allows BIOSs to control peripheral devices without OS awareness. Often used to implement power-management functions such as powering off and restarting disks without the OS being aware. SMM has access to regions of memory (SMRAM) that the regular OS cannot address. SMM is the state of the processor when servicing an SMI.

SMP—Simultaneous Multi-Processor. A multi-CPU computer system (a multicomputer) where the CPUs logically share memory so that ordinary accesses from one CPU are visible from other CPUs without additional software support. The rules for serialization and ordering of memory references and visibility among the multiple CPUs can vary a lot. The memory may be either uniformly accessible or non-uniformly accessible, where logically there is no difference, but there is a large performance difference depending on the locality of the memory being accessed.

SMRAM—System Management RAM. An alternate address space, or extension of the regular x86 physical address space, used in SMM.

SNaN—Signalling NaN. See NaN.

Southbridge—A part of a PC's chip set that implements several sundry functions: bridge between the PCI and ISA buses, power-control of the platform, IDE bus controllers, serial, parallel, mouse, and keyboard ports, USB controllers, audio devices, ROM control, etc. Connected to the Northbridge via a PCI or LDT interface and to both the northbridge and the CPU through additional signals.

SRAM—Static Random Access Memory. RAM implemented out of storage cells that do not decay if unattended. Much less dense than DRAM, but typically much faster and not requiring refresh. Common modern designs are made out of 6 or 4 transistor cells (6T or 4T).

SSE—Streaming SIMD Extensions. SIMD floating-point extensions to the x86 architecture. There are two levels: SSE proper which introduced SIMD 32-bit (single precision) floating-point operations, and SSE2 which introduced SIMD 64-bit (double precision) floating-point operations. The extensions also include integer and MMX operations on the new XMM registers.

SSM—State Save Map. Data structure in SMRAM where an x86 CPU saves its state on entry to SMM and from where it restores it on exit from SMM.

sticky bit—A bit in a register is said to be sticky when it records a condition in a monotonic way. That is, when the condition arises, it changes state, and does not change back even if the condition disappears. It is only changed back by explicit software intervention.

STPCLK—Stop Clock. A (logical) signal in a PC system originating in the southbridge and used to stop the CPU at an appropriate point. Such stop can be a prelude to power down, or a temporary stop due to power management events. STPCLKs can be expected, when the CPU initiates a power management transition or shutdown, or unexpected, when the southbridge initiates clock throttling often caused by overheating.

STPGRNT—Stop Grant. A (logical) signal in a PC system originating in a CPU and used to inform the southbridge that a preceding STPCLK has been honored. Typically the STPGRNT must follow quickly after the STPCLK because southbridges are intolerant of many intervening transactions.

strapping options—Hardware interface pins whose logic value is sampled at reset time by hardware in order to choose an option among initial configuration possibilities. Examples of such strapping options are processor number for SMP systems, device ID selection addresses for identical devices on a bus, etc.

strongly-serializing instruction—A serializing instruction that constitutes an absolute barrier to previous and following instructions or memory operations. All previous instructions are guaranteed to have completed by the time that the strongly serializing instruction executes, and all following instructions are guaranteed not to have started until after the strongly serializing instruction completes. Strongly serializing instructions cause write combiners to be drained.

SW—Software

T

TBD—To Be Determined. Something not yet fully understood or specified.

thread—A control-flow and register state within an address space. Several threads can execute within a single process. Each thread typically has its own logical register set and stack but shares the address space and O/S resources with other threads in the same process. Sometimes threads also have thread-local data. Win32 threads and POSIX threads are the most common thread APIs.

TLB—Translation Look-aside Buffer. A hardware (and sometimes software) structure used to accelerate memory references in a paged environment. It is a cache of mappings from virtual addresses to physical addresses and some attributes of the mappings used to avoid walking the page tables on every access. These virtual to physical mappings are called translations, but are not to be confused with the instruction translations that CMS produces.

trip count—The number of iterations that a loop executes when entered from outside. Integer loops often have very low trip counts, in the vicinity of 3. Floating-point and media loops often have very high trip counts (hundreds of thousands or millions). Many loop optimizations are only worthwhile if the typical trip count is moderately high.

U

ULP—Unit of Least Precision. The least significant bit position in a floating-point number. Often used to describe the accuracy of the implementation of some floating-point function (e.g. transcendentals). If the accuracy is within 1/2 ULP, the answer is exact. If within 1 ULP, then a result is off by at most the quantity that a bit in the least significant position of the mantissa would represent, and so on.

UMA—Unified Memory Architecture. A system setup where all graphics memory (frame buffer and off frame storage for display lists, textures, etc.) lives in system memory -- directly accessible by the CPU, rather than in dedicated memory that the CPU cannot directly access. In other contexts, the acronym stands for 'uniform memory architecture', as opposed to 'non uniform memory architecture'.

USB—Universal Serial Bus. A bus in modern PCs that is distinguished by using a serial protocol over a small number of wires. Several devices that used to be connected through dedicated interfaces are now connected to the computer through this bus.

V

VICE—Virtual In-Circuit Emulator. A software component of Code Morphing software, coupled with a software program running on a debugging host that together emulate a true x86 ICE.

Visible interrupts—Interrupts that are not invisible interrupts.

VLIW—Very Long Instruction Word. A style of ISA characterized for large (in bits) instructions that group several independent operations that are executed in parallel and atomically.

VNB—Virtual North Bridge. A software component of CMS that logically emulates the registers and operations of a physical northbridge chip, mapping them to whatever hardware the Transmeta product implements directly.

VSB—Virtual South Bridge. A software component of CMS that logically emulates the registers and operations of a physical southbridge chip, mapping them to whatever hardware the Transmeta product implements directly.

W

weakly-serializing instruction —A serializing instruction that has some reordering properties with respect to some other instructions but not all. Thus some instructions are serialized with respect to a weakly serializing instruction, but not all instructions are.

X

x86—Of or compatible with 80x86 processors manufactured by Intel. Common x86 platforms include Intel's Pentium processors, AMD Athlon processors, and Transmeta processors.

X-bus —A derivative of the 8088's original bus. The 8088 was a variant of the 8086 (first processor in the x86 architecture line from Intel) with only an 8-bit-wide data bus (20-bit addresses). It is still often found hanging off the southbridge in a PC system to connect to the flash ROM containing the BIOS, and a few very simple devices such as the keyboard controller.

Index