



# **Working with Dual SRAM for MIPS32® M4K™ Cores Application Note**

**Document Number: MD00536**

**Revision 02.00**

**April 22, 2007**

**MIPS Technologies, Inc.  
955 East Arques Avenue  
Sunnyvale, CA 94085-4521**

**Copyright © 2007 MIPS Technologies Inc. All rights reserved.**





# Table of Contents

<b>Section 1: Introduction</b> .....	<b>5</b>
<b>Section 2: Example Files</b> .....	<b>6</b>
2.1: File Summary .....	6
2.2: Source Code.....	6
2.3: Makefile .....	6
2.4: Linker Script.....	7
2.4.1: Excerpts from elf32mipssde.xn .....	7
2.5: MIPSsim Configuration Files .....	8
2.5.1: Excerpts from M4K.cfg.....	8
2.5.2: Excerpts from MipsMemInf-M4K.cfg .....	8
2.6: sde-gdb Command Script.....	9
2.6.1: Excerpts from gdbscript .....	10
<b>Section 3: Build and Debugging Environment</b> .....	<b>11</b>
3.1: Recommended Tools and Platforms .....	11
3.2: Installation and Setup of MIPSsim and SDE .....	11
3.3: Installing the Application .....	11
3.4: Building the Application (SDE 6.06.01 or higher) .....	11
3.5: Building the Application (SDE below 6.06.01) .....	12
3.6: Debugging on MIPSsim .....	12
3.7: Debugging on Malta .....	13
3.8: Program Output .....	13
3.9: Using ISRAM/DSRAM: Malta vs. MIPSsim .....	14
<b>Section 4: Programming the m4k_sram_dual_switch*.v</b> .....	<b>15</b>
4.1: Theory of Operation.....	15
4.2: Control Register Summary .....	16
4.3: ISRAM Configuration Register (0x1e400000) .....	16
4.4: ISRAM Mask Register (0x1e400010) .....	17
4.5: DSRAM Configuration Register (0x1e400020).....	17
4.6: DSRAM Mask Register (0x1e400030) .....	18
<b>Section 5: References</b> .....	<b>19</b>
<b>Section 6: Revision History</b> .....	<b>20</b>



# 1 Introduction

The MIPS32® M4K™ Processor Core is cache-less and instead uses a SRAM-style bus that may be configured with either a Dual Memory Interface or a Unified Memory Interface. The Dual Memory Interface has separate instruction (ISRAM) and data (DSRAM) memory interfaces, and provides a redirection mechanism that permits D-side references to be handled by the I-side. The Unified Memory Interface has a single memory interface that shares both data and instructions.

This application note will introduce you to working with the higher performance Dual Memory Interface (ISRAM and DSRAM). Source code and scripts accompany this application note that demonstrates how to build and debug an application in ISRAM/DSRAM. Target platforms include the MIPSsim™ software core simulator, and the Malta™/YAMON™ hardware platform.

## 2 Example Files

This section summarizes the source code and script files used to build and debug the example application. These files are located in the `.../ref_design/dual_sram` directory of a M4K core deliverable.

### 2.1 File Summary

A list of the examples files followed by brief descriptions is shown in [Table 1](#).

**Table 1 File List for Example Application**

File Name	Description
<code>sram_test.c</code>	Main program
<code>sram_mips16.c</code>	Test code overlay compiled for MIPS16
<code>sram_mips32.c</code>	Test code overlay compiled for MIPS32
<code>Makefile</code>	Makefile used to build the program with SDE
<code>elf32mipssde.xn</code>	Linker script (modified from default at <code>.../sde/lib/ldscripts/elf32mipssde.xn</code> )
<code>M4K.cfg</code>	MIPSSim configuration file
<code>MipsMemIntf-M4K.cfg</code>	MIPSSim I/O Sub System (memory) configuration file
<code>gdbscript</code>	Simulation script for sde-gdb
<code>README</code>	The README file

### 2.2 Source Code

The main program `sram_test.c` acts as the test harness. It is responsible for first copying the test code to system RAM or ISRAM/DSRAM, then running the test code, and finally displaying the performance results. It is compiled with user-defined options and is linked to execute from the loadable image shared with other SDE objects.

The test code is located in the files `sram_mips16.c` and `sram_mips32.c`. Both files are functionally identical, but the functions in `sram_mips16.c` are compiled for MIPS16 object code and the functions in `sram_mips32.c` are compiled for MIPS32 object code. That compiler attributes from `sram_mips16.c` and `sram_mips32.c` (shown below) override the compiler command line options.

```
#define MIPS_C __attribute__((mips16)) /* Compile for MIPS16 only */
#define MIPS_C __attribute__((nomips16)) /*Don't compile for MIPS16 */
```

The test code first writes a sequence-up pattern to memory followed by a Fletcher checksum memory test. Then it writes a ones-compliment sequence-up pattern to memory followed by another Fletcher checksum memory test. Finally it returns the elapsed cycle count to the main program.

### 2.3 Makefile

The Makefile uses typical SDE command options with the following exceptions:

```
CFLAGS += -Os -g -ffunction-sections -mno-data-in-code -mno-embedded-data
```

```
LDSCRIPT += -Wl,--script=elf32mipssde.xn
```

The compiler command line options “-mno-data-in-code” and “-mno-embedded-data” are necessary when building MIPS16 code to separate .data sections from .text sections. Also, the linker script command line option “-Wl,--script=elf32mipssde.xn” tells the linker to use the local linker script file elf32mipssde.xn.

## 2.4 Linker Script

The linker script file elf32mipssde.xn was modified from the default SDE script located at .../sde/lib/ldscripts/elf32mipssde.xn. The linker script has been changed to separate the .text sections from the other (i.e., .data) sections in the test objects sram\_mips16.o and sram\_mips32.o. For ISRAM/DSRAM testing, the instructions in the .text sections are copied to ISRAM, and the data in the other sections are copied to DSRAM. The excerpts from elf32mipssde.xn (shown in the following subsection) illustrate the modifications and additions to the default SDE linker script.

### 2.4.1 Excerpts from elf32mipssde.xn

```
*(.text .stub EXCLUDE_FILE (sram_mips32.o sram_mips16.o) .text.* .gnu.linkonce.t.*)

/* Set Base address for ISRAM & DSRAM */
__base_addr_isram32 = ALIGN(0x100000) + 0x100000;
__base_addr_isram16 = ALIGN(0x100000) + 0x102000;
__base_addr_dsram32 = ALIGN(0x100000) + 0x108000;
__base_addr_dsram16 = ALIGN(0x100000) + 0x10A000;

/* Overlay MIPS32 ISRAM text */
__isram32_text = .;
OVERLAY __base_addr_isram32      : AT (__isram32_text)
{
    .isram32 {sram_mips32.o(.text.*)}
}
__isram32_text_size = SIZEOF (.isram32) ;
. = __isram32_text + SIZEOF (.isram32) ;

/* Overlay MIPS16 ISRAM text */
__isram16_text = .;
OVERLAY __base_addr_isram16     : AT (__isram16_text)
{
    .isram16 {sram_mips16.o(.text.*)}
}
__isram16_text_size = SIZEOF (.isram16) ;
. = __isram16_text + SIZEOF (.isram16) ;

/* Overlay MIPS32 DSRAM data */
__dsram32_data = .;
OVERLAY __base_addr_dsram32    : AT (__dsram32_data)
{
    .dsram32 {*(.dsram32) sram_mips32.o(.rodata.*) sram_mips32.o(.data)}
}
__dsram32_data_size = SIZEOF (.dsram32) ;
. = __dsram32_data + SIZEOF (.dsram32) ;

/* Overlay MIPS16 DSRAM data */
__dsram16_data = .;
```

## 2 Example Files

```
OVERLAY __base_addr_dsram16      : AT (__dsram16_data)
{
    .dsram16 {*(.dsram16) sram_mips16.o(.rodata.*) sram_mips16.o(.data)}
}
__dsram16_data_size = SIZEOF (.dsram16) ;
. = __dsram16_data + SIZEOF (.dsram16) ;
```

The first line excludes the input `.text` sections of `sram_mips32.o` and `sram_mips16.o` from ending up in the output `.text` section of the application. The next four lines set the destination base addresses `__base_addr_isram32`, `__base_addr_isram16`, `__base_addr_dsram32`, and `__base_addr_dsram16` at which the four overlays will be copied into memory (by the application) and then executed.

The remaining lines create the two instruction-only overlay sections, `.isram32` and `.isram16`, and the two data-only overlay sections `.dsram32` and `.dsram16`. The instruction overlays will reside in the object at addresses `__isram32_text` and `__isram16_text`, and the data overlays will reside in the object at addresses `__dsram32_data` and `__dsram16_data`.

## 2.5 MIPSsim Configuration Files

The MIPSsim configuration file `M4K.cfg` defines how the core and simulator are configured prior to reset. The line excerpts from `M4K.cfg` (shown in the following subsection) are of particular interest for the example application, and they include comments that explain the configuration parameters.

### 2.5.1 Excerpts from M4K.cfg

```
# Load the file "sram_testrom.elf" into ISRAM
set ROM_FILE sram_testrom.elf

# Enable MIPS16 use
set CODE_COMPRESSION 1

# Identify core as M4K
set CPU_CORE 50

# Enable separate Instruction (ISRAM) & Data (DSRAM) interfaces
set DUAL_SRAM 1

# Read I/O Sub System (Memory) configuration file "MipsMemIntf-M4K.cfg"
set DEV_CFG MipsMemIntf-M4K.cfg
```

The MIPSsim I/O Sub System (Memory) is configured with the file `MipsMemIntf-M4K.cfg`. Excerpts from `MipsMemIntf-M4K.cfg` (shown in the following subsection) control how the memory subsystem is configured for MIPSsim.

### 2.5.2 Excerpts from MipsMemInf-M4K.cfg

```
# MEMORY ADDRESS MAP (physical addresses)
# 0x00000000-0x1fbfffff: SYSTEM RAM - Used for general purpose data and
#                               can only be accessed by loads & stores.
#
# 0x1fc00000-0x1fcfffff: SYSTEM ROM/RAM - Contains instructions and
#                               read-only data copied from the *.elf file.
```

```

#                               It is accessed by instruction fetches and
#                               load/stores (via redirection).
#
# 0x1fe00000-0x1fe07fff: ISRAM - Contains instructions (only) that
#                               are first copied from SYSTEM ROM/RAM (via
#                               redirection).
#
# 0x1fe08000-0x1fe0ffff: DSRAM - Contains data (only) and can only be
#                               accessed by loads & stores.
#
# ROM (0x1fc00000-0x1fcfffff)
MemIntf(ISRAM, SparseMemRam, 1, 0x1fc00000, 1M, MIPS_SparseMemRam)
# ISRAM (0x1fe00000-0x1fe07fff)
MemIntf(ISRAM, SparseMemRam, 1, 0x1fe00000, 32K, MIPS_SparseMemRam)
# DSRAM (0x1fe08000-0x1fe0ffff)
MemIntf(DSRAM, SparseMemRam, 2, 0x1fe08000, 32K, MIPS_SparseMemRam)
# RAM (0x00000000-0x1fbfffff)
MemIntf(DSRAM, SparseMemRam, 2, 0x00000000, 0x1fc00000, MIPS_SparseMemRam)
# REDIRECT DEVICES, Redirects loads & stores to instruction memory
MemIntf(DSRAM, REDIRECT, 2, 0x1fc00000, 1M, MIPS_SparseMemRam)
MemIntf(DSRAM, REDIRECT, 2, 0x1fe00000, 32K, MIPS_SparseMemRam)
# SPARSE MEMORY RAM
SparseMemRam(-1, , , 0, 0, 0, 0, 0)

```

The script file configures two ISRAM memory devices and two DSRAM memory devices. It also configures two REDIRECTION devices that allow the two ISRAM memories to be accessed by CPU load and store operations. DSRAM memory interface devices are only accessed by CPU load and store operations, and by default (with no REDIRECTION) ISRAM memory interface devices are only accessed by CPU instruction fetches.

The first line configures an ISRAM type memory interface device that resides at the boot address. In a typical system this might be “slower” ROM containing boot code and overlays that will be copied to “faster” ISRAM and DSRAM memories. The second and third lines configure the 32K ISRAM and 32K DSRAM type memory interface devices that are used for the test code overlays.

The fourth line configures a large DSRAM type memory interface device that’s used for the main stack, and various I/O devices. The fifth and sixth lines configure the REDIRECTION devices that allow loads and stores in those address ranges to be forwarded to the ISRAM memory interface devices. The last line in the file “SparseMemRam(-1, , , 0, 0, 0, 0, 0)” configures the memory timing (zero wait-state in this example) of the RAM device used for all of the ISRAM and DSRAM memory interface devices.

## 2.6 sde-gdb Command Script

The sde-gdb command script `gdbscript` can be invoked to run the example application in the debugger using a MIPSsim target. The command script could be modified for different targets or interactive debugging. Excerpts from `gdbscript` are shown in the following subsection.

## 2 Example Files

### 2.6.1 Excerpts from gdbscript

```
# Name of executable file to be debugged (must be *rom type image)
file sram_testrom

# Name of the configuration file to be loaded by MIPSsim
set mdi configfile M4K.cfg

# Use MDI to access MIPS Target/Device M4K, Cycle Accurate, Little Endian
target mdi 12:1

# Dynamically load sram_testrom into the running program
load

# Start debugged program
run

# Exit sde-gdb
quit
```

## 3 Build and Debugging Environment

This section covers the process of building the example application program and running it on the MIPSsim software simulator and the Malta hardware platform.

### 3.1 Recommended Tools and Platforms

A list of recommended tool and platform versions for developing the example code is shown in [Table 2](#).

**Table 2 Recommended Tools and Platforms**

Tool/Platform
Cygwin 1.5.11 or above with Microsoft NT, 2000 or XP.
RedHat Linux 7.1 or higher.
SDE 6.06.00 or higher.
MIPSsim 04.08.02 or higher.
Malta with M4K CoreFPGA3 card that includes m4k_sram_dual_switch_ec.v RTL module and at least 16K ISRAM and 16K DSRAM.
FS2 System Navigator (family) Probe with System Navigator Software version 2.1.7.7 or higher.

### 3.2 Installation and Setup of MIPSsim and SDE

Outlined below are the steps for installing and setting up the MIPSsim software simulator and the SDE software development environment.

1. Install and configure MIPSsim as described in the *MIPSsim™ Getting Started Guide for MIPS32™ M4K™ Cores*.
2. Install and configure SDE as described in the MIPS@SDE Programmers' Guide. Please remember to configure an "MDI fragment" that tells the sde-gdb debugger how to connect to the MIPSsim simulator.

### 3.3 Installing the Application

After the SDE and MIPSsim environments are setup, the source code can be installed using the following sequence:

1. Create "local" copies of the `.../sde/examples` and `.../sde/kit` directories.
2. Create a new directory called `.../sde/examples/sram`.
3. Copy all of the example application files into the new `.../sde/examples/sram` directory.

### 3.4 Building the Application (SDE 6.06.01 or higher)

Build the application using one or more of the following sde-make commands:

### 3 Build and Debugging Environment

```
$ cd ../sde/examples/sram
$ sde-make SBD=MSIM32L
$ sde-make SBD=MSIMM4KL
$ sde-make SBD=MALTA32L
$ sde-make SBD=MALTAM4KL
```

The MSIM32L and MSIMM4KL options build the `sram_testrom` elf file executable for MIPSsim, and the MALTA32L and MALTAM4KL options build the `sram_testram.s3` S-record file executable for Malta. The MSIM32L and MALTA32L options will create standard MIPS32 executables. The MALTAM4KL and MSIMM4KL options will create size optimized (MIPS16) executables (including `sram_test.o`), but this will not affect the size or performance of the test objects `sram_mips32.o` and `sram_mips16.o`. The source files `sram_mips32.c` and `sram_mips16.c` have compiler attributes that override the “make” command line options.

## 3.5 Building the Application (SDE below 6.06.01)

For SDE versions below 6.06.01 a new make file needs to be created and edited to add the local linker script file as described below:

1. Change to the source code directory and create the make file `SDEmakefile`.

```
$ cd ../sde/examples/sram
$ sde-make SBD=MSIM32L SDEmakefile
```

2. Copy the file `SDEmakefile` to the file `Makefile`.
3. Edit the file `Makefile` and replace the line “`-Wl,--script=$(KITDIR)/share/sde.ld`” with the line

```
“-Wl,--script=elf32mipssde.xn” in the sram_testram: & sram_testsa: targets.
```

4. Build the application

```
$ sde-make
```

To change the target platform edit the file `Makefile` and change the “SBD” line to something like `SBD=MALTA32L`. Then rerun the `sde-make` command.

## 3.6 Debugging on MIPSsim

Use the following sequence to run the executable on MIPSsim using the `sde-gdb` debugger:

```
$ cd ../sde/examples/sram
$ sde-gdb --command=gdbscript
```

The `sde-insight` graphical debugger may also be used with the following menu settings:

```
File -> Open: sram_testrom
File -> Target Settings -> Target: MDI Connection
File -> Target Settings -> Device: M4K (Cycle):M4K LE
File -> Target Settings -> Config: M4K.cfg
```

## 3.7 Debugging on Malta

Use the following sequence to run the executable on Malta via Yamon:

1. Copy the executable file sram\_testram.s3 to your tftp server.
2. Load the executable to Malta.

```
YAMON> load tftp://'tftp-ip-address'/'your-path'/sram_testram.s3
```

3. Run the executable on Malta.

```
YAMON> go
```

The FS2 probe may also be used to debug the code through the EJTAG interface. Debugging can be directly controlled from the FS2 System Navigator Console or through other applications with MDI interfaces such as: sde-gdb, sde-insight or MIPS IDE (eclipse). Prior to debugging you should enter the following command in the System Navigator Console:

```
config IsramConfigBase 0x1e400000
```

This tells the FS2 probe where the ISRAM control registers are located enabling it to copy code or software breakpoints into ISRAM.

## 3.8 Program Output

The program should generate similar output for Malta (shown below) and MIPSsim.

```
YAMON> go
Running sram_test

#####

TESTING IMAGE FROM SYSTEM MEMORY (SDRAM)

-> Copying MIPS32 Overlay to RAM
    Text Addr: 0x80300000
    Data Addr: 0x80308000
    Text Size: 388 bytes
    Data Size: 1024 bytes

-> Copying MIPS16 Overlay to RAM
    Text Addr: 0x80302000
    Data Addr: 0x8030A000
    Text Size: 268 bytes
    Data Size: 1024 bytes

-> Executing MIPS32 Overlay
    Cycle Count: 485966 clocks

-> Executing MIPS16 Overlay
    Cycle Count: 458294 clocks

#####

TESTING IMAGE FROM ISRAM & DSRAM
```

### 3 Build and Debugging Environment

```
-> Copying MIPS32 Overlay to RAM
    Text Addr: 0x80300000
    Data Addr: 0x80308000
    Text Size: 388 bytes
    Data Size: 1024 bytes

-> Copying MIPS16 Overlay to RAM
    Text Addr: 0x80302000
    Data Addr: 0x8030A000
    Text Size: 268 bytes
    Data Size: 1024 bytes

-> Executing MIPS32 Overlay
    Cycle Count: 45116 clocks

-> Executing MIPS16 Overlay
    Cycle Count: 71758 clocks

#####

SUMMARY REPORT

-> Text size difference from MIPS32 to MIPS16
    388 bytes to 268 bytes: 30.93% change

-> Performance difference from MIPS32 to MIPS16 using ISRAM/DSRAM
    45116 clocks to 71758 clocks: -37.13% change

-> Performance difference from System RAM to ISRAM/DSRAM for MIPS32
    485966 clocks to 45116 clocks: 977.15% change

-> Performance difference from System RAM to ISRAM/DSRAM for MIPS16
    458294 clocks to 71758 clocks: 538.67% change

User application returned with code = 0x00000000
YAMON>
```

### 3.9 Using ISRAM/DSRAM: Malta vs. MIPSsim

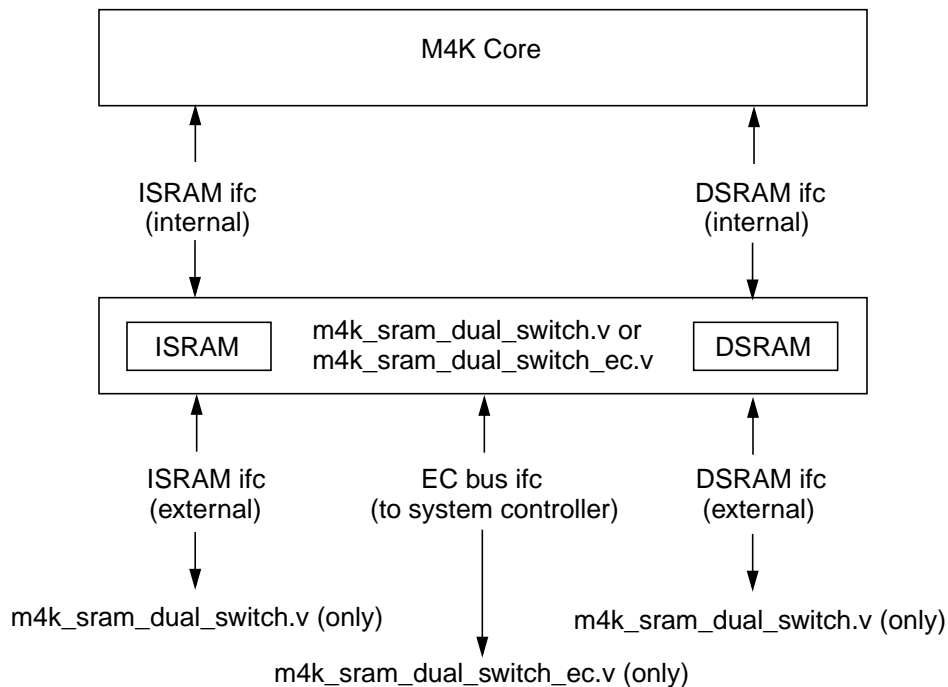
On Malta the ISRAM and DSRAM are “dynamically” controlled by software with the `m4k_sram_dual_switch_ec.v` hardware module. The application first disables ISRAM/DSRAM and copies and executes the test code overlays from external system memory. Then the application enables ISRAM/DSRAM and copies and runs the test code overlays from internal ISRAM/DSRAM. Finally the application displays the performance differences between slower system memory and faster ISRAM/DSRAM.

ISRAM and DSRAM are “statically” configured by MIPSsim before simulation using the settings in the memory configuration file `MipsMemIntf-M4K.cfg`. Thus ISRAM and DSRAM are always enabled. Under MIPSsim, the application copies and runs two (identical) iterations of the test code overlays from ISRAM/DSRAM. The application will display no performance differences between the two iterations. Also, the application generates a warning message indicating that the “hardware” `m4k_sram_dual_switch*.v` module isn’t implemented.

## 4 Programming the m4k\_sram\_dual\_switch\*.v

This section discusses programming the `m4k_sram_dual_switch.v` (for SRAM interface) and `m4k_sram_dual_switch_ec.v` (for EC bus interface) customer example modules. These files are located in the `$MIPS_PROJECT/proc/design/modules/user` directory of the M4K deliverable and they may be modified by the customer to add application-specific functionality. All Malta/CoreFPGA3 hardware implementations include the deliverable version of the `m4k_sram_dual_switch_ec.v` module. A block diagram of the system interface for the `m4k_sram_dual_switch.v` or `m4k_sram_dual_switch_ec.v` modules is shown in Figure 1.

**Figure 1 ISRAM/DSRAM Interface Block Diagram**



### 4.1 Theory of Operation

Following reset, the ISRAM and DSRAM in the `m4k_sram_dual_switch*.v` module are disabled. So all memory operations from the core bypass the ISRAM and DSRAM to the EC bus interface or SRAM interface. Software can enable or disable the ISRAM and DSRAM and configure their physical base address decoding. The base addresses of the ISRAM and DSRAM may be set at any physical address, so long as it's aligned with the physical size of the RAM, namely, 16K, 32K, etc. When the DSRAM is enabled, all CPU loads and stores within its configured address range go to the DSRAM. When the ISRAM is enabled, all CPU instruction fetches within its configured address range come from the ISRAM. Also, CPU loads and stores can go to the ISRAM if REDIRECTION is enabled by setting the *Redirection (R)* bit in the *ISRAM Configuration* register). Memory references outside the configured address ranges of ISRAM and DSRAM will be forwarded to system memory via the EC interface bus or SRAM interface.

## 4.2 Control Register Summary

Table 3 lists the control registers in order of physical address.

**Table 3 Control Registers**

Register Address (Physical)	Register Name	Function
0x1e400000	<i>ISRAM Configuration</i>	ISRAM Control
0x1e400010	<i>ISRAM Mask</i>	ISRAM Size and Alignment
0x1e400020	<i>DSRAM Configuration</i>	DSRAM Control
0x1e400030	<i>DSRAM Mask</i>	DSRAM Size and Alignment

## 4.3 ISRAM Configuration Register (0x1e400000)

The ISRAM is controlled by the *ISRAM Configuration* register located at physical address 0x1e400000. It contains a 22-bit field for setting the I-side physical base address. This is the physical starting address at which the ISRAM will be decoded, and it must be loaded with an address that is aligned with the block size of the ISRAM. The *Enable* (*E*) bit enables the decoding of CPU fetches from ISRAM. The *Redirection* (*r*) bit enables CPU loads and stores to ISRAM so instructions may be copied to it. Figure 2 shows the format of the *ISRAM Configuration* register; Table 4 describes the *ISRAM Configuration* register fields.

**Figure 2 ISRAM Configuration Register Format**



**Table 4 ISRAM Configuration Register Field Descriptions**

Fields		Description	Read / Write	Reset State
Name	Bits			
<i>I-Side Base Address</i>	31:10	I-side physical base address (must be aligned with block size).	R/W	0
0	9:2	Must be written with zero; returns zero on read.	0	0
R	1	Redirect D-side if hitting I-side address range (and not hitting D-side address range).	R/W	
E	0	Enable the ISRAM.	R/W	0

## 4.4 ISRAM Mask Register (0x1e400010)

The *ISRAM Mask* register is located at physical address 0x1e400010. It contains a 22-bit read-only mask used by software to mask and align the I-Side Address in the *ISRAM Configuration* register.

Figure 3 shows the format of the *ISRAM Mask* register; Table 5 describes the *ISRAM Mask* register fields.

**Figure 3 ISRAM Mask Register Format**



**Table 5 ISRAM Mask Register Field Descriptions**

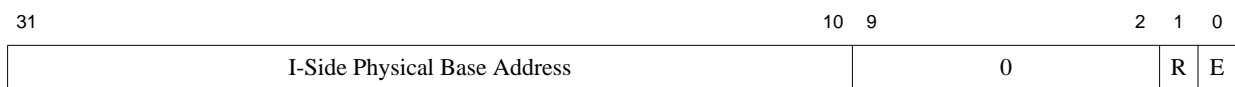
Fields		Description	Read / Write	Reset State
Name	Bits			
<i>I-Side Address Mask</i>	31:10	I-side address mask 32K=0xffff8000, 64K=0xffff0000.	R	0
0	9:0	Returns zero on read.	0	0

## 4.5 DSRAM Configuration Register (0x1e400020)

The DSRAM is controlled by the *DSRAM Configuration* register located at physical address 0x1e400020. It contains a 22-bit field for setting the D-side physical base address. This is the physical starting address at which the DSRAM will be decoded, and it must be loaded with an address that is aligned with the block size of the DSRAM. The *Enable (E)* bit enables the decoding of CPU loads and stores to DSRAM. The read-only *Redirection (R)* bit indicates a D-side access has been redirected to I-side.

Figure 4 shows the format of the *DSRAM Configuration* register; Table 6 describes the *DSRAM Configuration* register fields.

**Figure 4 DSRAM Configuration Register Format**



**Table 6 DSRAM Configuration Register Field Descriptions**

Fields		Description	Read / Write	Reset State
Name	Bits			
<i>I-Side Base Address</i>	31:10	I-side physical base address (must be aligned with block size),	R/W	0
0	9:2	Must be written with zero; returns zero on read.	0	0
R	1	Redirect D-side if NOT hitting D-side address range (if only access to external world from I-side).	R	0
E	0	Enable the DSRAM.	R/W	0

## 4.6 DSRAM Mask Register (0x1e400030)

The *DSRAM Mask* register is located at physical address 0x1e400030. It contains a 22-bit read-only mask used by software to mask and align the D-Side Address in the *DSRAM Configuration* register.

Figure 5 shows the format of the *DSRAM Mask* register; Table 7 describes the *DSRAM Mask* register fields.

**Figure 5 DSRAM Mask Register Format**



**Table 7 DSRAM Mask Register Field Descriptions**

Fields		Description	Read / Write	Reset State
Name	Bits			
<i>D-Side Address Mask</i>	31:10	D-side address mask 32K=0xffff8000, 64K=0xffff0000.	R	0
0	9:0	Returns zero on read.	0	0

## 5 References

This appendix lists other documents available from MIPS Technologies, Inc. that are referenced elsewhere in this document. These documents may be included in the `$MIPS_HOME/$MIPS_CORE/doc` area of a typical *Core-Name* soft or hard core release, or in some cases may be available on the MIPS web site, <http://www.mips.com>.

1. MIPS® SDE 6.x Programmers' Guide  
MIPS Document: MD00428
2. MIPSsim™ Software Release Notes for M4K™ Cores  
MIPS Document: MD00269
3. MIPSsim™ Software Getting Started Guide for M4K™ Cores  
MIPS Document: MD00270
4. MIPS32® M4K™ Processor Core Datasheet  
MIPS Document: MD00247
5. MIPS32® M4K™ Processor Core Integrator's Guide  
MIPS Document: MD00248
6. MIPS32® M4K™ Processor Core Software Users Manual  
MIPS Document: MD00249
7. Malta™ Users Manual  
MIPS Document: MD00048
8. MIPS® Malta™-R Development Platform User's Manual  
MIPS Document: MD00627
9. CoreFPGA™ 3 Users Manual  
MIPS Document: MD00481
10. EC™ Interface Specification  
MIPS Document: MD00052
11. YAMON™ Users Manual  
MIPS Document: MD00008
12. The GNU linker Manual “Using ld, version 2”

## 6 Revision History

Change bars (vertical lines) in the margins of this document indicate significant changes in the document since its last release. Change bars are removed for changes that are more than one revision old.

This document may refer to Architecture specifications (for example, instruction set descriptions and EJTAG register definitions), and change bars in these sections indicate changes since the previous version of the relevant Architecture document.

<b>Revision</b>	<b>Date</b>	<b>Description</b>
1.00	October 10, 2007	Initial external release.
2.00	April 26, 2007	Added SDE 6.06.01 update.

Copyright © 2007 MIPS Technologies, Inc. All rights reserved.

Unpublished rights (if any) reserved under the copyright laws of the United States of America and other countries.

This document contains information that is proprietary to MIPS Technologies, Inc. ("MIPS Technologies"). Any copying, reproducing, modifying or use of this information (in whole or in part) that is not expressly permitted in writing by MIPS Technologies or an authorized third party is strictly prohibited. At a minimum, this information is protected under unfair competition and copyright laws. Violations thereof may result in criminal penalties and fines.

Any document provided in source format (i.e., in a modifiable form such as in FrameMaker or Microsoft Word format) is subject to use and distribution restrictions that are independent of and supplemental to any and all confidentiality restrictions. UNDER NO CIRCUMSTANCES MAY A DOCUMENT PROVIDED IN SOURCE FORMAT BE DISTRIBUTED TO A THIRD PARTY IN SOURCE FORMAT WITHOUT THE EXPRESS WRITTEN PERMISSION OF MIPS TECHNOLOGIES, INC.

MIPS Technologies reserves the right to change the information contained in this document to improve function, design or otherwise. MIPS Technologies does not assume any liability arising out of the application or use of this information, or of any error or omission in such information. Any warranties, whether express, statutory, implied or otherwise, including but not limited to the implied warranties of merchantability or fitness for a particular purpose, are excluded. Except as expressly provided in any written license agreement from MIPS Technologies or an authorized third party, the furnishing of this document does not give recipient any license to any intellectual property rights, including any patent rights, that cover the information in this document.

The information contained in this document shall not be exported, reexported, transferred, or released, directly or indirectly, in violation of the law of any country or international law, regulation, treaty, Executive Order, statute, amendments or supplements thereto. Should a conflict arise regarding the export, reexport, transfer, or release of the information contained in this document, the laws of the United States of America shall be the governing law.

The information contained in this document constitutes one or more of the following: commercial computer software, commercial computer software documentation or other commercial items. If the user of this information, or any related documentation of any kind, including related technical data or manuals, is an agency, department, or other entity of the United States government ("Government"), the use, duplication, reproduction, release, modification, disclosure, or transfer of this information, or any related documentation of any kind, is restricted in accordance with Federal Acquisition Regulation 12.212 for civilian agencies and Defense Federal Acquisition Regulation Supplement 227.7202 for military agencies. The use of this information by the Government is further restricted in accordance with the terms of the license agreement(s) and/or applicable contract terms and conditions covering this information from MIPS Technologies or an authorized third party.

MIPS, MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS-3D, MIPS16, MIPS16e, MIPS32, MIPS64, MIPS-Based, MIPSsim, MIPSpro, MIPS Technologies logo, MIPS-VERIFIED, MIPS-VERIFIED logo, 4K, 4Kc, 4Km, 4Kp, 4KE, 4KEc, 4KEm, 4KEp, 4KS, 4KSc, 4KSd, M4K, 5K, 5Kc, 5Kf, 24K, 24Kc, 24Kf, 24KE, 24KEc, 24KEf, 34K, 34Kc, 34Kf, 74K, 74Kf, 74Kc, 1004K, 1004Kc, 1004Kf, R3000, R4000, R5000, ASMACRO, Atlas, "At the core of the user experience.", BusBridge, Bus Navigator, CLAM, CorExtend, CoreFPGA, CoreLV, EC, FPGA View, FS2, FS2 FIRST SILICON SOLUTIONS logo, FS2 NAVIGATOR, HyperDebug, HyperJTAG, JALGO, Logic Navigator, Malta, MDMX, MED, MGB, OCI, PDtrace, the Pipeline, Pro Series, SEAD, SEAD-2, SmartMIPS, SOC-it, System Navigator, and YAMON are trademarks or registered trademarks of MIPS Technologies, Inc. in the United States and other countries.

All other trademarks referred to herein are the property of their respective owners.

Template: nW1.03, Built with tags: 2B

Working with Dual SRAM for MIPS32® M4K™ Cores Application Note, Revision: 02.00

**Copyright © 2007 MIPS Technologies Inc. All rights reserved.**