Programmer's Guide for the 8051 Microprocessor Embedded in the BCM53125

53125-PG104-R

5300 California Avenue • Irvine, CA 92617 • Phone: 949-926-5000 • Fax: 949-926-5203

July 26, 2011

Revision History

Revision	*Date*	*Change Description*
53125-PG104-R	07/26/11	**Updated:** • "Building the Software" on page 24 **Added:** • "A Customization Region in External Serial Flash" on page 17
53125-PG103-R	05/12/11	**Updated:** • "Building the Software" on page 20 • Added note in "Software Functionality" on page 21 • Added note in "Service Abstraction Layer Functions" on page 26 • Table 8: "Customization Files," on page 32
53125-PG102-R	11/16/10	**Added:** • Section: "8051 Image Upgraded by External CPU," on page 14
53125-PG101-R	09/06/10	**Added:** • "Resource Arbitration" on page 13
53125-PG100-R	05/18/10	Initial release

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

Table of Contents

List of Figures

List of Tables

About This Document

Purpose and Audience

This document includes boot-up, source and tool chain, and software functionality information for the BCM53125 device and is intended for system/software architects.

Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use.

For a comprehensive list of acronyms and other terms used in Broadcom documents, go to: http://www.broadcom.com/press/glossary.php.

Document Conventions

The following conventions may be used in this document:

Convention	*Description*
Bold	User input and actions: for example, type **exit**, click **OK,** press **Alt+C**
`Monospace`	Code: `#include <iostream>` HTML: `<td rowspan = 3>` Command line commands and parameters: `wl [-l] <command>`
< >	Placeholders for *required* elements: enter your <username> or `wl <command>`
[ ]	Indicates *optional* command-line parameters: `wl [-l]` Indicates bit and byte ranges (inclusive): [0:3] or [7:0]

References

The references in this section may be used in conjunction with this document.

Note: Broadcom provides customer access to technical documentation and software through its Customer Support Portal (CSP) and Downloads and Support site (see Technical Support).

For Broadcom documents, replace the "xx" in the document number with the largest number available in the repository to ensure that you have the most current version of the document.

Document (or Item) Name	*Number*	*Source*
Other Items		
[1] software source code release downloading instructions	–	Your Broadcom FAE

Technical Support

Broadcom provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates through its customer support portal (https://support.broadcom.com). For a CSP account, contact your Sales or Engineering support representative.

In addition, Broadcom provides other product support through its Downloads and Support site (http://www.broadcom.com/support/).

Section 1: Overview

The BCM53125 has integrated the 8051 microprocessor and supports the serial Flash interface. The embedded 8051 is only for internal register configuration access. The main features of the BCM53125 embedded 8051 microprocessor are as follows:

- 16 KB of ROM
- 32 KB of RAM

Figure 1 shows the block diagram of the BCM53125 8051 microprocessor.

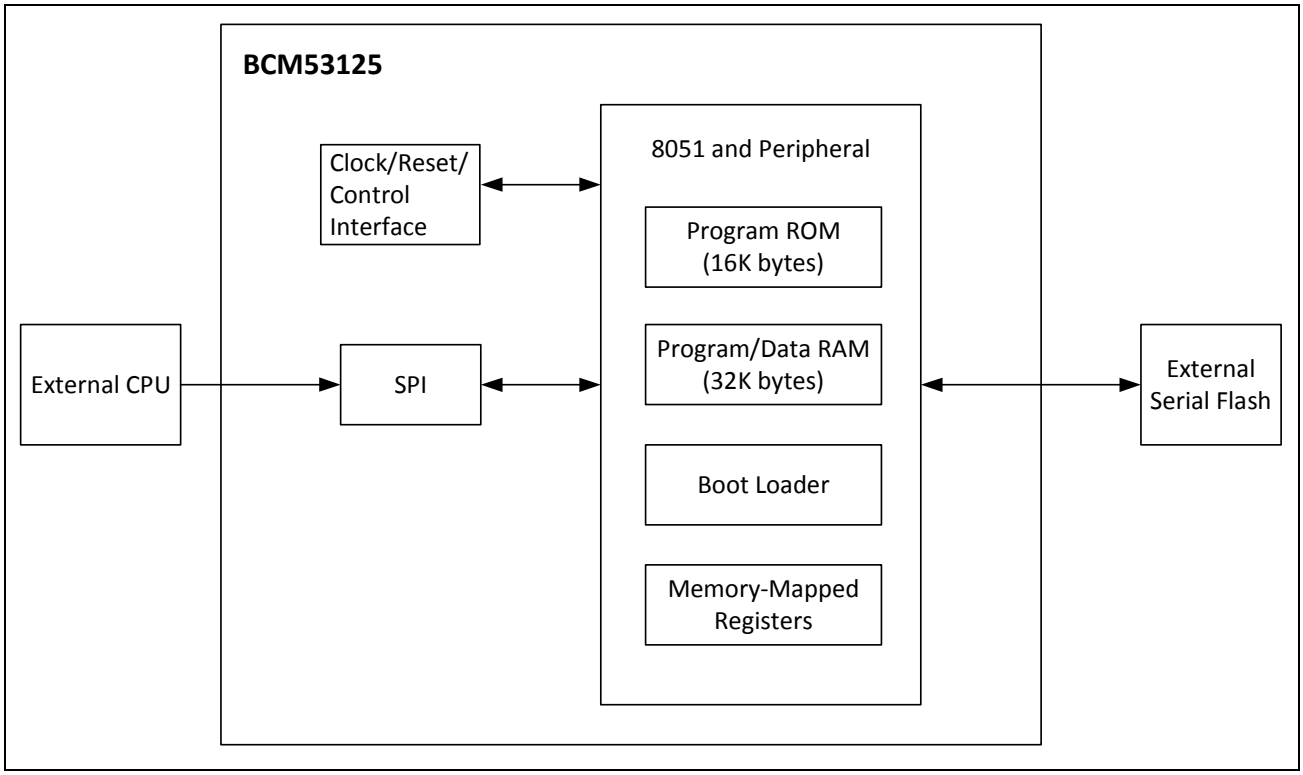


Figure 1: BCM53125 8051 Microprocessor Subsystem Architecture

Section 2: Boot-Up and Memory Mapping

8051 Boot-Up Sequence

The 8051 boots up from ROM or RAM depending on the existence of an external serial Flash device. When the reset is released, the boot loader takes the precedence of the boot over the 8051, detects the existence of the external serial Flash, and starts to load the contents from the serial Flash into the RAM if the detection is successful. The boot-up sequence is shown in Figure 2.

Note: During the loading, the access of external CPU is disabled.

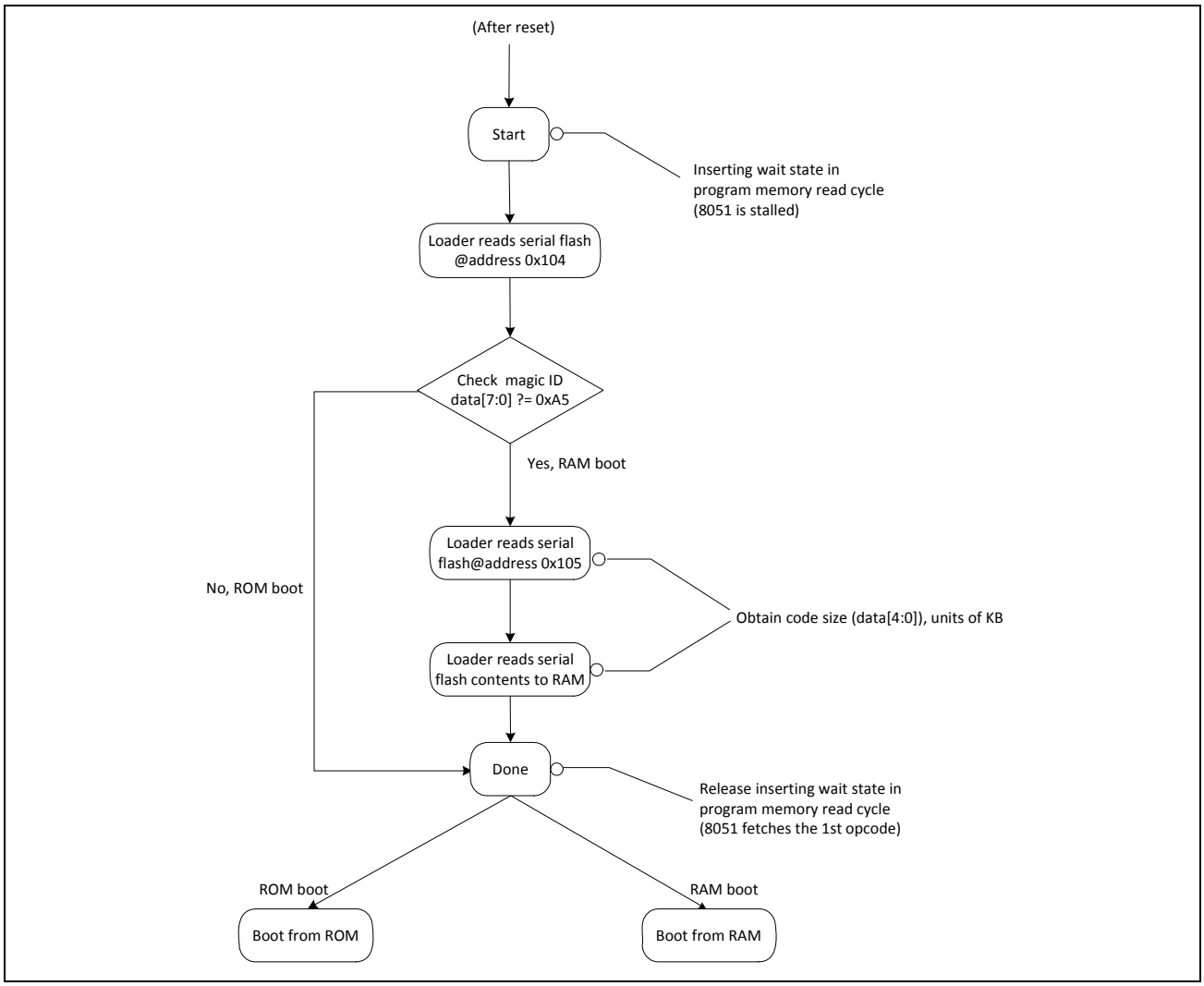


Figure 2: 8051 Boot-Up Sequence

Booting Operation

The 8051 of the BCM53125 separates program and data memory space and is naturally designed to support the memory space up to 64 KB, with 16-bit addresses for program and data memory, respectively.

The BCM53125 is equipped with 16 KB of ROM and 32 KB of RAM for program and data, respectively. Generally, the ROM stores the instruction codes in the program memory space and the RAM stores data in the data memory space. However, if the external serial Flash is present and the boot code is from the external serial Flash, the RAM will be partitioned into two segments. The lower address space of the RAM is used for program memory and the reset space of 32 KB is used for data space. Figure 3 shows the ROM boot memory amp and Figure 4 shows the RAM boot memory map.

Note: Software program must not jump over the code size or access the empty or reserved space.

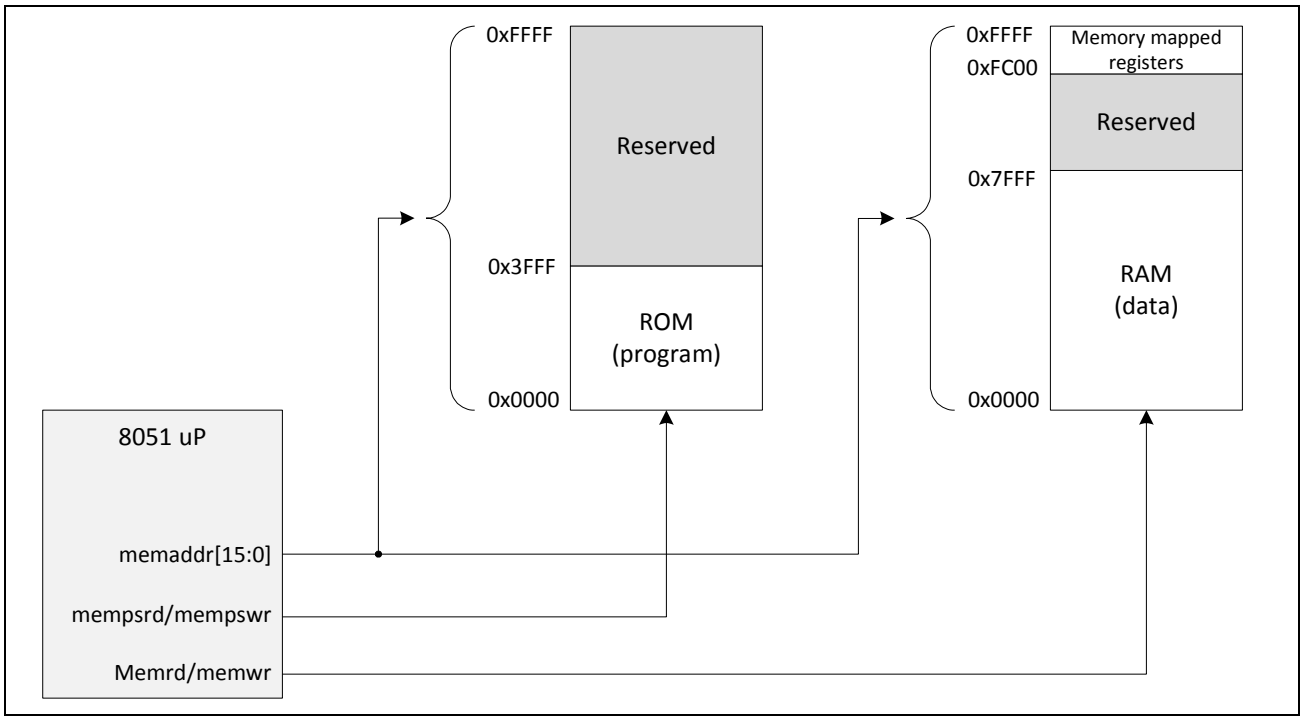


Figure 3: Memory Map for Booting from ROM

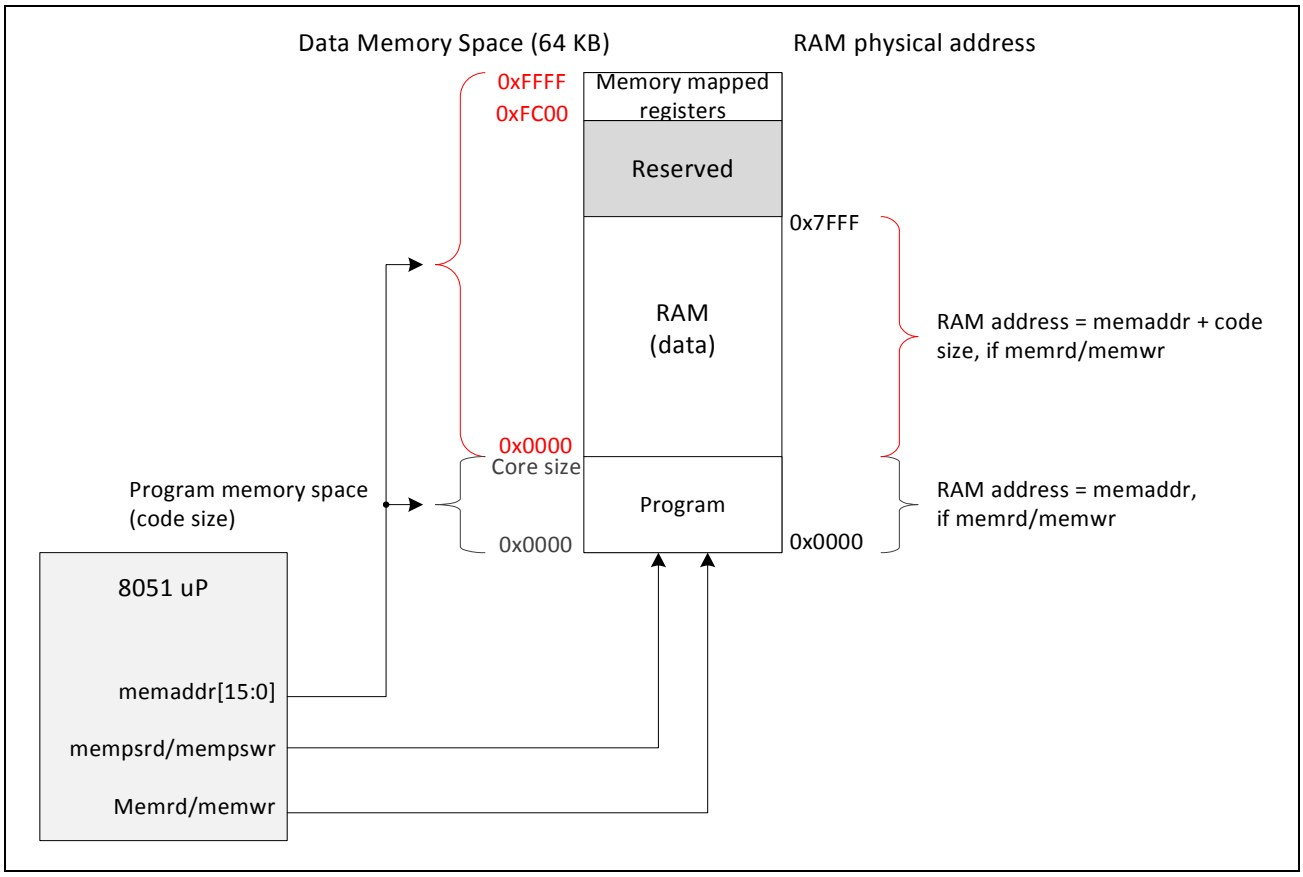


Figure 4: Memory Map for Booting from RAM

Resource Arbitration

The BCM53125 on-chip control and status registers (Switch and Gphy registers) are shared resources. They allow to be accessed by both external CPU and internal 8051.

When the BCM53125 internal 8051 is enabled and both the external CPU and the internal 8051 want to access these registers, arbitration is required. The BCM53125 provides a CPU resource arbiter for software to manage. Table 1 shows the resource arbiter register descriptions.

- The external CPU must assert the REQ signal and be granted access by the arbiter before accessing the shared resources registers. This is achieved by setting the EXT_CPU_REQ bit of the CPU_RESOURCE_ARBITER Register (Page 00h: Address A0h) to assert the REQ signal. The external CPU must then wait until the EXT_CPU_GNT bit is set by the resource arbiter.
- The external CPU must deassert the REQ signal after exiting the shared resources access by setting the EXT_CPU_REQ bit of the CPU_RESOURCE_ARBITER Register (Page 00h: Address A0h).

Table 1: CPU_RESOURCE_ARBITER Register (Page 00h: Address A0h)

Bit	*Name*	*R/W*	*Description*	*Default*
7:2	Reserved	RO	Reserved	0
1	EXT_CPU_REQ	R/W	REQ signal for external CPU. When the external CPU needs to access the shared resources registers, it asserts the REQ signal for arbitration. When the access is granted by the arbiter, the GNT signal is asserted to inform the requester. The requester continues asserting the REQ signal to occupy the arbiter. When the access done, the requester deasserts the REQ signal to release to the other requester. 1: Assert 0: Deassert	0
0	EXT_CPU_GNT	RO	GNT signal for external CPU. 1 = Access granted by arbiter.	0

8051 Image Upgraded by External CPU

The BCM53125 internal 8051 image can be upgraded by external CPU through SPI interface using the following procedure:

1. Put the 8051 into reset state by setting the IN_CPU_RST field of IN_CPU_CTRL register (page 0x0 offset 0x90).
2. Disable the MAC low power mode if it is enabled, as follows
   - Set Reg(Page 00h: Address dch) = 0xC0.
   - Set Reg(Page 00h: Address dch) = 0x00.
   - Set Reg(Page 00h: Address dfh) = 0x00.
3. Write the entire image into the 8051 RAM section using MEM registers (page 0x8) as follows:
   - Select the RAM type of 8051 RAM by Reg(Page 08h: Address 00h).
   - Write each memory entry with 8 bytes of image data by Reg(Page 08h: Address 08h).
   - Set the write operation and data index by Reg(Page 08h: Address 01h).
4. Set the code size (units = KB) by IN_CPU_CTRL register first (page 0x0 offset 0x90).
5. Clear the 8051 reset state and set the 8051 to start running at RAM address 0x0.

The internal 8051 will start running with the new image.

A Customization Region in External Serial Flash

For BCM53125, there is a serial interface to read the register level settings of the customization features and functions from an external EEPROM device.

In addition to the EEPROM device, the same mechanism can now be supported in a serial Flash device. The external serial Flash customization region for EEPORM shown in Table 2.

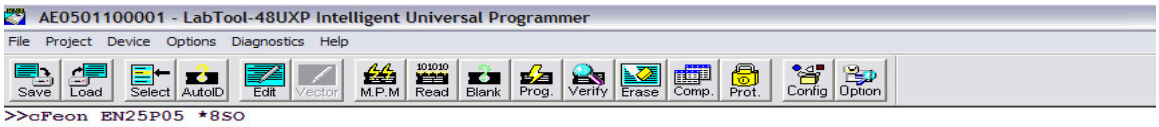
Table 2: Register Level Customization Region

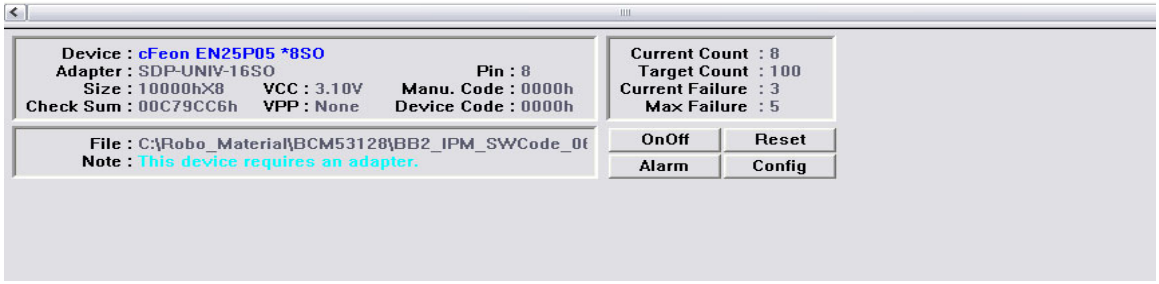
Supported Devices	*Starting Offset in Serial Flash*	*Total Size*	*Data Format*
BCM53125	0x5000	4K Bytes (0x1000)	The same binary format of EEPROM compiler for ROBO devices

Programming Step

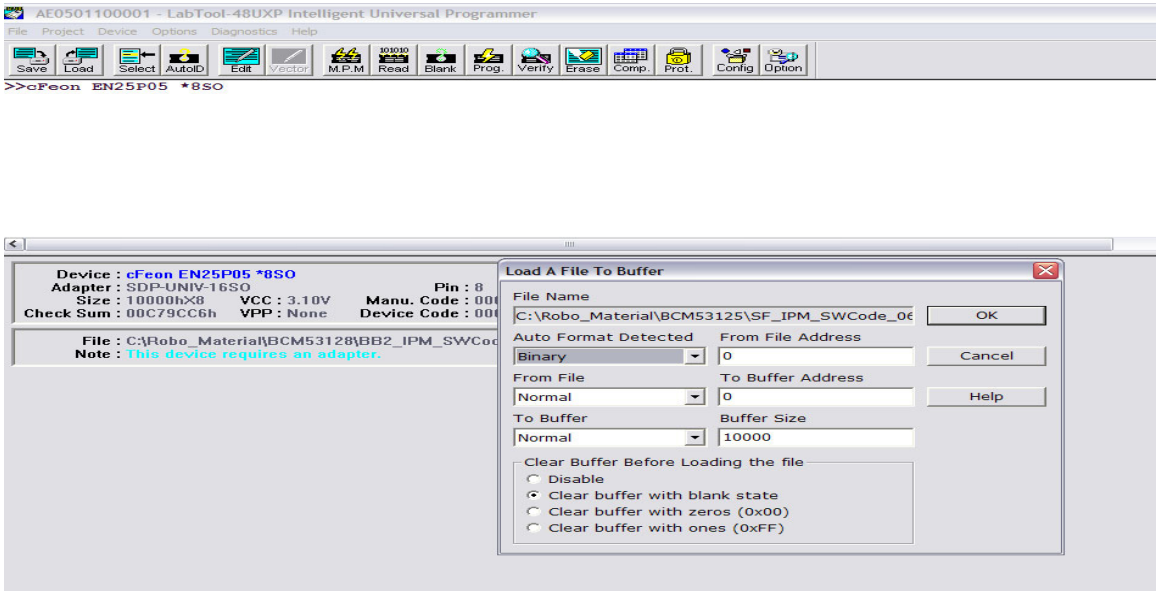
To program the BCM53125 8051 software image and EEPROM binary code into the serial Flash device through the programmer, use the following steps for an example of software image "sfpower.bin."

1. Select the serial Flash device "EN25P05*8SO" or compatible in programmer.

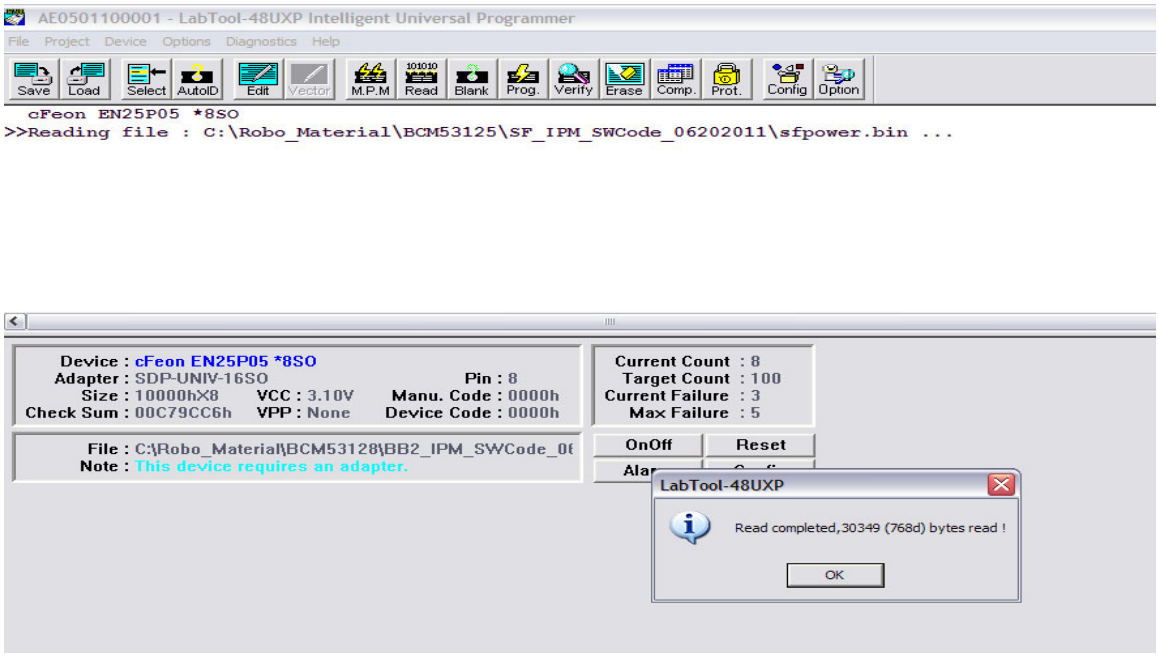




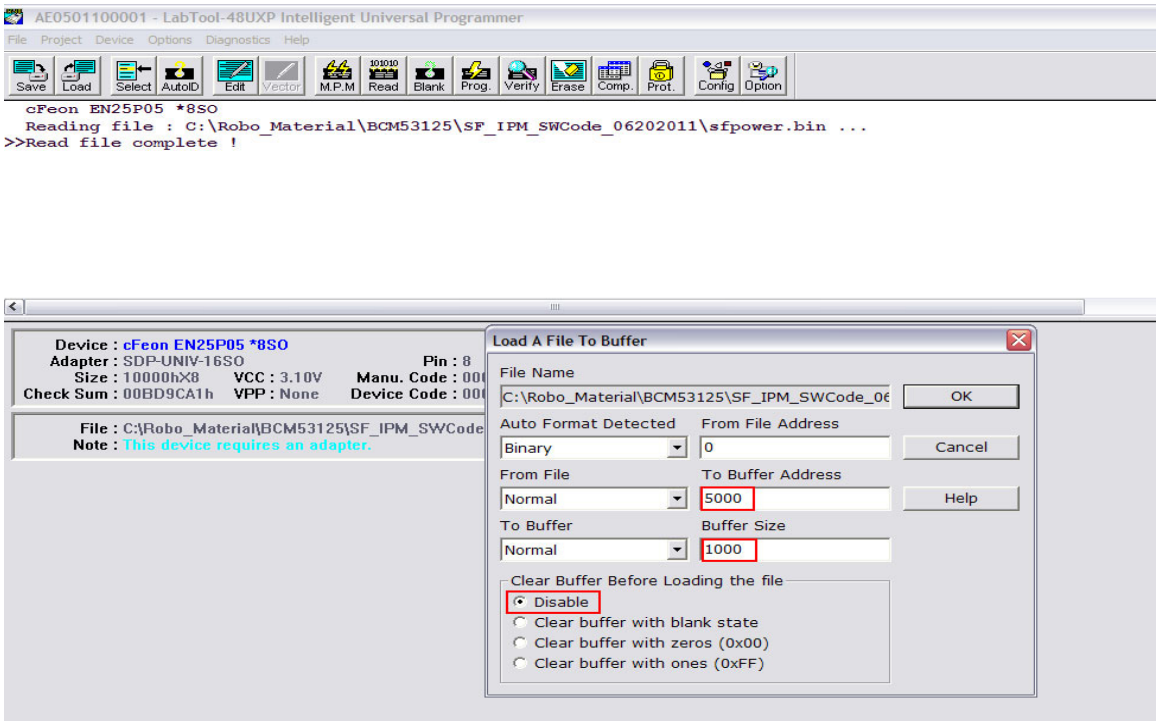
2. Load the software image into the programmer.



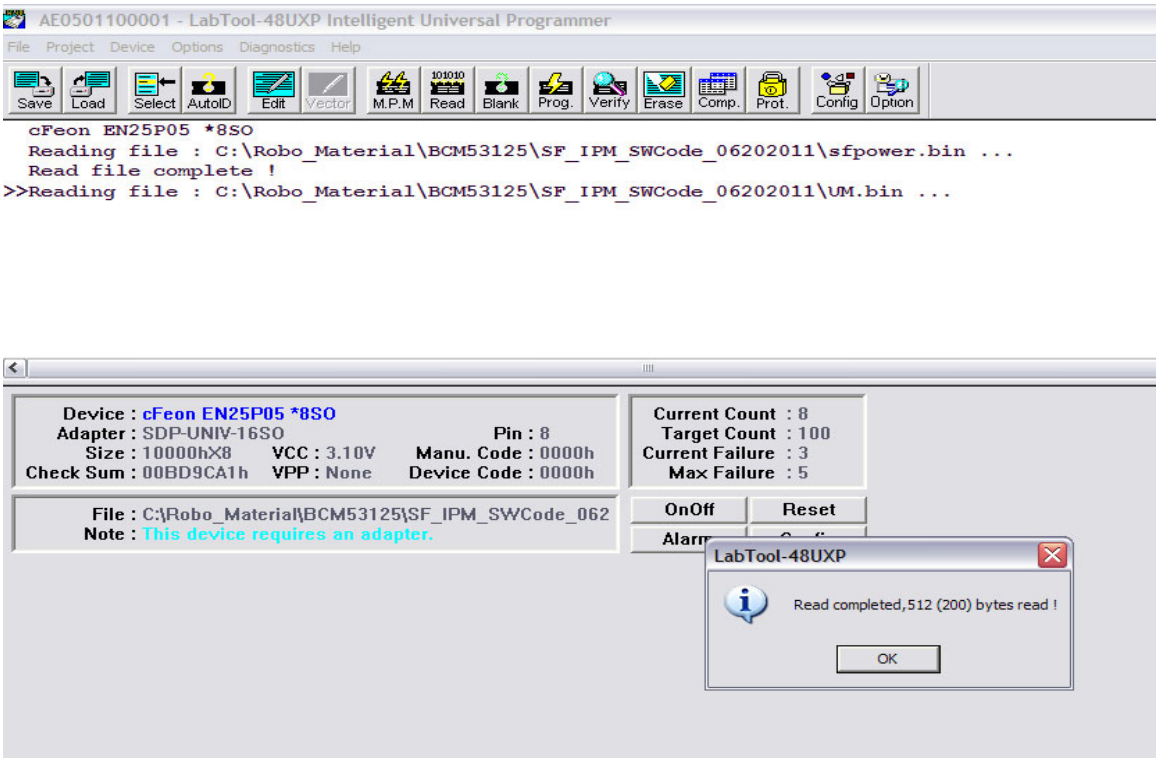
3. The software image read is complete in the programmer.



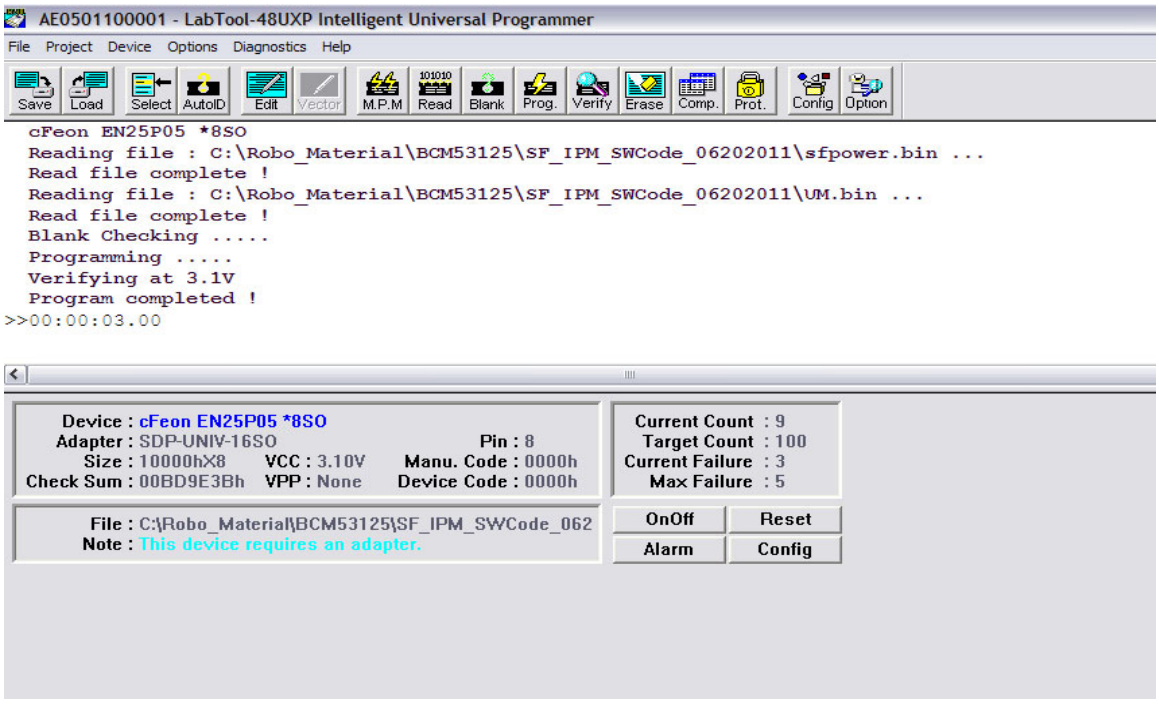
4. Load the EEPROM binary code in the programmer and follow the buffer settings as shown below.



5. The EEPROM binary code read is complete in the programmer.



6. **Click** the Prog. icon of the programmer to program the software image and EEPROM binary code into the serial Flash device.



Section 3: Source and Tool Chain

Software

The required software contains the software source code release. Contact your local Broadcom Field Applications Engineer (FAE) to obtain the downloading instructions.

Tool Chain

The required tool chain enables the 8051 compiler to build the released software for Microsoft® Windows® PCs. Broadcom recommends that the Keil™ µVision4 embedded software development tool be used to build the released software code for this environment's tool chain.

Section 4: Source Directory Layout

Top-Level Directory

The top-level directory contains subdirectories listed in Table 3.

Table 3: Top-Level Directory

Subdirectory	*Contains*
`um/`	Source code for unmanaged system

The top-level directory contains subdirectories listed in Table 4.

Table 4: Top-Level Subdirectories

Subdirectory	*Contains*
`include/`	All include files necessary to compile the programs using unmanaged code ***Note:*** Include this directory in the search path for include files (see "include/" on page 22).
`src/`	All system-independent source code
`systems/`	All system-dependent source code

Top-Level Subdirectories

include/

This directory contains a number of subdirectories where the header files for various parts of the code are located as listed in Table 5.

Table 5: include/ Subdirectories

Subdirectory	*Contains*
`8051/`	8051 access register header file
`appl/`	Header files for CLI command
`soc/`	Header files including MAC and PHY driver
`utils/`	Header files for utility for application

src/

This directory contains platform-independent code and is organized in subdirectories, each of which corresponds to the major components of the unmanaged code.Table 6 lists `src/` subdirectories.

Table 6: src/ Subdirectories

Subdirectory	*Contains*
appl/	Various applications written on top of kernel
board/	System-dependent information on top of driver
driver/	Source code for chip's driver
kernel/	Main function and TX/RX application
sal/	Used for all layers including console, timer, debug, assert, allocation, and memory/string operations
utils/	Source files for utility for application

systems/

This directory contains platform-dependent code. The unmanaged code provides support for a number of Broadcom reference platforms and loadable binary images can be built for them. Table 7 lists systems/ subdirectories.

Table 7: systems/ Subdirectories

Subdirectory	*Contains*
bcm953125/	Build environment

Section 5: Building the Software

The following steps are used to build the software code in the Keil Software µVision4 tool chain:

1. Load the project file (see Table 8) to open the environment.
2. For the compile step, select the Project->Build target for the Keil software uVision4 tool.
3. Compile and link to generate the binary file which is located in `\um\systems\bcm953125\proj\build`. For the binary file name refer to the Output field in Table 8.

Table 8: Options For The Power Saving Customization

Devices	*Available Keil/8051 Options*	*Output*	*Description*
BCM53125	bcm53125_power.uvproj	sfpower.bin	Standard power saving option.
	bcm53125_app.uvproj	sfapp.bin	Customer's option for power saving features.

Section 6: Software Functionality

The BCM53125 8051 software source code includes the API functions shown below.

Note: The timer and task functions are not supported in software image "sfpower.bin."

Register Read/Write Functions

Most functions have a `unit` parameter, which is used for different ROBO devices. The reference code below accesses the specific register address that will apply to any ROBO device in the future.

```
/* Get Register function */
example_get_reg(uint32 unit, uint32 page, uint32 offset, uint8 *buf, uint8 len)
{
    soc_switch_t *soc;
    sys_error_t     r;

    soc = board_get_soc_by_unit(unit);
    SAL_ASSERT(soc != NULL);
    if (soc == NULL) {
        return;
    }

    r = (*soc->robo_switch_reg_get)(unit, page, offset, buf, len);
    if (r != SYS_OK){
        sal_printf("Error!\n");
        return;
    }
}

/* Set Register function */
example_set_reg(uint32 unit, uint32 page, uint32 offset, uint8 *buf, uint8 len)
{
    soc_switch_t *soc;
    sys_error_t     r;

    soc = board_get_soc_by_unit(unit);
    SAL_ASSERT(soc != NULL);
    if (soc == NULL) {
        return;
    }

    r = (*soc->robo_switch_reg_set)(unit, page, offset, buf, len);
    if (r != SYS_OK){
        sal_printf("Error!\n");
        return;
    }
}
```

Timer Functions

timer_add

Timer registration/notification.

Syntax

```
#include "system.h"
BOOL
timer_add(TIMER_FUNC func, void *arg, uint32 usec) REENTRANT
```

Parameters

func	Registration function. ***Note:*** See "Implementation of Timer Function" on page 27 for more information about the callback function prototype.
arg	Argument to pass when calling registered callback function.
usec	Microsecond for this timer.

Description

The timer is used for the registration function which needs to update periodically.

Returns

TRUE	Success
FALSE	False

Example

```
timer_add(bcm53128_linkscan_task, NULL, LINKSCAN_INTERVAL);
timer_add(cli_switch_tx_async_timer, pkt, gap);
```

timer_remove

Timer unregistration.

Syntax

```
#include "system.h"
BOOL
timer_remove(TIMER_FUNC func) REENTRANT
```

Parameters

`func`	Registration function. ***Note:*** See "Implementation of Timer Function" on page 27 for more information about the callback function prototype.

Description

Remove the registration function.

Returns

`TRUE`	Success
`FALSE`	False

Example

```
timer_remove(cli_switch_tx_async_timer);
```

Implementation of Timer Function

Syntax

```
#include "system.h"
typedef void (*TIMER_FUNC)(void *) REENTRANT;
```

Parameters

`arg`	Function argument.

Description

Add the timer task function.

Returns

None.

Task Functions

task_add

Add a function to poll periodically.

Syntax

```
#include "system.h"
void
task_add(BACKGROUND_TASK_FUNC func, void *arg) REENTRANT
```

Parameters

`func`	Registration function. ***Note:*** See "Implementation of Background Task Function" on page 29 for more information about the callback function prototype.
`arg`	Argument to pass to function.

Description

Add a function to be called periodically in the background polling loop.

Returns

None.

Example

```
task_add(bcm53125_rxtx_task, NULL);
```

task_remove

Remove a periodic function.

Syntax

```
#include "system.h"
void
task_remove(BACKGROUND_TASK_FUNC func) REENTRANT
```

Parameters

`func`	Registration function. ***Note:*** See "Implementation of Timer Function" on page 27 for more information about the callback function prototype.

Description

Remove a function from the background polling loop.

Returns

None.

Implementation of Background Task Function

Tasking: background task registration.

Syntax

```
typedef void (*BACKGROUND_TASK_FUNC)(void *) REENTRANT;
```

Parameters

`arg`	Function argument.

Description

Add the background task function.

Returns

None.

Example

```
void bcm53125_rxtx_task(void *arg) REENTRANT
```

Service Abstraction Layer Functions

The service abstraction layer (SAL) provides system services such as memory management, string management, time management, and console services. Some services can also be found in the C standard library; however, to make your application portable and reusable, use SAL functions instead of the C library.

Note: `sal_sleep` API function is not supported in software image "sfpower.bin."

Table 9: SAL Functions API

API	*Description*
sal_malloc	Memory allocation
sal_free	Memory free
sal_get_ticks	Timer: Get current ticks
sal_get_us_per_tick	Timer: Microseconds per tick
sal_usleep	Timer: Sleep for microseconds (but resolution is still in ticks)
sal_sleep	Timer: Sleep for number of ticks (background tasks would be executed)
sal_printf	Console: `printf`
sal_char_avail	Console: Whether character is available for input
sal_getchar	Console: Get input character (may block if no character is available)
sal_get_last_char	Console: Return the last input character
sal_putchar	Console: Output one character

sal_malloc

Allocates general-purpose system memory.

Syntax

```
#include <sal.h>
void *sal_malloc(uint32 size) REENTRANT;
```

Parameters

`size` Size of memory block to allocate, in bytes.

Description

Allocates size bytes and returns a pointer to the allocated memory.

Returns

<pointer>	Success: pointer to allocated memory.
NULL	Error.

Example

```
pkt = (sys_pkt_t *)sal_malloc(sizeof(sys_pkt_t))
```

sal_free

Free allocated memory using `sal_malloc` (above).

Syntax

```
#include <sal.h>
void sal_free(void *p) REENTRANT;
```

Parameters

p	Pointer to memory to be freed.

Description

Free memory previously allocated with `sal_alloc` and pointed to by `p`.

Returns

None.

Example

```
sal_free(pkt)
```

sal_get_ticks

Timer: get current ticks.

Syntax

```
#include <sal.h>
tick_t sal_get_ticks(void) REENTRANT;
```

Parameters

None.

Description

Get the system tick time.

Returns

`tick_t`	Tick value.

Example

```
curr = sal_get_ticks();
```

sal_get_us_per_tick

Timer: microseconds per tick.

Syntax

```
#include <sal.h>
uint32 sal_get_us_per_tick(void) REENTRANT;
```

Parameters

None.

Description

Every tick is based on a number of microseconds.

Returns

`<value>`	Number of microseconds.

sal_usleep

Timer: sleep for micro seconds (but resolution is still in ticks).

Syntax

```
#include <sal.h>
void sal_usleep(uint32 usec) REENTRANT;
```

Parameters

`usec`	Number of microseconds to sleep.

Description

Suspend the calling thread for the specified number of `usec` microseconds. `sal_usleep` does not execute background tasks.

Returns

None.

sal_sleep

Timer: sleep for number of ticks (background tasks would be executed).

Syntax

```
#include <sal.h>
void sal_sleep(tick_t ticks) REENTRANT;
```

Parameters

`ticks`	Sleep for number of ticks.

Description

Suspend the calling thread for the specified number of ticks. `sal_sleep` executes background tasks.

Returns

None.

sal_printf

Format and print data.

Syntax

```
#include <sal.h>
void sal_printf(const char *fmt, ...)
```

Parameters

`fmt`	Standard `printf` style format string specifying how subsequent arguments are formatted for output.
`arg`	Variable number of arguments.

Description

Print data according to format. Output is written to the standard output string.

Returns

None.

sal_char_avail

Console: Whether the character is available for input.

Syntax

```
#include <sal.h>
BOOL sal_char_avail(void) REENTRANT;
```

Parameters

None.

Description

Console: Whether the character is available for input.

Returns

Available or not.

sal_getchar

Console: Get the input character (may block if no char available).

Syntax

```
#include <sal.h>
char sal_getchar(void) REENTRANT;
```

Parameters

None.

Description

Console: Get the input character (may block if no character is available).

Returns

The number of characters.

sal_get_last_char

Console: Returns the last input character.

Syntax

```
#include <sal.h>
char sal_get_last_char(void) REENTRANT;
```

Parameters

None.

Description

Console: Returns the last input character.

Returns

The number of characters.

sal_putchar

Console: Output one character.

Syntax

```
#include <sal.h>
char sal_putchar(char c) REENTRANT
```

Parameters

c	String or character.

Description

Console: Output one character.

Returns

Character.

Customization

Table 10: Customization Files

File Path	*Description*
`um/systems/bcm953125/src/board.h`	Board and project-specific configuration
`um/systems/bcm953125/src/config.h`	Feature configuration (enable/disable)
`um/systems/bcm953125/src/board.c` `um/systems/bcm953125/src/board_init.c`	• `board_early_init` API: board devices initialization at first stage • `board_init` API: board devices initialization • Other functions in `board.c`: board-specific customization, including logical/physical port mapping
`um/bcm53125/src/appl/appl_init.c`	In case new applications are added
`um/systems/bcm953125/src/custom.c`	The following functions are callback functions, available in software image sfpower.bin for customers to implement their own software requirement: • `custom_init`: initialization for customer features • `custom_poll`: polls to do something for customer features • `custom_handle_link_up_early`: handles link-up events and is called each time a port's link goes up and before any other operation • `custom_handle_link_up`: handles link-up events and is called each time a port's link goes up • `custom_handle_link_down`: handles link-down events and is called each time a port's link goes down **Note:** Software image sfapp.bin does not support the custom_init, custom_poll, custom_handle_link_up_early, custom_handle_link_up and custom_handle_link_down functions.

Connecting everything®

BROADCOM CORPORATION
5300 California Avenue
Irvine, CA 92617

Phone: 949-926-5000
Fax: 949-926-5203
E-mail: info@broadcom.com
Web: www.broadcom.com

53125-PG104-R July 26, 2011