Freescale Semiconductor
Application Note

AN2757
Rev. 3, 06/2006

MPC5200 Quick Start

and MPC5200 Graphical Configuration Tool

by: Michal Hanak
Roznov Czech System Center
TSPG, Freescale Semiconductor

This document describes the MPC5200_Quick_Start environment for creating non-operating system applications for the Freescale MPC5200 device. The environment also includes an easy-to-use Graphical Configuration Tool (GCT) which simplifies definition of the startup configuration for MPC5200 on-chip peripherals. The MPC5200 microcontroller is based on an e300 C0 core using the PowerPC™ instruction set.

A current version of the MPC5200_Quick_Start tool is primarily designed for, and integrated with, CodeWarrior development tools. There are more tools likely to be supported in the future; please see the release_notes.txt file n the MPC5200_Quick_Start installation for an up-to-date list of tools supported by the latest release. In this document, it is assumed the user is already familiar with the target development environment.

All MPC5200 embedded-side code was tested with the CodeWarrior MGT Edition Version 8.1 and the Lite5200 (IceCube) evaluation board.

New in the MPC5200_Quick_Start Release 0.9

The new MPC5200B device is now supported in MPC5200_Quick_Start. The code was also tested with CodeWarrior MGT Edition Version 8.1 on a Lite5200B evaluation board.

Table of Contents

freescale™ semiconductor

1 Introduction

This document describes the MPC5200_Quick_Start tool version 0.9 at the time of its release. Version 0.9 brings support for a new MPC5200B processor while maintaining a fully backward-compatible support for the original MPC5200.

In this documentation, MPC5200/B refers to both MPC5200 and MPC5200B devices.

1.1 Features

The MPC5200_Quick_Start is composed of the following components:

- Framework for creating MPC5200 and MPC5200B non-operating system applications
  - CodeWarrior project stationery (project templates)
  - Startup code enabling boot-from-flash standalone operation of Lite5200 and Lite5200B
  - Linker command files for different targets (debugging, standalone, etc.)
  - Interrupt dispatcher with the support of GCT
- Graphical Configuration Tool (GCT)
  - An easy-to-use Windows-based application
  - All MPC5200/B modules supported (except USB)
  - A graphical representation of all control bits and bit fields of peripheral modules
  - Generates constants to be directly written to the processor control registers
- MPC5200/B peripheral modules initialization code
  - Applies GCT-created configuration to the MPC5200 peripheral registers
  - Optionally initializes the MPC5200 device before the `main()` procedure is entered
- Compatible with original Lite5200 board support package
  - BSP sources included in the MPC5200_Quick_Start
  - BSP register space header files reused in MPC5200_Quick_Start

1.2 Suggested Reading

Before starting with MPC5200 programming, the user should become familiar with 32-bit PowerPC architecture and G2_LE PowerPC core implementation. The following books are freely available from www.freescale.com and the Freescale Literature Distribution Center in the PDF form.

- *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture* (MPCFPE32B) — Describes resources defined by the PowerPC architecture.
- *G2 Core Reference Manual* (G2CORERM) — Describes the G2_LE core used in MPC5200 and MPC5200B
- *MPC5200 Users Guide* (MPC5200UM)
- *MPC5200B Users Guide* (MPC5200BUM)

There are also several application notes available from www.freescale.com and the Freescale Literature Distribution Center in the PDF form that are related to the MPC5200 device:

- AN2551: MPC5200 Startup Code.
- AN2604: Introduction to BestComm.
- AN2458: MPC5200 LocalPlus Bus Interface.

2 Installation

The MPC5200_Quick_Start tool setup pack is distributed as a single self-extracting executable file. Before installing this tool, Microsoft Internet Explorer 5.5 or higher must be installed on the host computer. It may also be an advantage to install the CodeWarrior MGT edition before installing MPC5200_Quick_Start. If this sequence is followed, Quick Start project stationery will be installed and integrated directly into the CodeWarrior environment.

Important: After the MPC5200_Quick_Start is installed, *before* any project or sample application is opened in CodeWarrior, the path to the MPC5200_Quick_Start source code must be registered in the CodeWarrior development environment. Unfortunately, this step cannot be automated in the installation process and must be done manually by the user. Actions required are specified in detail in the doc\todo_CW.txt file n the MPC5200_Quick_Start installation and are also briefly described in the following section.

2.1 Configuring CodeWarrior IDE

The following procedure registers the MPC5200_Quick_Start source code path in the CodeWarrior Integrated Development Environment (IDE). This path is used by all projects created using the MPC5200_Quick_Start Stationery as well as by all sample applications.

1. Launch the CodeWarrior IDE and select menu Edit / Preferences. The IDE Preferences dialog window should appear.
2. Select the Source Trees panel on the left-hand side IDE Preferences Panels list, as displayed in Figure 1.
3. In the Name box, type (exactly) the string MPC5200_Quick_Start Source (there is a space before the word Source).
4. In the Type drop-down list, select the Absolute Path type.
5. Click on the Choose button, and locate the `src` folder in the MPC5200_Quick_Start installation directory. This is typically the `C:\Program Files\Freescale\MPC5200_Quick_Start rX.Y\src`
6. Click the Add button; then, the path specified above should be added to the list.
7. Click OK to finish

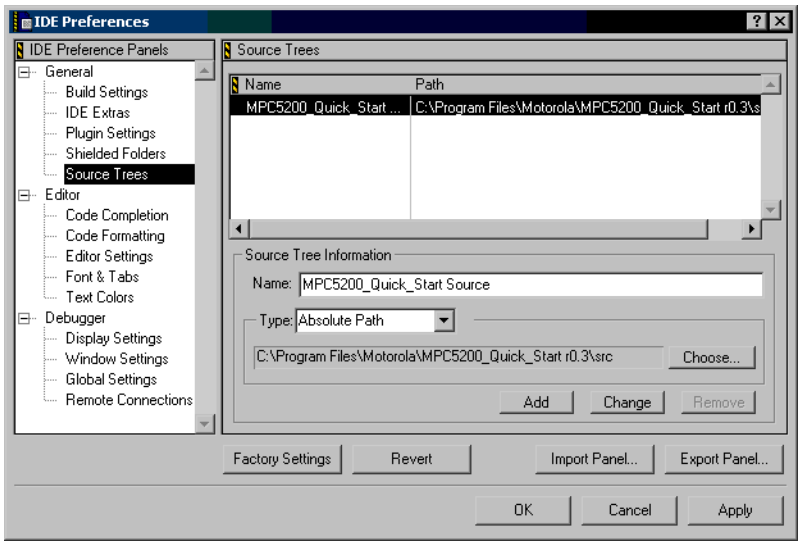

Figure 1. CodeWarrior IDE Preferences Window

3 Your First Hello World Application

After the MPC5200_Quick_Start is installed, run the CodeWarrior IDE and select menu File / New. The MPC5200_Quick_Start stationery should appear in the list of available project templates.

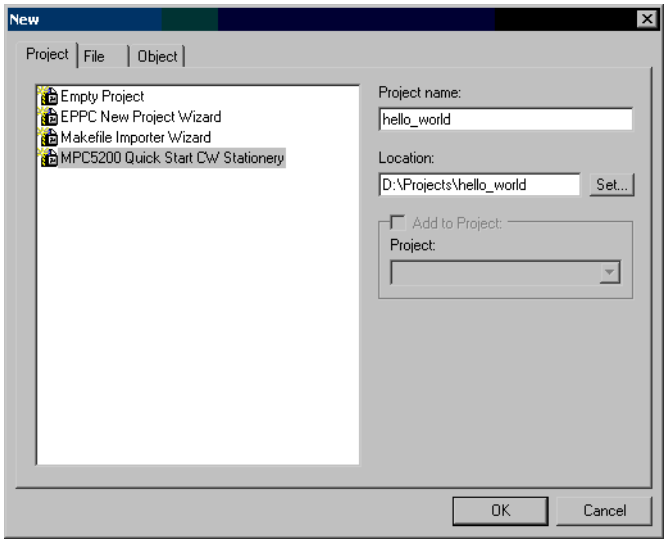

Figure 2. CodeWarrior Project Stationery

Select the MPC5200_Quick_Start stationery and create the MPC5200 or MPC5200B project using one of the three available project templates (Figure 3). When a new project is loaded into the CodeWarrior workspace, double click the `main.c` file item in the project tree to open the file in the editor window. A typical Hello World application code is prepared by default (Figure 4).

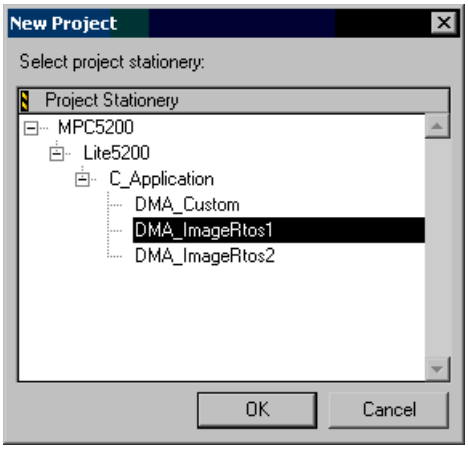

Figure 3. Quick Start Project Templates

By default, all the embedded-side code of MPC5200_Quick_Start supports the WireTAP CCS BDM common on-chip processor (COP) interface, which is also the default interface included with the Lite5200 or Lite5200B evaluation board. A different BDM interface can be selected in the project settings window after pressing the Alt+F7 key (Figure 5).

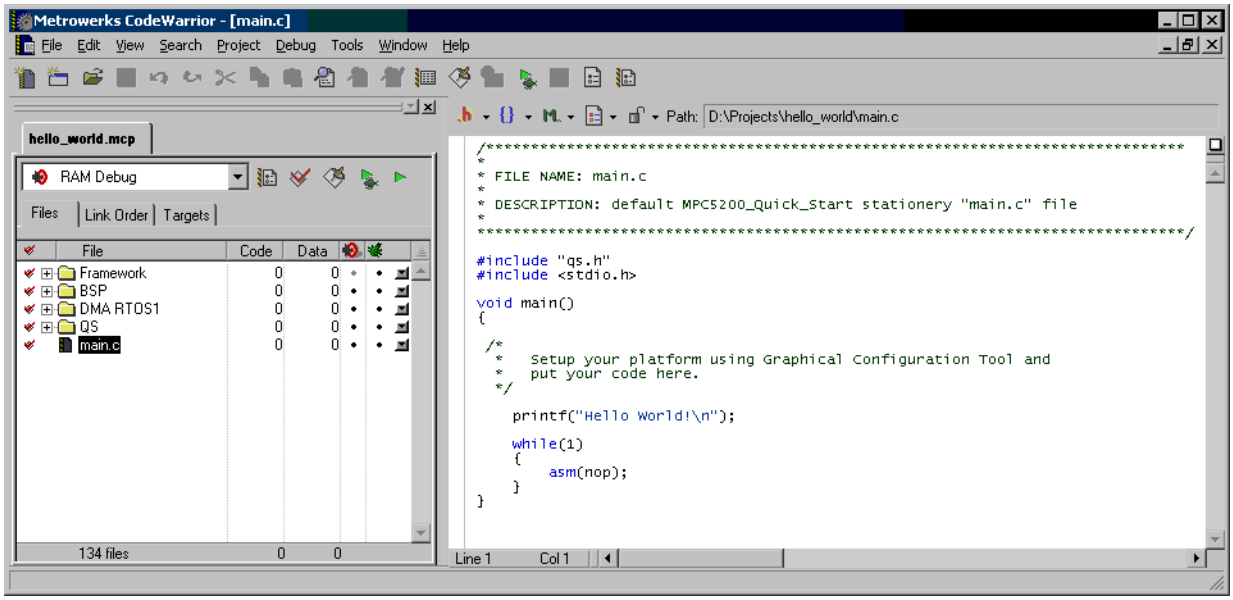

Figure 4. Hello World **Application in CodeWarrior**

The Hello World application sends its output to the console which is initially configured as a PSC1 UART serial line of speed 115200 bps, no-parity, one-stop bit. How to use the MPC5200 GCT to re-configure the PSCI console parameters will be described later in this document.

Use the null-modem cable to interconnect the PSC1 UART port of the evaluation board with the COM port on the host PC. Then run the console terminal application (such as Hyperterminal for Microsoft Windows), and configure the COM port for 115200-N-8-1 and open it.

The jumper switches on the evaluation board should be set in their default factory positions; otherwise, there is risk that the MPC5200 peripheral clock will run on a frequency different from the one configured for the Hello World application. In this case, the serial baud rate of the PSC1 interface will not match the COM port settings on the PC side, and no output will be displayed on a console window.

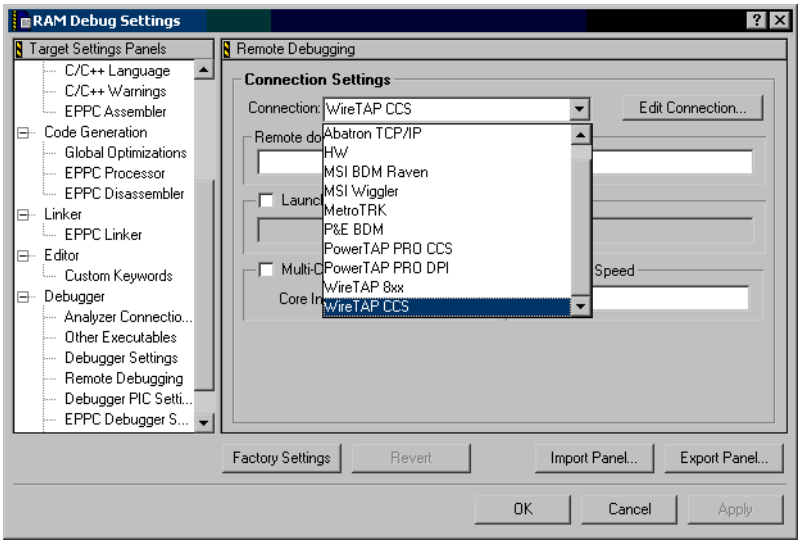

Figure 5. CodeWarrior Remote Debugging Options (Selecting BDM Interface)

The Hello World application can be built by pressing the F7 key in the CodeWarrior IDE. The build process should finish with no errors and warnings. If everything goes well, pressing the F5 key should run the application under the CodeWarrior debugger. The evaluation board is first configured over the BDM, the SDRAM (DDR on Lite5200B) memory is automatically enabled, and the compiled application executable is downloaded into the operation memory. The application is automatically started and an execution is halted at the default breakpoint on the first line of the `main()` function.

When the F5 key is pressed again, the execution resumes and the Hello World output is sent over the serial line and displayed on the console window.

The following sections describe the MPC5200_Quick_Start framework, GCT, project templates, and other details briefly mentioned during running the Hello World application.

4 MPC5200 Quick Start Projects

This section describes different kinds of CodeWarrior projects that can be created using project templates available in the MPC5200_Quick_Start tool.

4.1 Project Stationery and Templates

A project template can be viewed as a completely prepared and configured project where copy is saved under a custom name and used as a starting point for the user's own development. Such copying is done automatically by CodeWarrior when the user selects the project template and specifies a new project name

(see Figure 2 and Figure 3 on page 5). A grouped set of different project templates is called Project Stationery. Currently, there are four project templates each for the MPC5200 and MPC5200B within the MPC5200_Quick_Start Stationery. The project templates differ in BestComm DMA microcode image and DMA tasks availability:

- **DMA_Custom** - An empty DMA microcode image by default. The user is responsible for creating his own set of tasks, building the DMA microcode, and adding task C-API source files to the project. The GCT and BestComm Configuration Tool can be used to help implement the DMA functionality. Please be aware that BestComm development tools are not included in the MPC5200_Quick_Start installation.
- **DMA_ImageRtos1** - Precompiled RTOS1 DMA image included in the project. C-API files for all RTOS1 tasks are already included in the project, and the DMA image cannot be further configured by the BestComm Configuration Tool. All MPC5200_Quick_Start sample applications are based on this template.
- **DMA_ImageRtos2** and **DMA_ImageRtos3** - Same as the one above, except that the RTOS2 and RTOS3 images are used. This image contains slightly modified set of DMA tasks. See BestCommAPIUserGuide.pdf in the MPC5200_Quick_Start installation for more details about RTOS images.

4.2 Project Targets

Except for the BestComm and DMA functionality, all three project templates in the MPC5200_Quick_Start Stationery are identical. This section describes the project targets available in each project and how to use the targets to debug or to prepare a standalone application.

A project target is a named configuration of a project, including the set of files to compile, actual settings of the compiler/linker, and settings of the debugger environment. The following targets are available in each MPC5200_Quick_Start project:

- **RAM Debug** - This target is primarily used for debugging of embedded application over a BDM link. The CodeWarrior debugger uses the BDM interface to prepare the evaluation board (Clocks, SDRAM memory, etc.) *before* it downloads the application which is executable directly into the RAM for debugging.
- **ROM Image** - This target can be used for debugging without BDM interface or to deploy applications to firmware-based systems. The application is compiled into a compact self-extracting executable image (relocatable), which can be loaded and started by the evaluation board firmware (dBug on Lite5200 or U-BOOT on Lite5200B). The firmware is typically capable of loading the image over an ethernet network using a TFTP protocol and capable of saving the image into the non-volatile memory. In the case of debugging, the image is typically downloaded and run manually using the firmware console commands. When making an application standalone, the firmware can be configured to run the image automatically after the system boots up.

  When the image is run (by jumping to its base address), it relocates itself into operational RAM and begins execution of the `main()` program. Memory relocation typically means that the firmware's variables and exception vectors are lost, and firmware operation can not resume even if the application finishes.

NOTE

Since the MPC5200 system is not in the post-reset state when running an application of the ROM Image target, it is highly recommended to enable the Generate All Register Values setting in the GCT options. Otherwise, the GCT saves modified (non-reset value) register values only into the `appconfig.h` file. As the firmware configures some modules for its own use, there is a risk that the modules are only partially re-configured by the Quick Start initialization code. See Section 6.2.2, "GCT Options" for more details.

- **Standalone BL** - After an application is debugged using one of the targets described above, the Standalone BL target can be used to compile a standalone executable image. When this image is programmed to the Boot Flash memory starting at Boot Flash address 0, the application is ready for boot-low standalone operation that does not require any firmware. The startup code takes care of relocating the Flash memory to the end of the address space, initializing SDRAM controller and SDRAM/DDRAM memory from address 0x00000000, relocating the code, and invoking the `main()` function from RAM.

As both the Lite5200 and Lite5200B come with their firmware as a boot-high option by default (address 0xFFF00000), the application image built with Standalone BL target can co-exist with the firmware. The user selects the boot-low or boot-high option using a jumper switch on the evaluation board.

Table 1. Comparing MPC5200_Quick_Start Targets

	RAM Debug	ROM Image	Standalone BL
SDRAM Memory	Set up by debugger	Set up by firmware	Set up by application itself
Flash Memory	Not used	Not required (can be used by firmware to store the image)	Used for boot-low at address 0x00000000. Later relocated to 0xFF000000
Code Execution	From SDRAM only	Starts by jumping to the Image base address, continues in SDRAM	Starts in Boot Flash, continues in SDRAM
Executable Name	ramdebug.elf / ramdebug.mot	romimage.elf / romimage.mot	runram_bl.elf / runram_bl.mot
Entry Point	__start	offset 0 of the image	Boot-low: 0x00000100 (__reset)
Prefix File Macro	TARGET_RAMDEBUG	TARGET_ROMIMAGE	TARGET_RUNRAM

4.3 Making the Application Standalone

Once the application is debugged using the RAM Debug or ROM Image targets, it can be rebuilt using the Standalone BL target and programmed into the non-volatile Flash memory for a standalone operation. The following sections will briefly describe how to use CodeWarrior's Flash Programmer to achieve standalone operation.

4.3.1 MPC5200/B Boot Process

After the system reset signal is de-asserted, the MPC5200 boot process begins at one of two addresses, 0x00000100 or 0xFFF00100, in the Boot CS space. The address selection depends on the state of the B H/L board configuration jumper (named HI/LO on the Lite5200B Evaluation Board):

- **Boot-Low** - Execution starts at address 0x00000100 with an exception prefix set to 0x00000000 (MSR.IP bit cleared).
- **Boot-High** - Execution starts at address 0xFFF00100 with an exception prefix set to 0xFFF00000 (MSR.IP bit set).

After the Lite5200/B board reset is released, the Boot CS space is automatically mapped to the small area of non-volatile Flash memory at one of the two addresses above. It is a responsibility of the boot code to enable the rest of the Flash space and to remap the Flash space using the CS0 signal. The CS0 signal shares the physical pin with the Boot CS while using a different address-mapping register. Using the CS0 signal, even the Flash-running code is capable of remapping its own Flash space to the area not overlapping with future RAM space. As the last step, the SDRAM controller and the RAM memory should be enabled, and a code should be relocated from Flash to RAM for execution.

The startup code of the MPC5200_Quick_Start Standalone BL target performs all the steps described above to prepare the evaluation board for running the application. By default, a firmware is factory-programmed at the end of non-volatile Flash memory for the boot-high mode. The Standalone BL target uses boot-low mode so that the application may co-exist with the firmware code in Flash memory. Setting the B H/L jumper to the boot-low or boot-high option selects either an application or a firmware for execution. See also the Section 5.3, “Startup Code later in this document.

4.3.2 CodeWarrior Flash Programmer

CodeWarrior Flash Programmer is an application which can be used to program a target Flash memory over the BDM interface. In theory, the Flash Programmer performs the following tasks:

- Uses the CodeWarrior debugger initialization file to reset and set up the MPC5200 target over the BDM (see Section 5.2.3, “Debugger Initialization Files)
- Uses the BDM interface to download a Flash burning algorithm (a driver) suitable for the Flash memory selected by the user
- Uses the BDM interface to instruct a driver to perform requested operation (Flash Blank Check, Sector Erase, Memory Write). When any data are needed to be passed to the driver, it is downloaded by the BDM interface as well.

All the operations, including resetting the device, are performed by the Flash Programmer over a BDM interface. The target MPC5200 does not need to be in known state prior programming. The Flash Content may even be totally invalid while the Flash Programmer can still re-program it.

To make the Standalone BL application really standalone, select menu Tools / Flash Programmer in the CodeWarrior IDE. In the Flash Programmer window (Figure 6), first press the Load Settings... button and load the `flash_prog.xml` file located in the `sample_applications` folder for selected evaluation board.

On the Target Configuration page, the user specifies the debugger configuration file to be used to initialize the target system. As the Flash Programmer was invoked with the CodeWarrior Project loaded and with Standalone BL target active, the default Debugger Configuration file for the target will be used (see Section 5.2.3, “Debugger Initialization Files”).

The Target Memory Buffer Address and Size define the RAM memory area used by the Flash programmer for its operation. Note that addresses loaded from the `flash_prog.xml` configuration file specify a post-reset location of the MPC5200 Static RAM memory, which is always valid, and there is no need to use SDRAM memory.

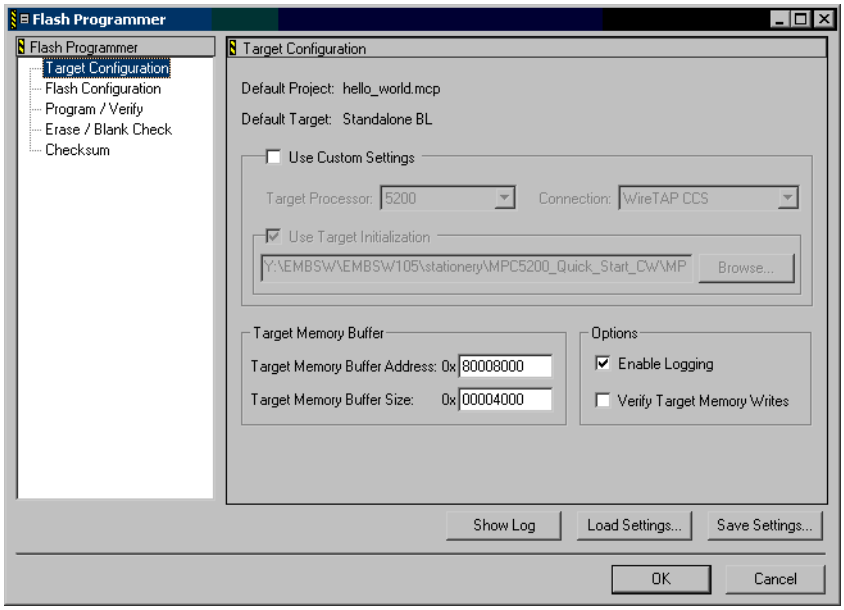

Figure 6. CodeWarrior Flash Programmer: Target Configuration

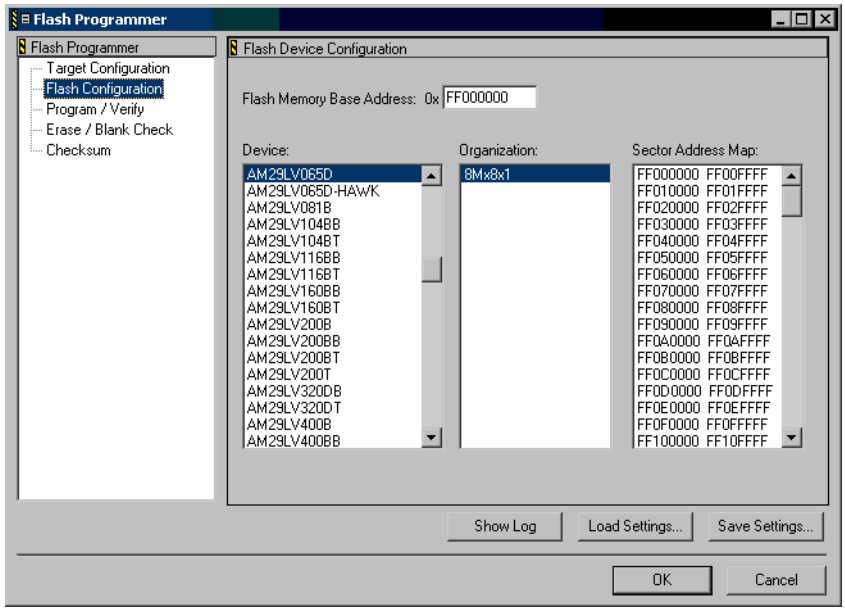

Figure 7. CodeWarrior Flash Programmer: Flash Configuration

The second Flash Configuration page of the Flash programmer (Figure 7) shows the type and address of the Flash device to be programmed. For both the Lite5200 and Lite5200B, the proper device is set already by loading the appropriate `flash_prog.xml` configuration file. A Flash Memory Base Address should be set to 0xFF000000, which is the address assigned to the Flash (CS0) memory space by the script in the Debugger Configuration File (Section 5.2.3, "Debugger Initialization Files").

On the third Program / Verify page of the Flash Programmer (Figure 8), it is possible to specify an s-record file (with the `.mot` extension) to be programmed and to perform the Flash programming itself; however, before programming the Flash memory, it should be first checked to ensure that the Flash sectors to be programmed are erased and blank. Go to the fourth Erase / Blank Check page of the Flash Programmer, select the Flash sectors to be checked, and press the Blank Check or Erase button. The amount of memory (a number of sectors) to be checked or erased depends on the size of the s-record file to be programmed. The s-record file can be examined with a text editor. The last S3-record gives an information about how long the image is (see Figure 9). In the case of the Hello World application created in Section 4, "MPC5200 Quick Start Projects," the S-record is about 0x9400 bytes long.

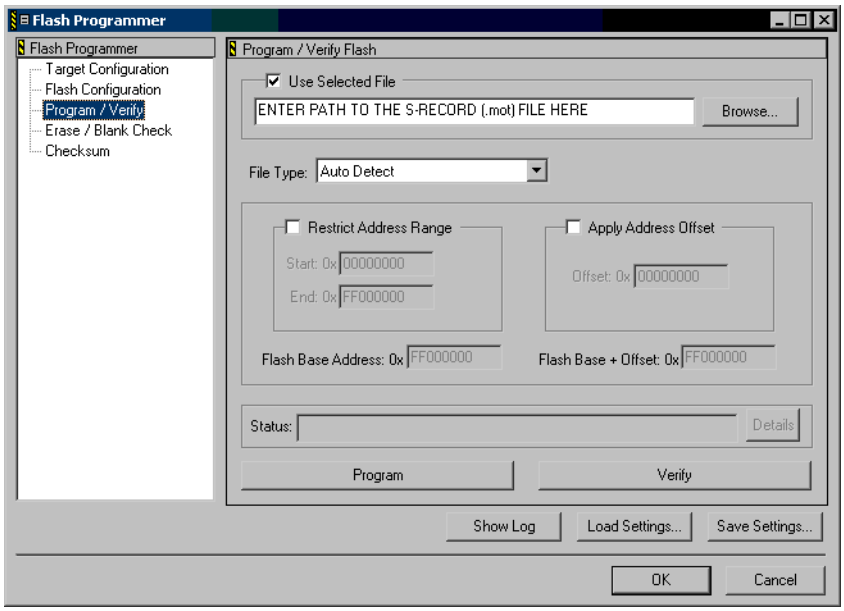

Figure 8. CodeWarrior Flash Programmer: Program / Verify

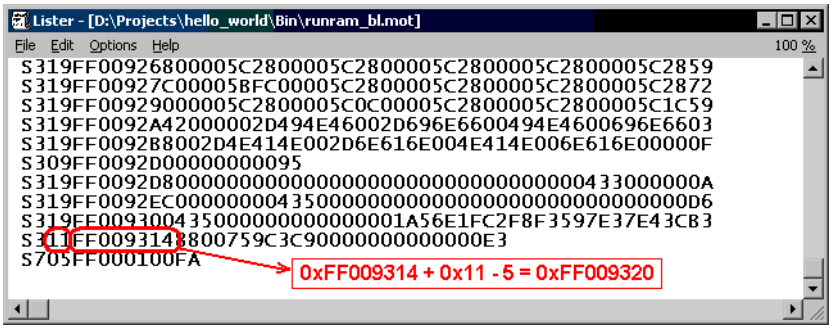

Figure 9. Examining the Image SIze in the S-Record File

Go back to the Program / Verify page in the Flash Programmer and specify a path to the s-record file to be programmed (`runram_bl.mot` for the Standalone BL target) and press the Program button. If the BDM interface is properly connected, and the evaluation board is powered on, the Flash Programmer Status line should display the progress of the operation. The Show Log button can be used to display a detailed report of a Flash programming operation.

When the B H/L jumper switch is set to the boot-low option on the Lite5200 board, the Standalone BL application is ready for booting from the Flash memory.

5 Application Framework

This section describes the content and key parts of each MPC5200_Quick_Start-based project.

In the CodeWarrior project tree window, a project can be seen, logically divided into a tree-like structure of virtual folders and project files (Figure 10). The following subsections will describe each project item in detail.

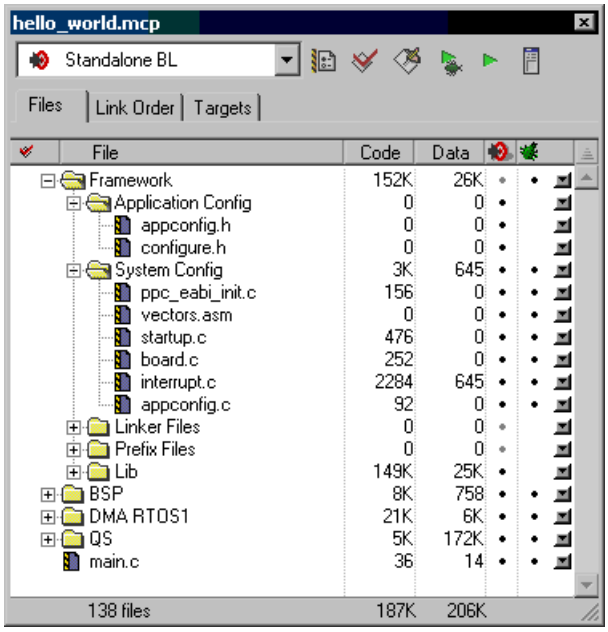

Figure 10. Project Content

5.1 Application Configuration Files

The Application Config project folder contains two project header files, physically located in the `ApplicationConfig` sub-directory within a project folder on a hard disk.

The `configure.h` file is included by the MPC5200 BSP source files (see Section 9, “MPC5200 BSP”). For the BSP-only applications, this file can define parameters of the PSC1 console and several macros for the MPC5200 clock frequency calculations. This file is superseded by the `appconfig.h` file in the Quick Start applications and is not used at all.

The `appconfig.h` file is the main application configuration file for the Quick Start-based applications. It contains the initialization values of all important control registers for each MPC5200/B peripheral module the user wants to configure. The GCT can be used to edit the content of this file graphically. See Section 6, "Graphical Configuration Tool" for more details.

5.2 System Configuration Files

The System Config project folder contains startup files, linker command files, prefix files and some CodeWarrior configuration files required by the Quick Start project. All these files are physically located in the `SystemConfig` sub-directory within a project folder on a hard disk. The files are described in the following sections.

5.2.1 Prefix Files

Prefix file is an standard C header file included by default into every C file being compiled. A prefix file can be specified among other C compiler settings globally for all the C files in the project. When set, a prefix file behaves exactly like it is included using an `#include` directive at the beginning of each C file of the project.

In the Quick Start, there is a different prefix file for each project target. All prefix files are located in the `SystemConfig` subdirectory of the project folder. Each prefix file defines its macro (such as TARGET_RAMDEBUG in RAM Debug target), using which a source code identifies the target it is being compiled for. See Table 1 for prefix file-defined macros of each target.

5.2.2 Linker Command Files

Linker Command Files are processed by the linker when it is placing code and data segments to specific memory locations. The linker command file defines memory areas by the means of base address and size, and assigns code and data segments declared by the C compiler into those areas.

In MPC5200_Quick_Start, there is a separate Linker Command File for each project target. All Linker Command Files are located in the `SystemConfig` sub-directory of the project folder on the disk. In each target, the file defines the location of the exception vectors, placement of code and variables into RAM, and defines a software stack of a reasonable size (256kB). The Linker Command File syntax used in CodeWarrior is described in the Targeting Embedded PPC User Manual located in the CodeWarrior `Help\PDF` folder.

Unlike the linkers from other vendors, CodeWarrior's Linker Command File syntax does not support the AT or LOAD directives which are typically used when building a compact image suitable for writing into non-volatile memory. Instead of this feature, there is a Generate ROM Image check box in the CodeWarrior linker settings. With this option enabled, when the linking is finished, the linker walks through all initialized memory sections and puts them one after one starting at the specified ROM-image address. Simultaneously, it generates a special array of data structures in which the original and ROM-image addresses are specified for each such section.

In the MPC5200_Quick_Start, there are two targets that make use of the Generate ROM Image feature of the linker, as follows:

- The ROM Image target uses this feature to build an self-extracting image, which can be run by the firmware code. In a startup code of the image, the image uses the information from the CodeWarrior Linker-generated descriptor array and copies all memory sections from the ROM Image to the destination addresses in RAM.
- The Standalone BL target uses this feature similarly like the ROM Image, except that the SDRAM controller and SDRAM memory are initialized before the ROM Image is extracted. See Section 4.3.1, "MPC5200/B Boot Process" for more details.

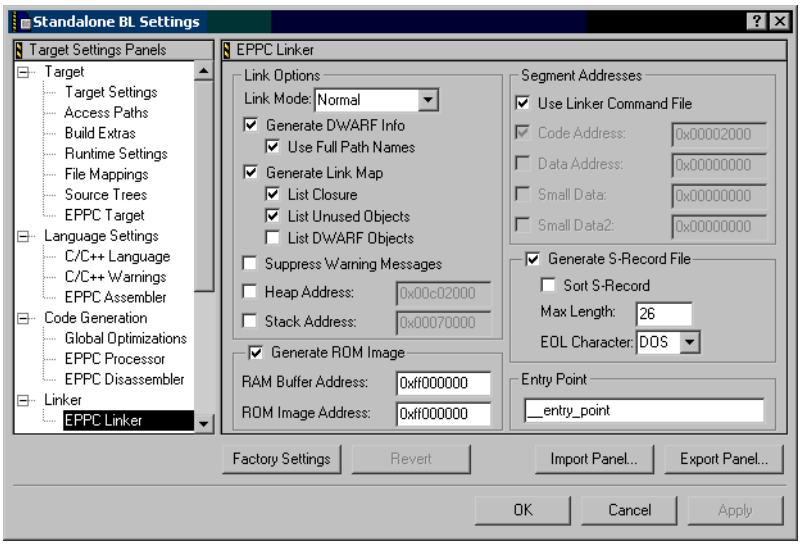

Figure 11. Project Linker Settings

5.2.3 Debugger Initialization Files

Debugger initialization and configuration files are used by the CodeWarrior whenever it is about to connect to the board over a BDM interface. There are two kinds of Debugger files, both selected for each project target in the project settings window (Figure 12), as follows:

- **Target Initialization File**: Uses very simple language to initialize the PowerPC core registers and MPC5200 memory-mapped control registers. Typically, its job is to configure Flash access space, SDRAM controller, and SDRAM memory before the Debugger downloads the application for debugging or before a Flash Programmer begins its operation.
- **Memory Configuration File**: Describes the memory areas of the target system. Using a simple language, the memory regions can be declared as Read/Write, Read Only or No-Access. A CodeWarrior Debugger uses the information from this file when it is about to display the content of memory or memory-mapped registers. This file is applied also when processing a Target Initialization File, so the memory locations accessed by script in the Target Initialization File must also be defined as valid (Read/Write) in the Memory Configuration File being used.

There are two Target Initialization files available in the MPC5200_Quick_Start, as follows. The first one is used by all projects and project targets.

- **`init_ram.cfg`** - Initializes Flash memory space to be in range 0xFF000000...0xFFFFFFFF; initializes SDRAM controller for 64MB RAM memory starting at address 0x00000000 (256MB DDRAM on Lite5200B) and enables the PowerPC Core time-base counter (TB special purpose register).
- **`init_flashonly.cfg`** - Initializes only the Flash and TB counter only. Can be used with Flash Programmer as it does not require SDRAM memory to be valid. (Flash Programmer uses MPC5200 built-in static RAM for its operation.)

There are two Memory Configuration Files in the MPC5200_Quick_Start, as follows. The first one is used by all projects and project targets:

- **`mmap_ram.mem`** - Defines the SDRAM memory area starting at address 0x00000000, Flash memory area starting at address 0xFF000000 and two memory areas for MPC5200 memory-mapped registers. One such area is based at address 0x80000000 which is the reset value of the MBAR peripheral-base address register. This address is used by the script in the Target Initialization File. The second memory-mapped registers area is opened for MBAR at 0xF0000000, which is the default operational location of MBAR in Quick Start applications. The MBAR base is defined in the `appconfig.h` configuration file and can be set by the GCT. For proper operation of the CodeWarrior debugger, the `mmap_ram.mem` file should be updated any time the MBAR is assigned different value in GCT. See Section 6.4.2, “PowerPC Core (CORE)” for more details.
- **`mmap_fbl.mem`** - Is similar to the mmap_ram.mem above, with the exception that Flash memory is defined to start at address 0x00000000 and SDRAM memory is not used at all. This Memory Configuration file can be used when debugging the Flash-based boot-low applications directly from the Flash memory.

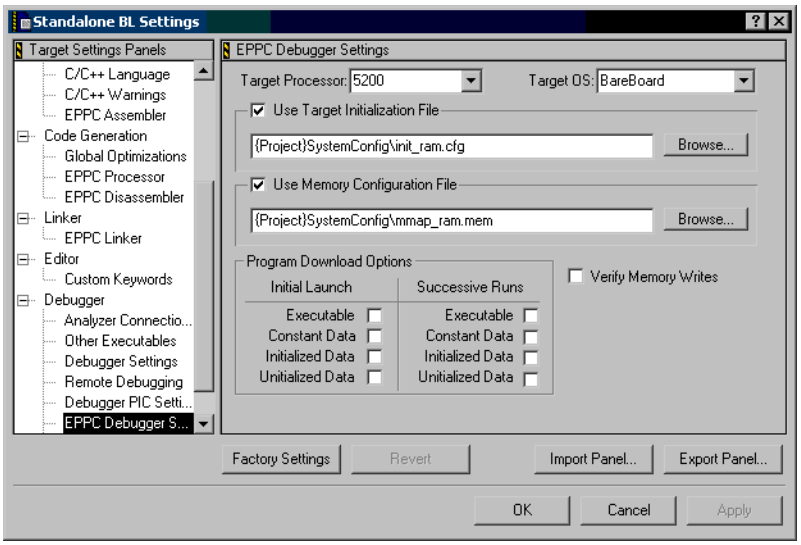

Figure 12. Debugger Settings

5.3 Startup Code

Startup code is a key part of each MPC5200_Quick_Start project. It is implemented in the source files located in the `SystemConfig` sub-directory of the project folder on a hard disk.

- **`startup.c`** - Contains the `__start` global entry point. The `__start` function is relocatable, which means it can be run from any location and its purpose is to call several other subroutines to initialize the PowerPC Core registers, initialize board and memory, initialize EABI registers and stack pointer, relocate the code from flash ROM Image into a proper RAM destination, jump to the RAM location, and continue by invoking the `main()` function.
- **`board.c`** - Contains the `__reset` exception handler, which invokes the `__start` startup code. It also contains some hardware-specific operations like enabling the Flash memory in CS0 space at address 0xFF000000 and preparing the SDRAM or DDRAM memory.
- **`ppc_eabi_init.c`** - Contains functions to initialize or free the C++ environment (if used) and to call `__pre_main()` and `__post_main()` user functions.
- **`appconfig.c`** - This file contains a default implementation of the `__pre_main()` and `__post_main()` functions. In the MPC5200_Quick_Start, the `__pre_main()` function can be configured by GCT to perform automatic initialization of MPC5200 peripheral modules before the `main()` function is entered.

5.4 Interrupt Dispatcher

The Interrupt Dispatcher is a thin piece of software layer which handles all exceptions generated by the MPC5200 system, saves the EABI context, and calls the user-supplied exception service routine.

For the case of external interrupts (External Interrupt, Critical Interrupt, System Management Interrupt) the Dispatcher is also capable of decoding an interrupt source which needs to be serviced with highest priority and invoking the user-supplied service routine. Interrupt service routines can be assigned to each peripheral interrupt source either dynamically in run-time or statically using the `appconfig.h` file and the GCT. The user may also want to save Floating-Point context and to re-enable the Floating Point unit before passing an execution to the selected interrupt service routines.

In other words, the Interrupt Dispatcher effectively hides the differences between various peripheral interrupt sources of the MPC5200 and makes the interrupt handling easy and straightforward. The Interrupt Dispatcher is also natively supported by the GCT. At each peripheral module configuration page, it is possible to specify an interrupt service routine, assign an interrupt source priority, and mask or unmask the interrupt source. For more information, see the Section 6.4.6, “Interrupt Controller (SIM / ICTL).”

The Interrupt Dispatcher is implemented in the following two files, all located in the `SystemConfig` subdirectory of the project folder on a hard disk.

- **`vectors.asm`** - This file contains Reset exception handler (a branch to `__reset` code in the `board.c` file) and prologue/epilogue code for all other PowerPC exceptions. Prologue and epilogue code take care of saving volatile EABI registers and exception return address onto the software stack and of invoking a user-supplied exception handler routine.

- **`interrupt.c`** - This file implements an Interrupt Dispatcher functionality as described above in this section. When the Interrupt Dispatcher is used, the three external exceptions in the `vectors.asm` code are hard-routed to handlers supplied in the `interrupt.c` file

5.5 BSP Source Code

The MPC5200_Quick_Start is based on original Board Support Package (BSP) code for the MPC5200. To maintain compatibility with older BSP-based applications, the most of the BSP source files are included in each project created using the MPC5200_Quick_Start Stationery. All BSP files are physically located in the `src\support\bsp` folder in the MPC5200_Quick_Start installation. In the CodeWarrior project, all the BSP files used can be found in the BSP virtual project folder.

Section 9, "MPC5200 BSP" gives a brief overview of the BSP code reuse in the MPC5200_Quick_Start applications.

5.6 DMA Files

Each MPC5200_Quick_Start project includes a BestComm DMA microcode image as well as a set of the C source files implementing DMA tasks API. As described in the Section 4.2, "Project Targets," there are three project templates in the MPC5200_Quick_Start Stationery differing only by the DMA code.

The projects created using the any of the DMA_ImageRtos templates include all required DMA source files by default. The source code files can be found in the DMA RTOSx virtual project folders in the CodeWarrior project tree.

The project created using the DMA_Custom project template includes only the source files containing the (empty) DMA microcode image. In this case, the external tools (such as BestComm Configuration Tool) are used to build DMA image and task C API files. It is then the user's responsibility to add all generated source files to the CodeWarrior project.

In all cases, the DMA files are located in the `dma_image` subdirectory of the project folder on the disk.

5.7 MPC5200_Quick_Start Source Code

The last group of files included in all MPC5200_Quick_Start projects contains the source files implementing the low-level initialization code for each MPC5200 peripheral module. All files are located in the QS virtual project folder in the CodeWarrior project tree.

In addition to several system files, there is a pair of .c and .h files for each MPC5200 peripheral module. The following table briefly describes the content of the QS source files.

Table 2. Quick Start Source Files

File Name	Description
qs.h	This is a master header file for the Quick Start application. It includes all other system and MPC5200 header files from the MPC5200_Quick_Start environment.
qs_version.h	Defines Quick Start version macros

Table 2. Quick Start Source Files (continued)

File Name	Description
qs_system.h	Defines system macros used in Quick Start applications (QSTRACE, QSASSERT,...). Also defines a key **ioctl()** system call described in <Blue>Section 7, <Blue Italic>Module Initialization Code
qs_arch.h, qs_periph.h	Defines the memory map, special register bit values and other architecture-dependent macros for the target processor.
qs_arch.c	Implements an architecture-dependent code which was not made inlined in qs_arch.h
qs_ata.h, qs_ata.c	ATA Controller support (includes ATA Hard Drive detection functions)
qs_core.h, qs_core.c	PowerPC Core Initialization Code
qs_cdm.h, qs_cdm.c	Clock Distribution Module support
qs_fec.h, qs_fec.c	Fast Ethernet Controller support
qs_qpio.h, qs_gpio.c	General Purpose I/O Module support
qs_gpt.h, qs_gpt.c	General Purpose TImers support
qs_i2c.h, qs_i2c.c	I2C Controller support
qs_ictl.h, qs_ictl.c	Interrupt Controller support and Interrupt Dispatcher API definition
qs_ipbi.h, qs_ipbi.c	IPBI (Memory Map) Module support
qs_lpc.h, qs_lpc.c	LocalPlus Bus Controller support
qs_mscan.h, qs_mscan.c	MSCAN Module support (includes complete MSCAN low-level driver, see the mscan_demo sample application for the low-level driver usage)
qs_pci.h, qs_pci.c	PCI Local Bus Controller support (includes PCI database lookup functions)
qs_pcidb.h	Content of PCI database (PCI vendor and PCI device names)
qs_psc.h, qs_psc.c	Programmable Serial Controller support
qs_rtc.h, qs_rtc.c	Real Time Clock Module support
qs_sdma.h, qs_sdma.c	BestComm Module support
qs_sdram.h, qs_sdram.c	SDRAM Controller support (includes TAP-delay detection code for DDRAM memories)
qs_sdram_dflts.h	Lite5200 board default values for SDRAM Controller
qs_slt.h, qs_slt.c	Slice Timer Module support
qs_spi.h, qs_spi.c	Serial Peripheral Interface support
qs_usb.h, qs_usb.c	Placeholder for the future USB support implementation. USB is not supported by the current version of MPC5200_Quick_Start
qs_xlarb.h, qs_xlarb.c	XLB Arbiter Module support

5.8 The main.c File

Each MPC5200_Quick_Start project contains a single source file named `main.c`. This file contains typical Hello World application source code.

If a user application grows beyond the single source file (which is most likely to happen with MPC5200 applications), it is the user's responsibility to add other source files to the CodeWarrior project.

6 Graphical Configuration Tool

This section describes the Graphical Configuration Tool (GCT) which is included in the MPC5200_Quick_Start development environment. The GCT is a standard Microsoft Windows-based application used to graphically edit (read and write) the project's `appconfig.h` file. All control bits and bit-fields of each MPC5200 peripheral module are presented in an easy-to-understand graphical form. The register initialization values edited by graphical controls can be immediately displayed and/or written back to the `appconfig.h` file. Here, they are used by the module initialization code to set up the individual peripheral modules. As it was already described in Section 5.3, "Startup Code," the GCT can be also used to select MPC5200 peripheral modules which are to be automatically configured before the code execution hits the application's `main()` function.

6.1 Integration into CodeWarrior IDE

The most effective use of the GCT is to integrate it into the CodeWarrior IDE and assign a hot-key or menu-item for invoking it. In such a configuration, the GCT is automatically opened for the project currently active in the CodeWarrior IDE.

A detailed description of how to integrate the GCT into CodeWarrior IDE can be found in the todo_CW.txt document in the MPC5200_Quick_Start installation. The procedure is also briefly described below.

In CodeWarrior, select the menu Edit / Commands and Key Bindings. In the Customize IDE window (see Figure 13), select the menu group in which you want to create a new GCT menu item (such as Project).

1. Click on the New Command button
2. Type MPC5200 Configuration Tool to the Name box
3. Click on Appears in Menus check box and check it
4. Click on button on the right hand side of the Execute edit box and browse for the `gct5200.exe` GCT executable. Typically, you can find the executable file in: `C:\Program Files\Freescale\MPC5200_Quick_Start rX.Y\config_tool`.
5. Click on the button on the right hand side of the Arguments edit box and select the Project File directory item from the popup menu
6. If you want to assign a key binding, click on the New Binding button and press chosen key combination, such as Ctrl+F12
7. Press the Save button and close the dialog box

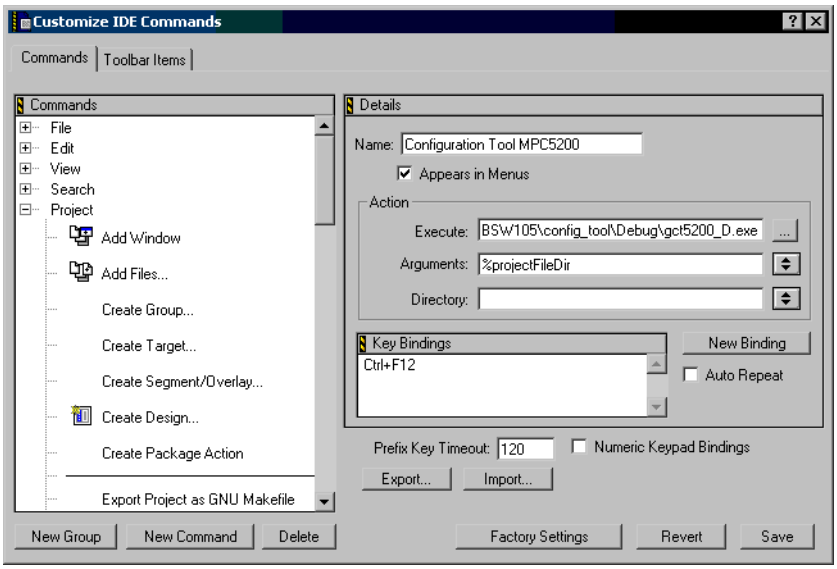

Figure 13. Key Binding for MPC5200 GCT

Now you can run the MPC5200 GCT using an assigned key shortcut or by selecting a newly created item in the CodeWarrior menu.

6.2 GCT User Interface

When starting the GCT, the path to the project directory should be passed to it as a command line argument (this is done automatically when invoking the GCT from CodeWarrior IDE). GCT first locates the `ApplicationConfig\appconfig.h` file in the current project, reads it, and displays the loaded configuration in its main application window.

The main GCT application window is split into three panes. On the left hand side, there is a tree-like view of MPC5200/B peripherals. Tree items are logically grouped into branches, each representing a set of peripherals with similar or same functionality. The following sections describe each item in the peripheral tree.

The right hand side of the window displays a configuration page for a peripheral module selected in the tree view. Above the tree view on the left hand side, there is a brief summary of key system clocks as configured in the current project. See Section 6.4.1, “Clock Distribution Module (CDM)” for more details about setting system clocks.

6.2.1 GCT Input/Output File - appconfig.h

The GCT opens and saves the MPC5200 configuration into the project’s `appconfig.h` file located in the `ApplicationConfig` subdirectory of the project folder on a disk. A configuration is saved as a set of macro values (`#define`) in the form of a standard C-header file. This header file is included by all MPC5200_Quick_Start project files, and the values from this file are used to initialize the MPC5200/B control and PowerPC core registers.

Most of the items in the tree view do have a check-box field. Using it, the user selects the modules for which the configuration is to be saved in the resulting `appconfig.h` file. When the check box is not checked, and the module configuration in the GCT differs from module post-reset state, the user is warned about that the modified configuration will be lost.

Depending on the GCT settings, the configuration saved in the `appconfig.h` file may also contain human-readable comments describing the configuration values as they were displayed in the GCT.

As the MPC5200_Quick_Start is built on top of the original MPC5200 BSP, the GCT saves the configuration in the 8-, 16-, and 32-bit BSP format of peripheral register values. Be aware that this format often does not correspond exactly to the 32-bit register definitions as stated in the MPC5200 Users Guide.

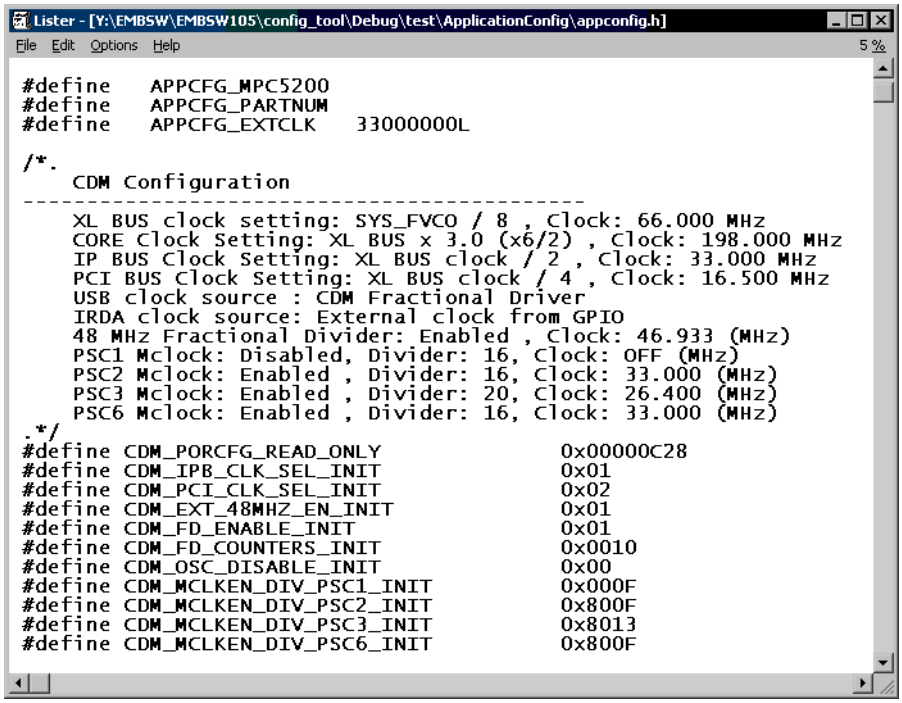

Figure 14. Example of GCT-Saved Configuration

6.2.2 GCT Options

The File / Options... menu in the GCT opens the Configuration Tool Options dialog with a few settings controlling the `appconfig.h` file output.

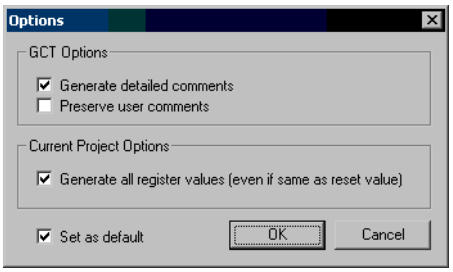

Figure 15. GCT Options

- **Generate detailed comments** - Enables saving of the human-readable commentary describing the configuration of each module. An example of the comments generated for the CDM module is on the Figure 14 above.
- **Preserve user comments -** When checked, this check-in box assures that user comments placed after the individual macro values are not lost when generating the `appconfig.h` file. This option is rarely used as there is typically no need for the user to manually edit any comments in the `appconfig.h` file.
- **Generate all register values** - Enables saving of *all* register values of selected peripheral modules into the `appconfig.h` file. When this option is not enabled, only registers with non-reset values, such as those modified in the GCT, are saved. Omitting the reset-value registers in the `appconfig.h` file can reduce the size of the module initialization code. (Those values not defined in the file are not written to the peripheral registers.) This approach requires that all modules being initialized be in post-reset state; otherwise, the result may be unpredictable.

NOTE

The **Generate all register values** option should always be set when compiling the application for ROM Image target. Such applications are typically invoked by the firmware code which may modify a configuration of peripheral modules so they will not necessarily be in post-reset state.

6.3 MPC5200/B Pinout Page

When the GCT is started, the MPC5200 Pinout page is displayed, showing the schematic part-like view of the processor.

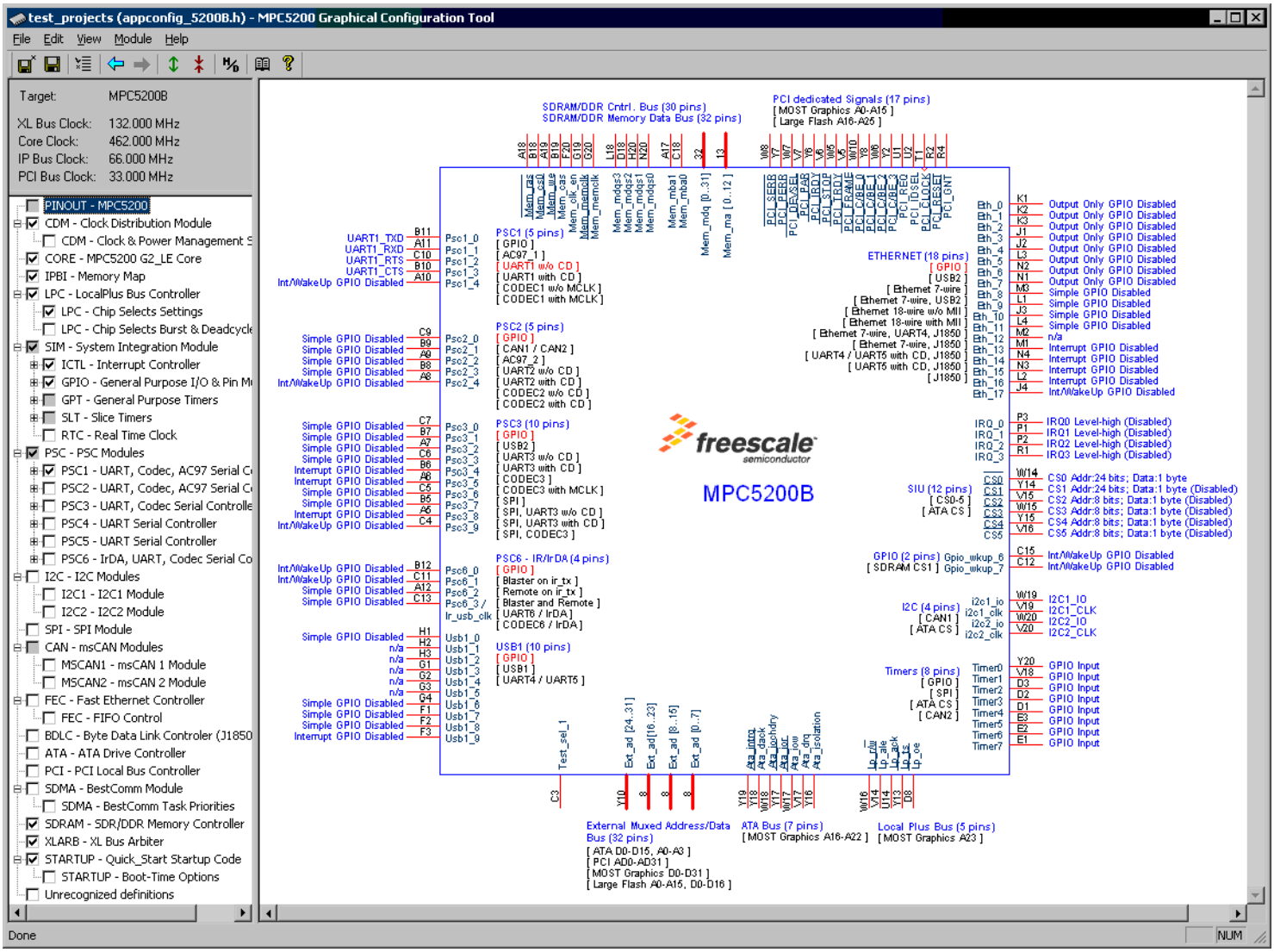

Figure 16. MPC5200B Pinout Page

Package pins are labeled with both the official pin name, including the BGA ball identifier and the pin function, assigned by the MPC5200/B GPIO port multiplexer. See Section 6.4.7, "General Purpose I/O (SIM / GPIO)." Most labels on the page are active hypertext links which, when clicked, open the appropriate control page in the GCT.

There are two kinds of hypertext labels, as follows:

- **Pin Labels** (blue) - Show the pin function currently assigned by the pin multiplexer. The hypertext links of these labels activate the GCT page where the pin function can be re-defined.

- **Mode Labels** (black list, blue head item) - Each group of these labels display the peripheral modules and their operational modes supported by one pin/port multiplexer. The multiplexer mode currently selected in the GPIO configuration is highlighted in red. A hypertext link of the mode labels activates the control page of applicable peripheral modules.

6.4 MPC5200/B Peripheral Modules

The use of the GCT is very intuitive and does not require a detailed description. The following sections will display screen shots of each MPC5200 (or MPC5200B) configuration page and will give a brief overview of settings or specific behavior of the graphical controls.

6.4.1 Clock Distribution Module (CDM)

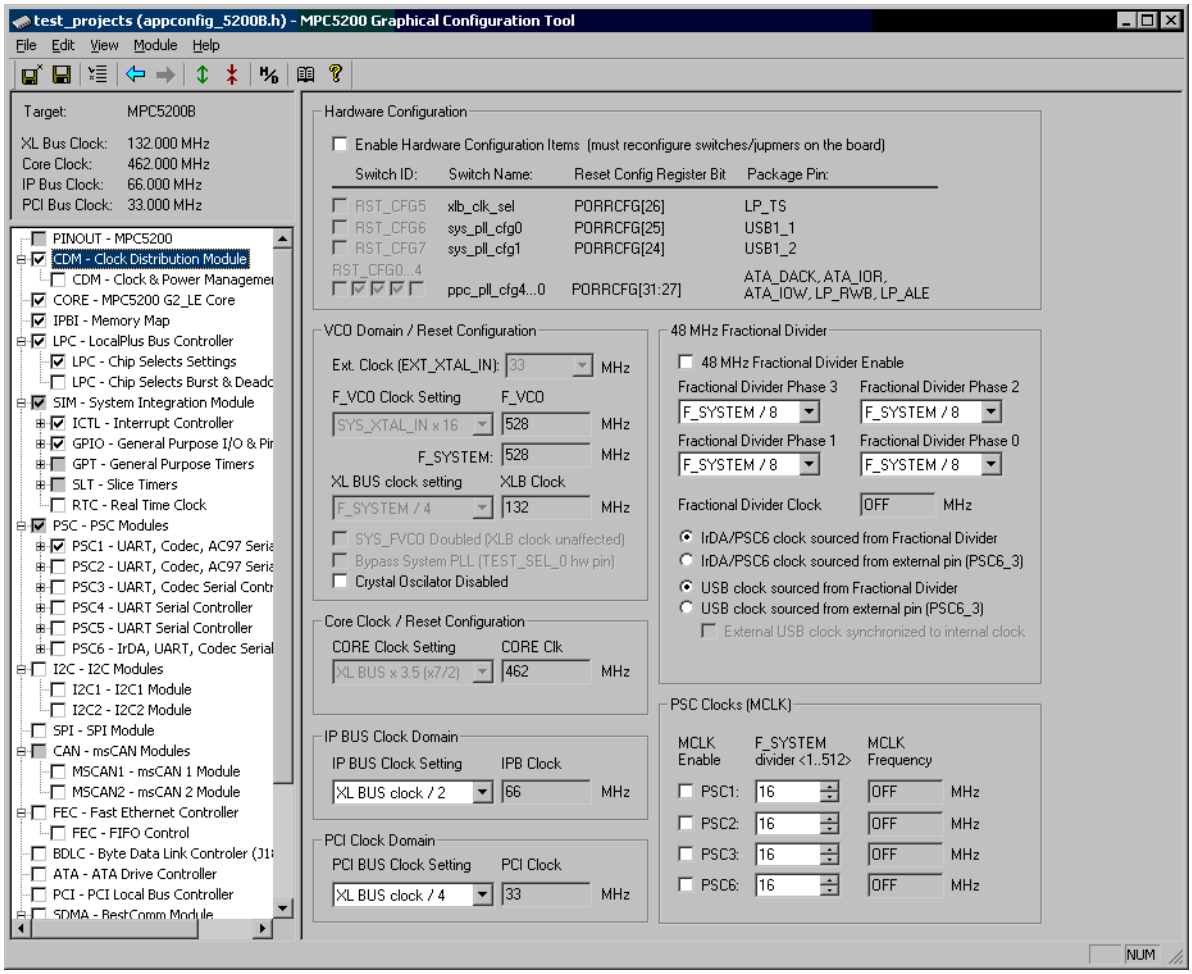

Figure 17. Graphical Configuration Tool, CDM Control Page

The control page of the Clock Distribution Module (CDM) is displayed above in Figure 17. A subset of CDM settings can be physically set only by installing electrical switches (jumpers) on the Lite5200 or Lite5200B boards. The GCT displays such settings grayed and disabled until the user enables it by clicking on the Enable Hardware Configuration Items check box (see Figure 18 below). The hardware settings can

then be modified either by clicking on the jumper check boxes directly or clicking on the graphical controls representing a hardware configuration (XLB Clock, Core Clock, etc.).

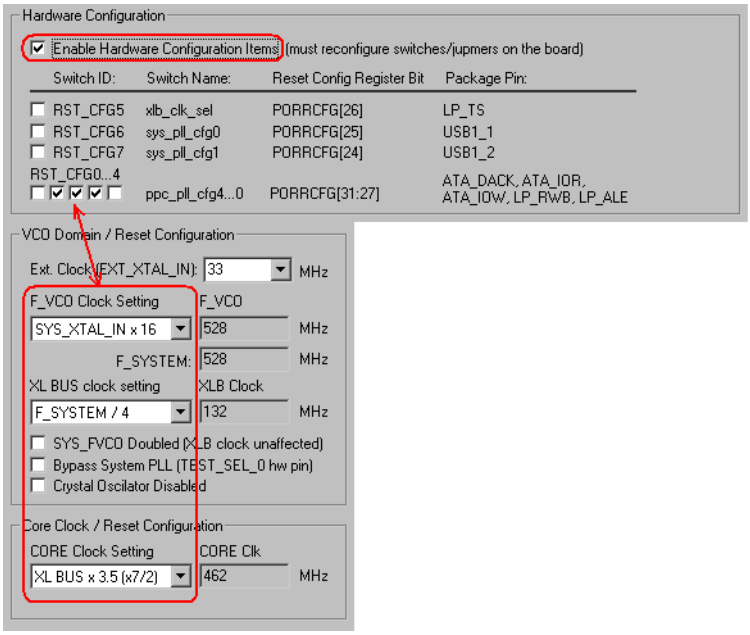

Figure 18. Enabling Hardware Configuration Items

6.4.2 PowerPC Core (CORE)

The CORE control page enables setting of key bits in several PowerPC G2 core registers. Namely, this control page enables configuration of the MPC5200/B Peripheral Base Address register (MBAR), PowerPC core interrupts in the MSR register, Memory Management Unit operation in the MSR and IBAT/DBAT registers, cache control in the HID0 register, and other miscellaneous settings in the HID0 and HID2 registers.

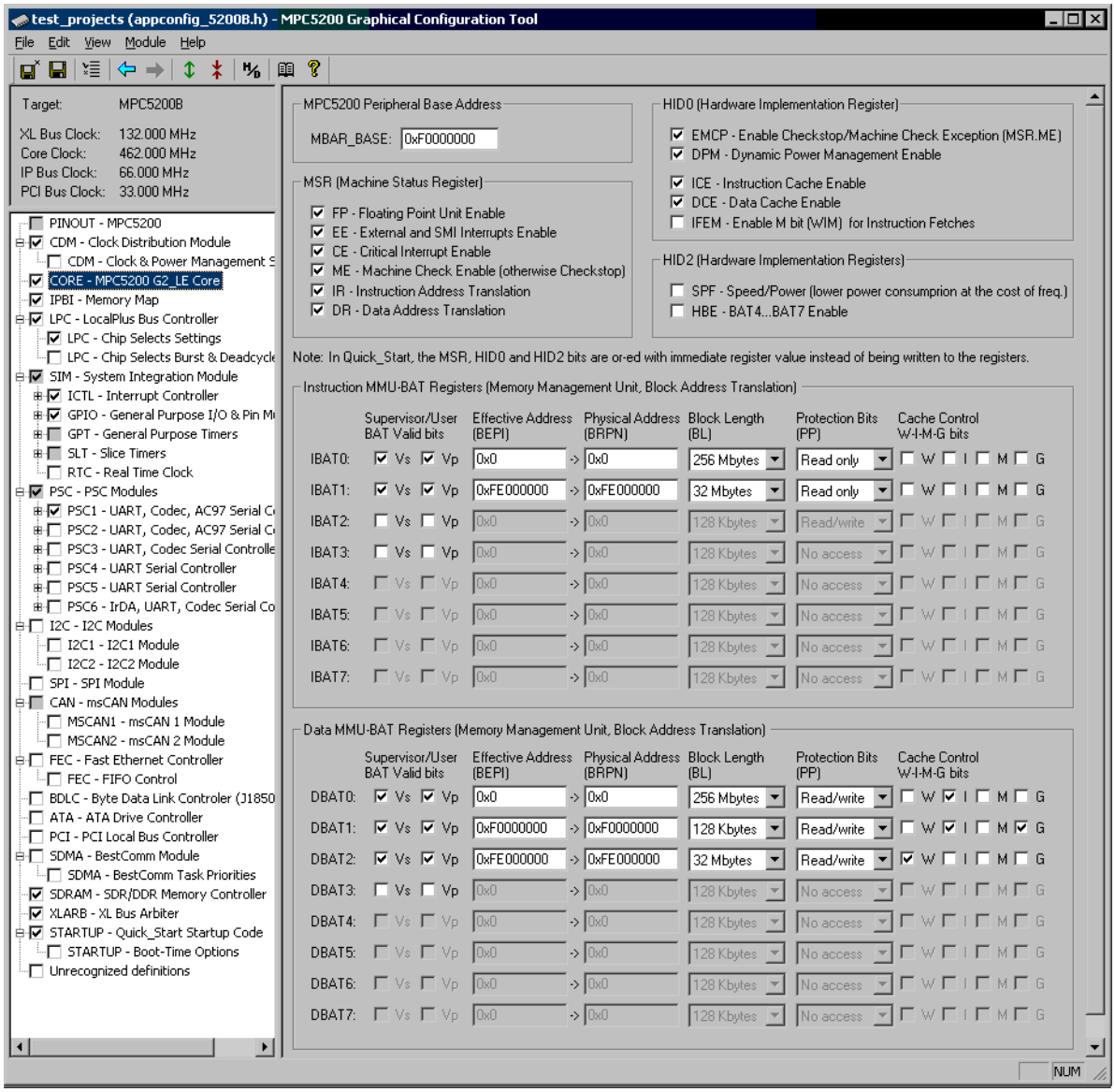

Figure 19. G2 PowerPC Core Control Page

6.4.3 Memory Map Module (IPBI)

The memory map module of the MPC5200 controls an assignment of various memory areas to the LocalPlus Bus interface and the SDRAM Controller.

On this control page, like on many others, there are hypertext links to the logically connected pages in the GCT (LocalPlus Bus page and SDRAM Controller page). GCT strictly follows the structure of MPC5200 peripheral modules; control registers from different peripheral modules are never mixed on a single page. The use of hypertext links is a convenient way how to display a logical connection between separate peripheral modules.

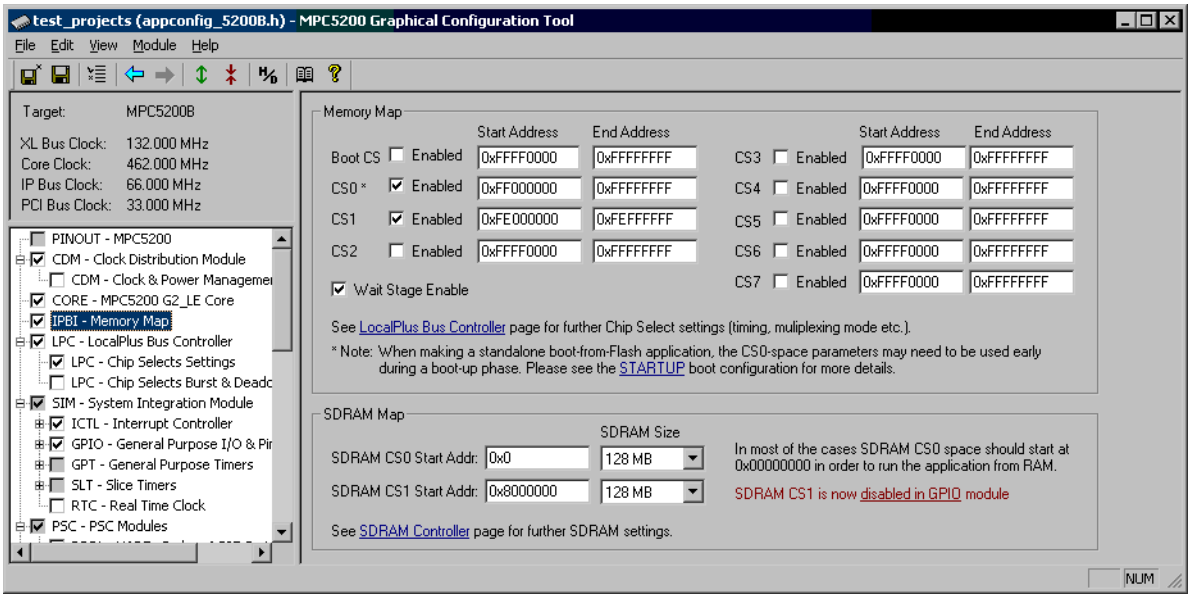

Figure 20. Memory Map Control Page (IPBI)

Another useful GCT feature can also be demonstrated on the IPBI page example. Like many other MPC5200 peripheral modules, there are control registers specifying address values (CS Start and Stop addresses in this case). Control registers often (intentionally) do not implement full 32-bits of the address, and rather implement, for example, only 16 upper address bits assuring the 64 kB address alignment. On the other hand, the GCT always displays the address fields as a full 32-bit value, so it is better understood by the user. When the address is modified in a way that a new value contains bits not implemented in the peripheral register, the user is warned and offered a nearest-aligned address value to be used.

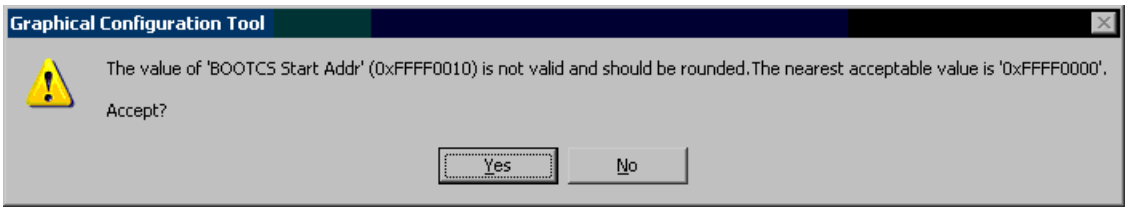

Figure 21. Automatic Address Correction

6.4.4 LocalPlus Bus (LPC)

LocalPlus Bus settings are spread over several control pages in the GCT. Like the CDM module's control page (see Section 6.4.1, "Clock Distribution Module (CDM)),"" the LPC Chip Selects control page displays the settings controlled by a configuration of electrical switches on the MPC5200 board (Boot CS settings in this case). Similarly as on the CDM page, the user must purposely enable modification of hardware-controlled configuration items.

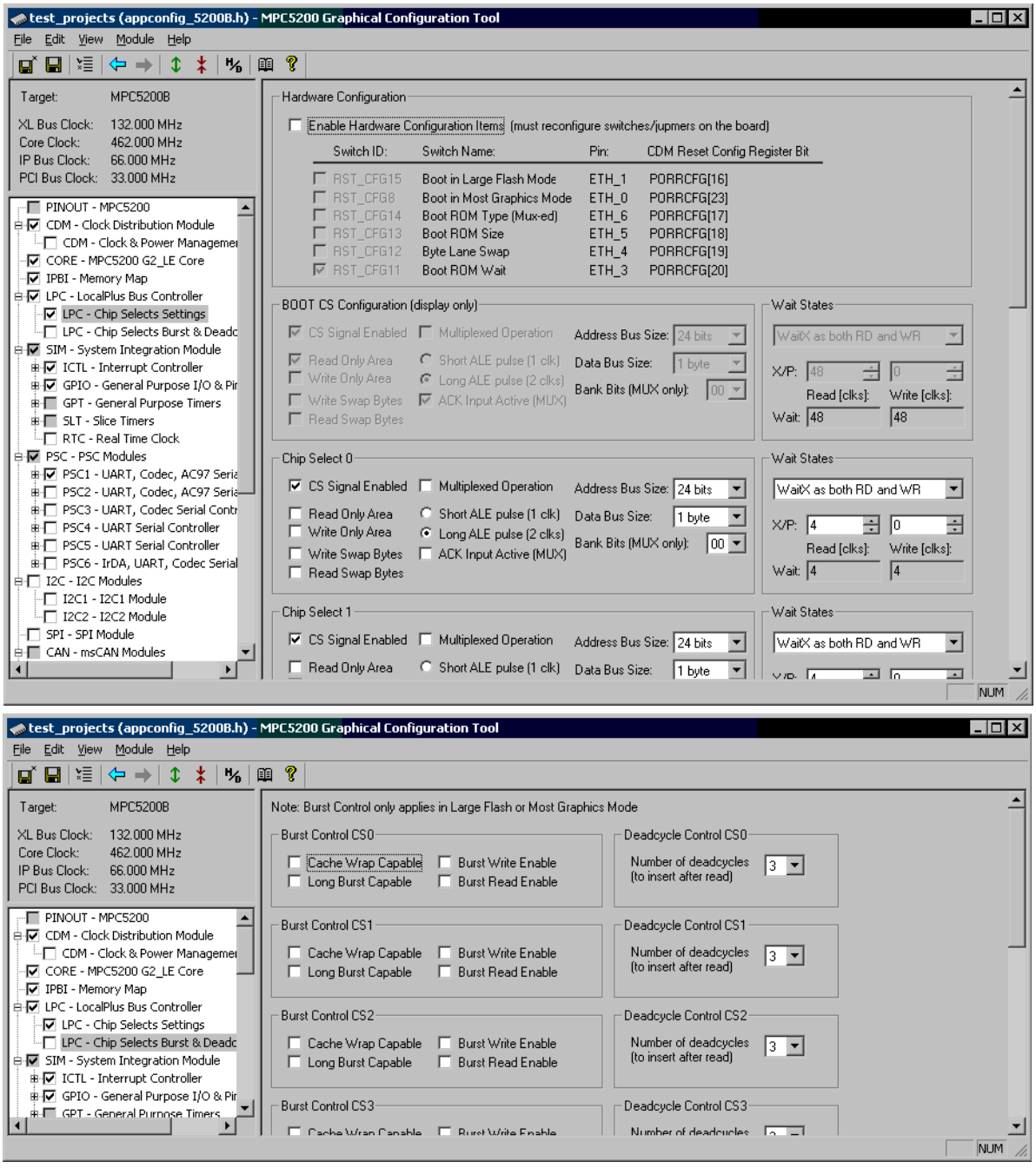

Figure 22. LocalPlus Bus Control Pages (LPC)

6.4.5 System Integration Module (SIM)

The System Integration Module (SIM) of the MPC5200/B contains several peripheral modules, each described and configured on a separate control page in the GCT.

- Interrupt Controller
- General Purpose I/O Module
- General Purpose Timers
- Slice Timers
- Real Time Clock

As there are no settings related to the SIM module as a block, the control page does not contain anything but links to the control pages of individual sub-modules.

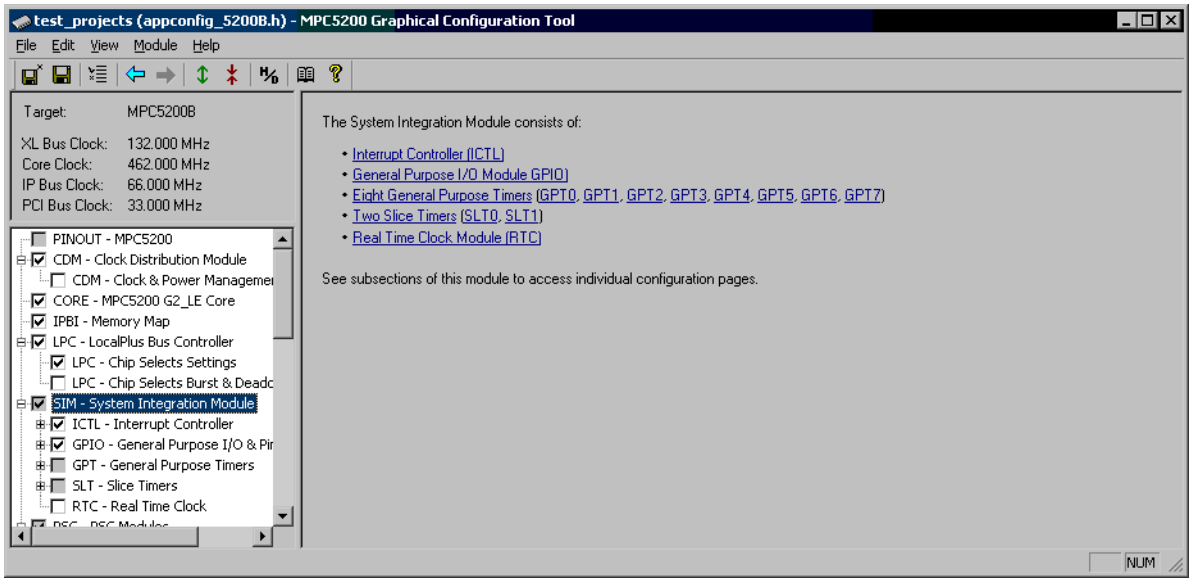

Figure 23. System Integration Module Control Page (SIM)

6.4.6 Interrupt Controller (SIM / ICTL)

The Interrupt Controller (ICTL) pages contain graphical controls for both the hardware registers of the ICTL module as well as the controls for the MPC5200_Quick_Start Interrupt Dispatcher configuration.

Without an Interrupt Dispatcher enabled, only the PowerPC G2 exception service routines can be specified in the GCT (the `vectors.asm` source file is implemented only (see Section 5.4, "Interrupt Dispatcher" for more details).

6.4.6.1 Interrupt Dispatcher

Enabling the Interrupt Dispatcher causes three external exception handlers (0x0500, 0x0A00 and 0x1400) to be hard-routed to the MPC5200_Quick_Start implementation of the Dispatcher and the user is not allowed to specify his own handlers for them. On the other hand, thanks to the Interrupt Dispatcher being enabled, the user is able to specify handler routines for each main and peripheral interrupt source, regardless what physical exception vector they are routed to.

Using the Enable Floating Point Unit check box, a support for Floating Point context saving can be enabled in the Interrupt Dispatcher. A Floating Point context save can then be selectively enabled for individual peripheral interrupt service routines (see Figure 26 below).

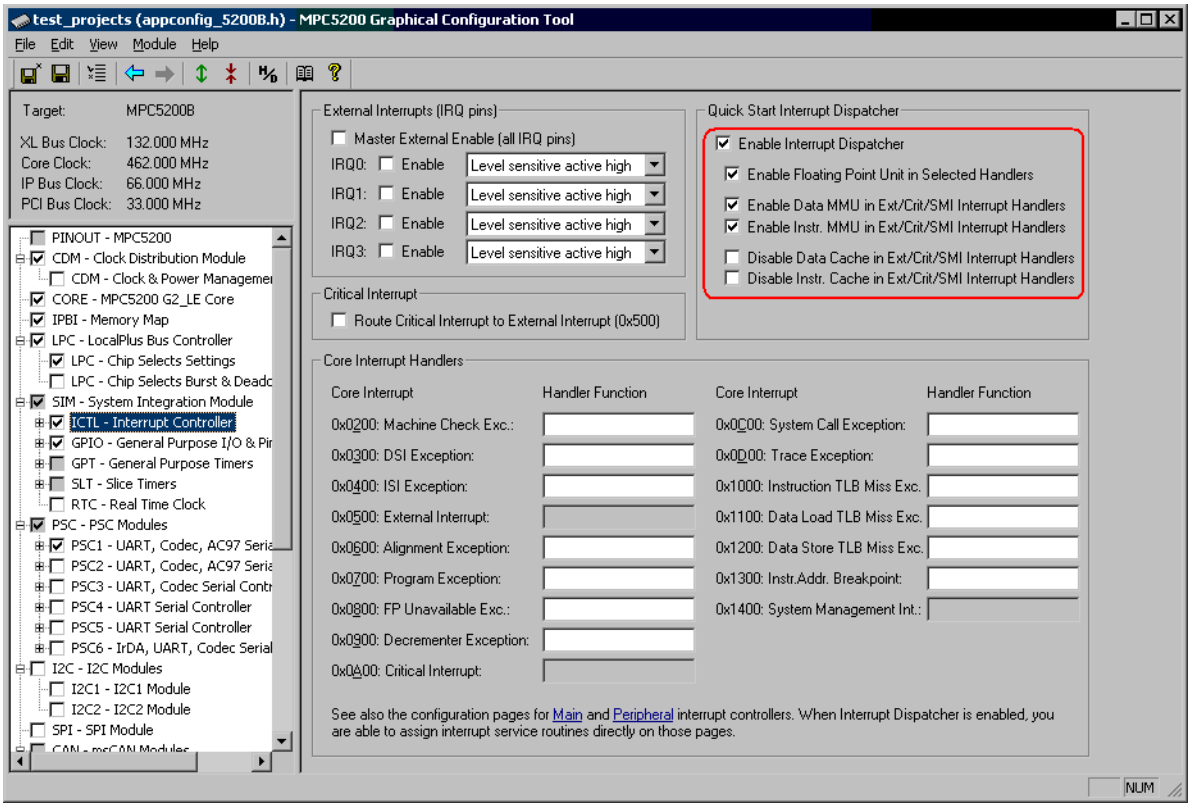

Figure 24. Interrupt Controller Control Page (ICTL)

6.4.6.2 MMU and Cache Control in Interrupt Dispatcher

Since the MPC5200_Quick_Start release 0.9, the Interrupt Dispatcher can also be configured to automatically re-enable the PowerPC MMU before invoking any peripheral interrupt service routine. The reason for implementing such a feature is simple: In typical applications, the user requires data and instruction caches to be enabled to achieve optimal MPC5200/B performance. In most cases though, together with enabling the data cache, the user has to configure the PowerPC MMU as well and specify what areas of memory are not to be cached. For example, it would be a disaster for an application to keep accessing MBAR-based peripheral register space with a data cache enabled. Typically, the user assigns one of the DBAT register pair to declare peripheral register space as cache-inhibit and guarded memory space (I and G bits in WIMG bit-field of DBATxL).

When the PowerPC exception is being processed, the MMU is automatically disabled in MSR register while the caches remain enabled, making the MBAR-based area unusable. The user has to choose one of the two possible actions before accessing the peripheral registers in the interrupt routine. Both of the options are supported natively by Interrupt Dispatcher and GCT.

- **Disable cache** - By disabling the cache, the MBAR space can be safely accessed even without the MMU unit re-enabled. On the other hand, disabling the caches may negatively affect the application performance and prolong the interrupt processing time.
- **Re-enable MMU** - By re-enabling the MMU, the I and G WIMG control bits assure safe MBAR-based area access.

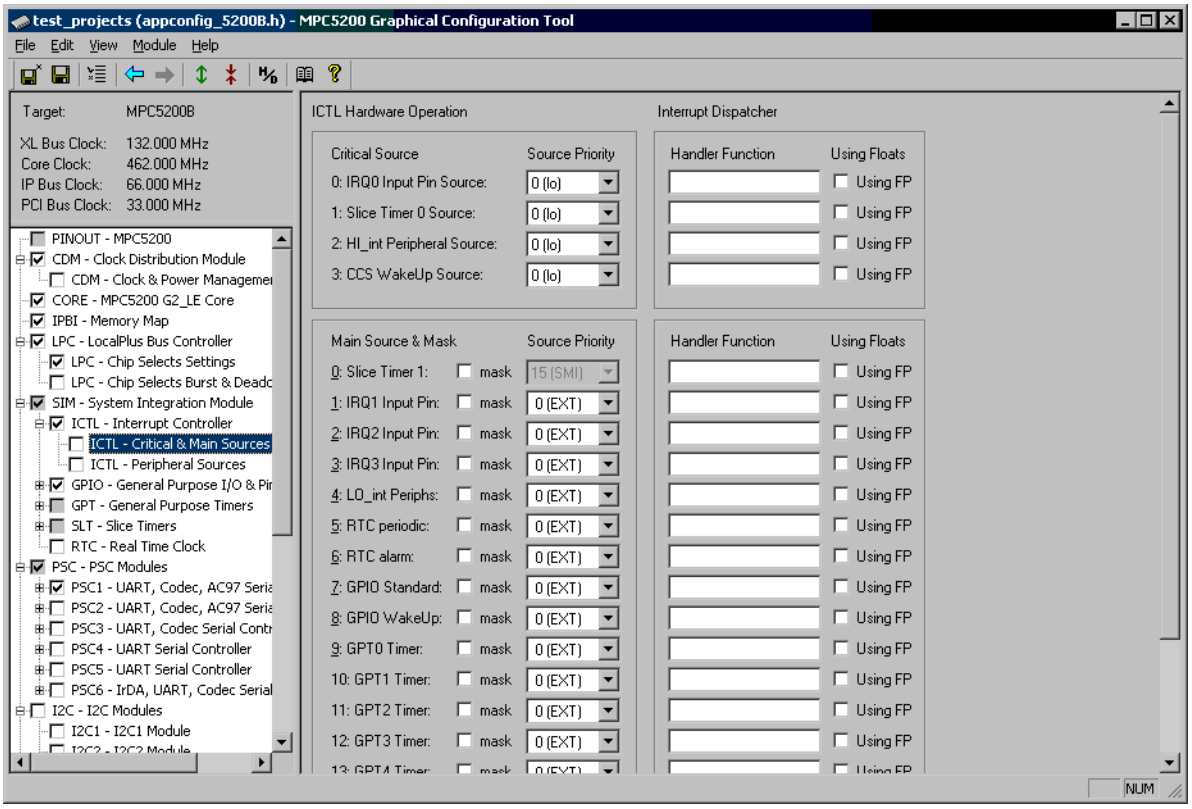

Figure 25. Main Interrupt Controller Control Page (ICTL)

6.4.6.3 Main and Peripheral Interrupt Controllers

There are two sub-pages under the Interrupt Controller page, one for each Interrupt Priority Decoder of the MPC5200/B. The first page (Figure 26) displays four Critical Priority interrupt sources and all 17 Main Interrupt Controller sources. The second page (Figure 26) displays all 24 Peripheral Interrupt Controller sources.

Each source can be assigned a relative priority and can be generally masked (disabled) or unmasked (enabled). For the Main Interrupt Controller sources, the priority selection also defines which PowerPC core interrupt is physically generated for each source (either External 0x500 interrupt or SMI 0x1400 interrupt).

As the Peripheral Interrupt decoder operates as a client to both Main Interrupt decoder and Critical Interrupt decoder, the priority selection of each Peripheral source also defines whether the main interrupt source (LO_int) or Critical Interrupt source (HI_int) will be used by the peripheral source.

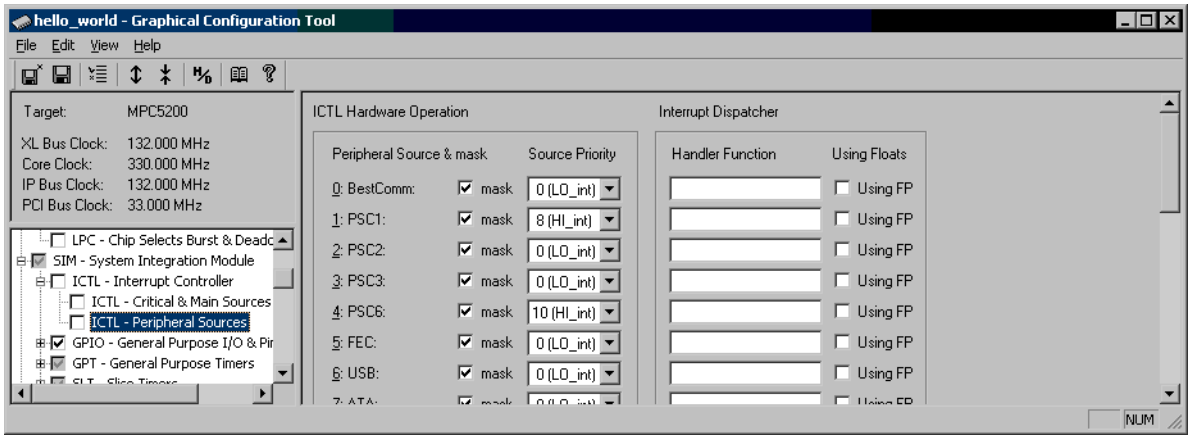

Figure 26. Peripheral Interrupt Controller Control Page (ICTL)

6.4.7 General Purpose I/O (SIM / GPIO)

A configuration of a General Purpose I/O is spread over seven control pages (eight on MPC5200B). All GPIO control registers are assigned to the main GPIO control page only, so only the root GPIO item in the MPC5200 peripheral tree view contains the `appconfig.h` output check box (see Section 6.2.1, "GCT Input/Output File - appconfig.h"). The rest of GPIO pages are display-only, and their graphical controls are linked to the parent item's control registers. This is why the check boxes at those pages are disabled and the pages cannot be selectively excluded or included in the `appconfig.h` output.

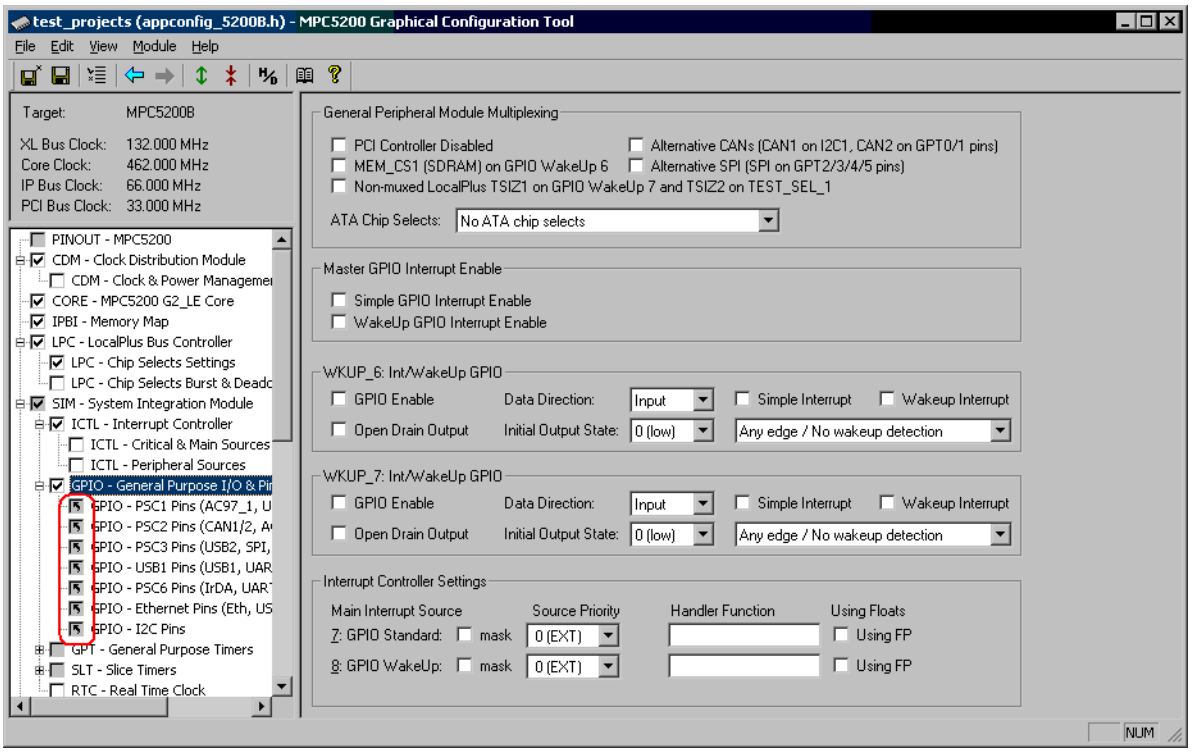

Figure 27. General Purpose I/O Control Pages (GPIO)

6.4.8 General Purpose Timers (SIM / GPT)

There are eight General Purpose Timers in the MPC5200/B. The GCT control page enables a complete configuration of all timer features.

The GCT page is also a good example of a control page displaying mirrored settings from the Interrupt Controller page. The Interrupt Controller settings (interrupt source, mask, handler routine and Floating Point context save) are displayed at the bottom side of the GPT control page. Any changes made in the interrupt controller settings on this page will be automatically displayed also on the Main Interrupt Controller control page, and vice versa (see Section 6.4.6, “Interrupt Controller (SIM / ICTL)).”

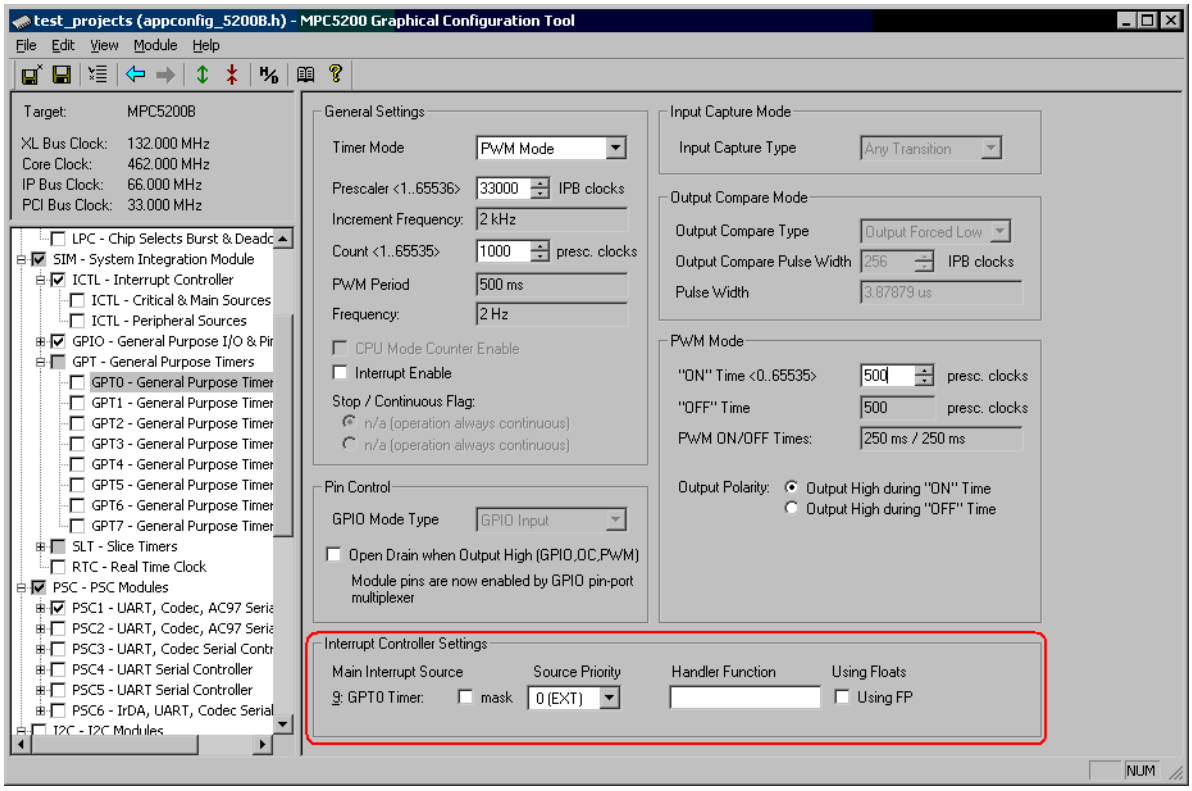

Figure 28. General Purpose Timer Control Page (GPT)

6.4.9 Slice Timers (SIM / SLT)

There are two precise Slice Timer modules (SLT) on the MPC5200/B. The configuration page enables complete SLT configuration as well as assigning an SLT interrupt service routine.

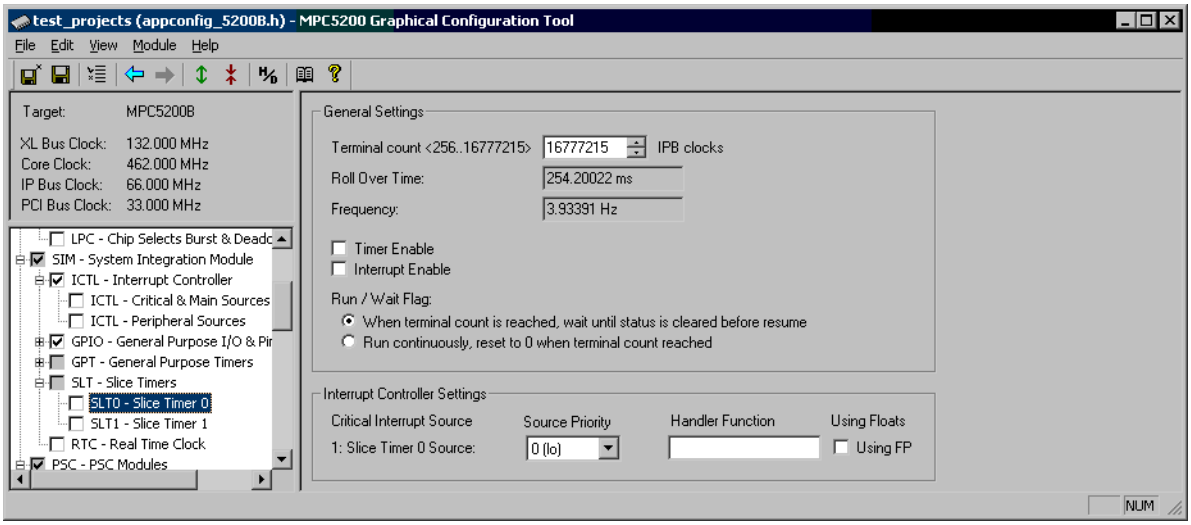

Figure 29. Slice Timer Control Page (SLT)

6.4.10 Real Time Clock Module (SIM / RTC)

There is one Real Time Clock (RTC) module on the MPC5200/B. The configuration page enables complete RTC configuration, specifying post-reset initial timer value as well as assigning the RTC interrupt service routines.

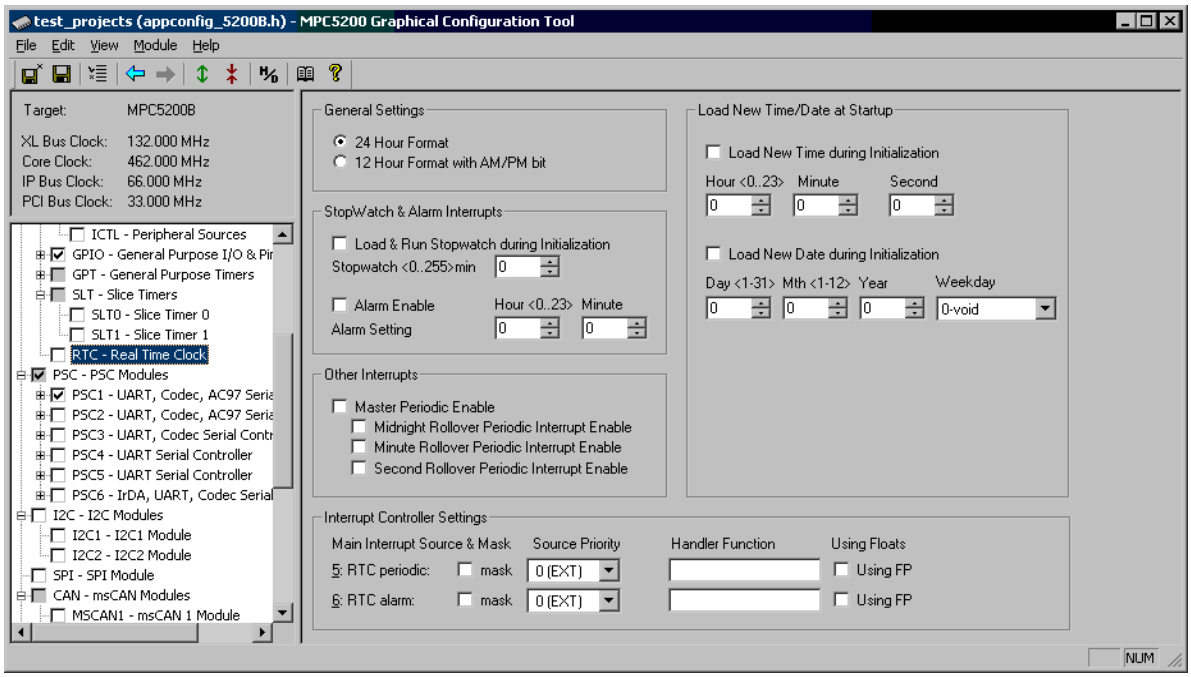

Figure 30. Slice Timer Control Page (SLT)

6.4.11 Programmable Serial Controller (PSC)

There are six Programmable Serial Controllers (PSC) on the MPC5200/B, each of which can be configured for different modes of operation (UART, Codec, Codec-SPI, Codec-I2S, AC97, IrDA). As not all of the PSC modules support all operation modes, and the PSC control registers are mostly used in a different way in each mode, setting up the PSC manually is a complicated task. Thanks to its graphical interface, a GCT can save much work when configuring the PSC for the required operation.

For each PSC mode selected by a drop-down list box at the top of the page, the content of the control page changes and displays only the graphical controls applicable to the selected mode. Figure 31 shows the PSC control page in the UART mode.

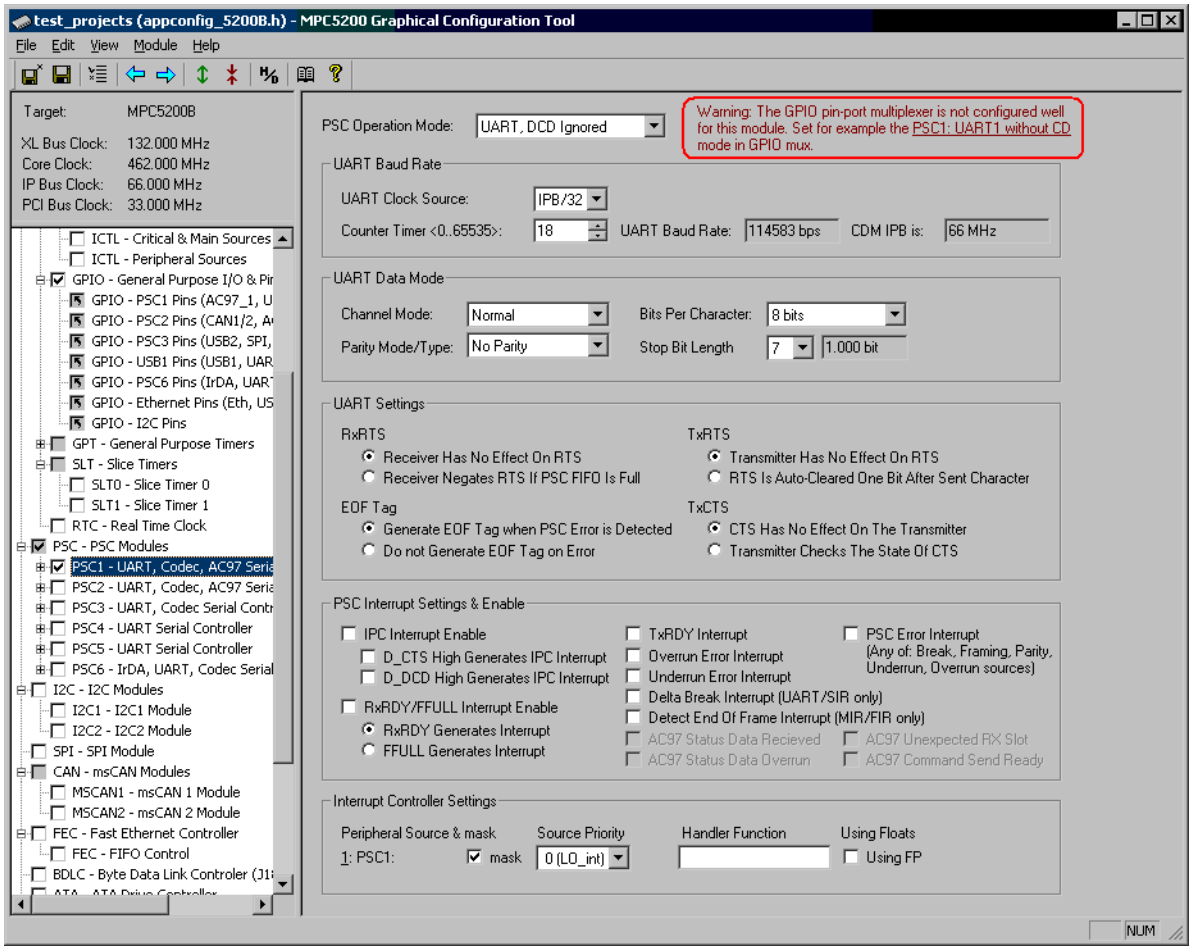

Figure 31. Programmable Serial Controller (PSC), UART Mode

Figure 31shows another GCT feature, not yet described before in this document. As the MPC5200/B hardware pins are mostly shared between different peripheral modules, there is a pin-port multiplexer which is part of the SIM GPIO module. The port multiplexer must be configured properly for a given operational mode of each peripheral module that requires any MPC5200 device pins. For example, the PSC in the UART mode requires the port multiplexer to be configured a different way than when using the PSC in Codec mode. A GCT displays a red warning message when there is any inconsistency between operation mode of the peripheral module and the current GPIO port multiplexer settings. In most cases,

the warning message contains hypertext link or links which activate the GCT page where a change is needed or an attention is required.

Figure 32 shows the PSC control page in Codec (I2S). The Figure 33 shows the PSC control page in AC97 Mode.

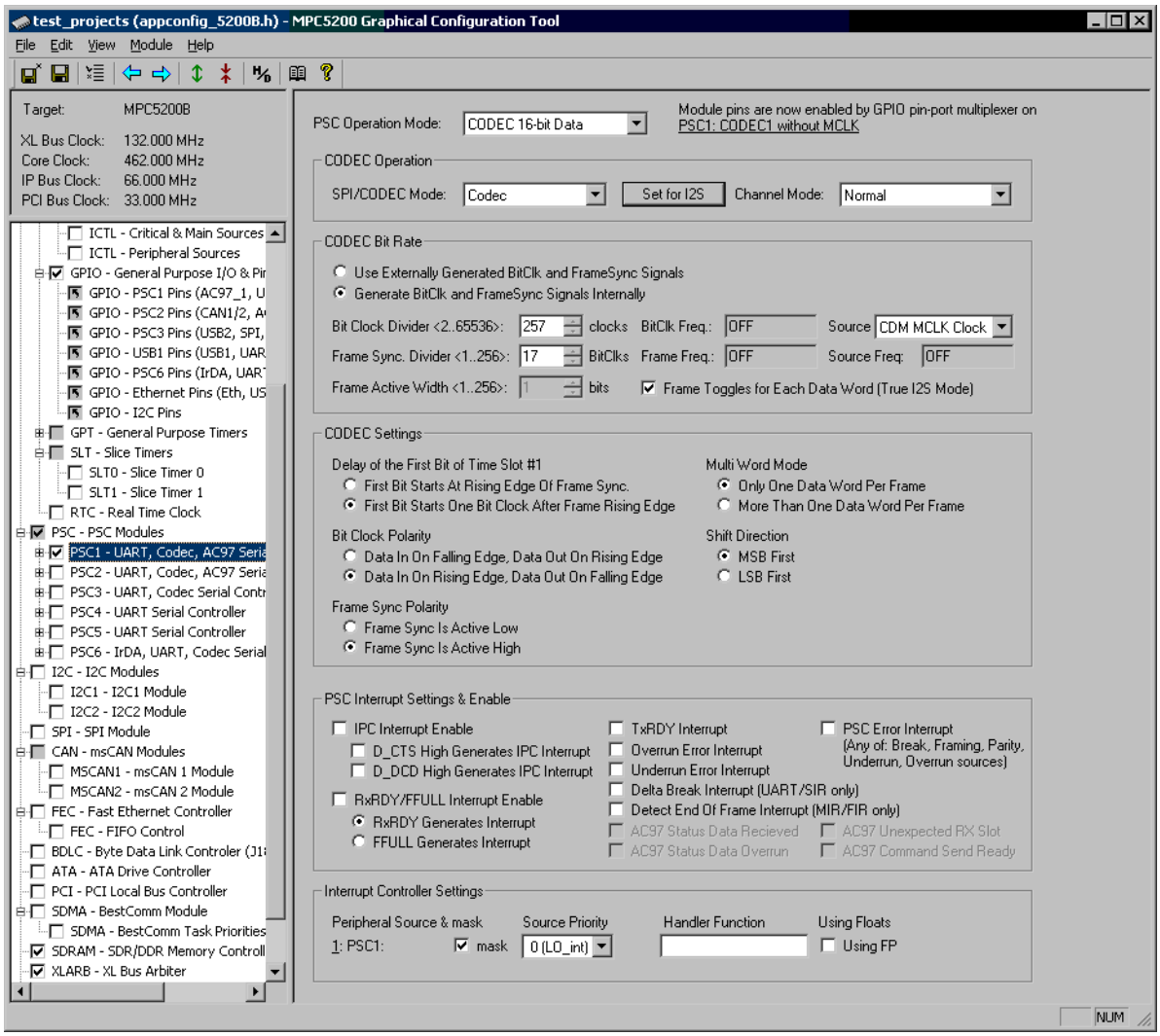

Figure 32. Programmable Serial Controller (PSC), Codec Mode

Like many other peripheral modules, each PSC contains a FIFO to enable buffered data operation and BestComm DMA access. There is a sub-page under each PSC module page which contains a graphical controls to set up a FIFO parameters (Figure 34).

On the FIFO page, there are also several check boxes not bound directly to any PSC control register. These controls enable users to select what actions are performed on the PSC during initialization phase (see Section 7, “Module Initialization Code”).

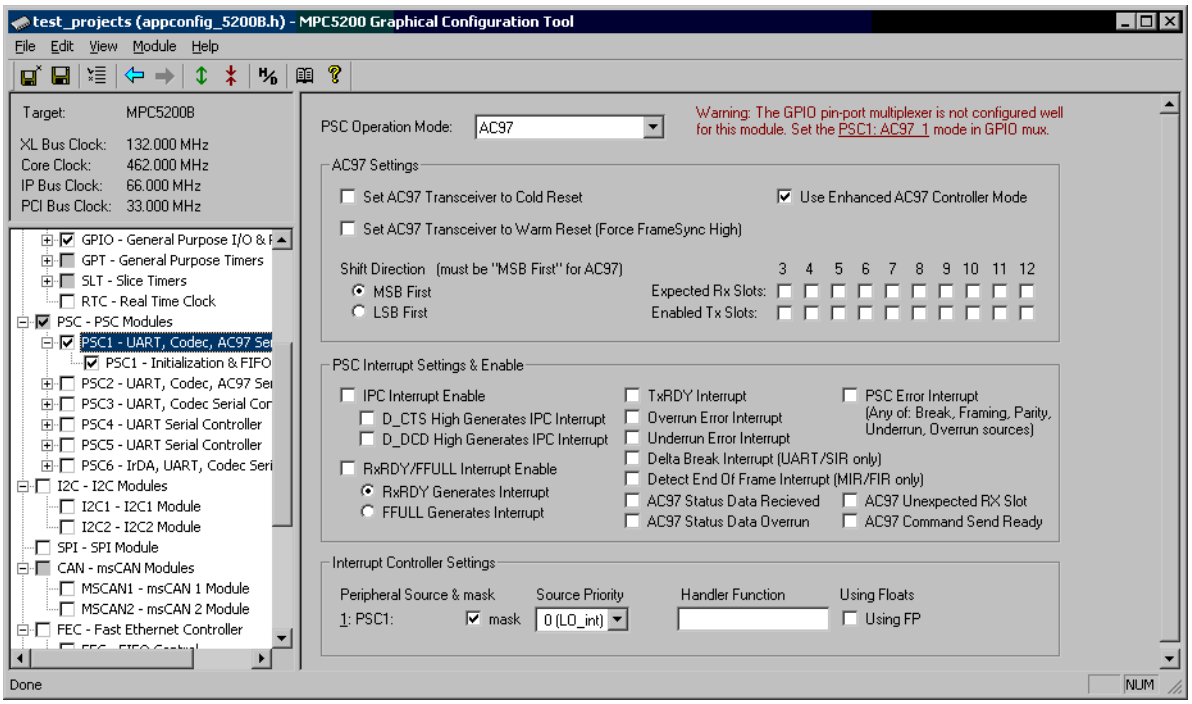

Figure 33. Programmable Serial Controller (PSC), AC97 Mode

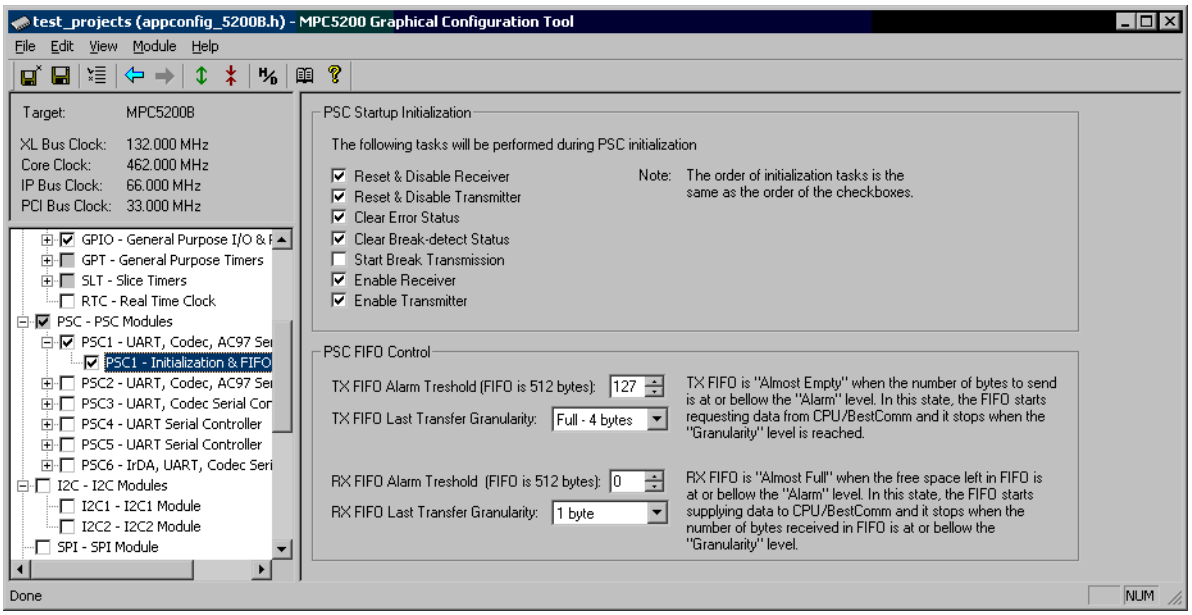

Figure 34. Programmable Serial Controller (PSC), FIFO Interface

6.4.12 I²C Controller (I2C)

There are two I²C Bus Controllers on the MPC5200. Like for many other modules, the GPIO port multiplexer should be set properly for the I²C module. The red message displays a hypertext warning when the port multiplexer is not configured well for the I²C operation.

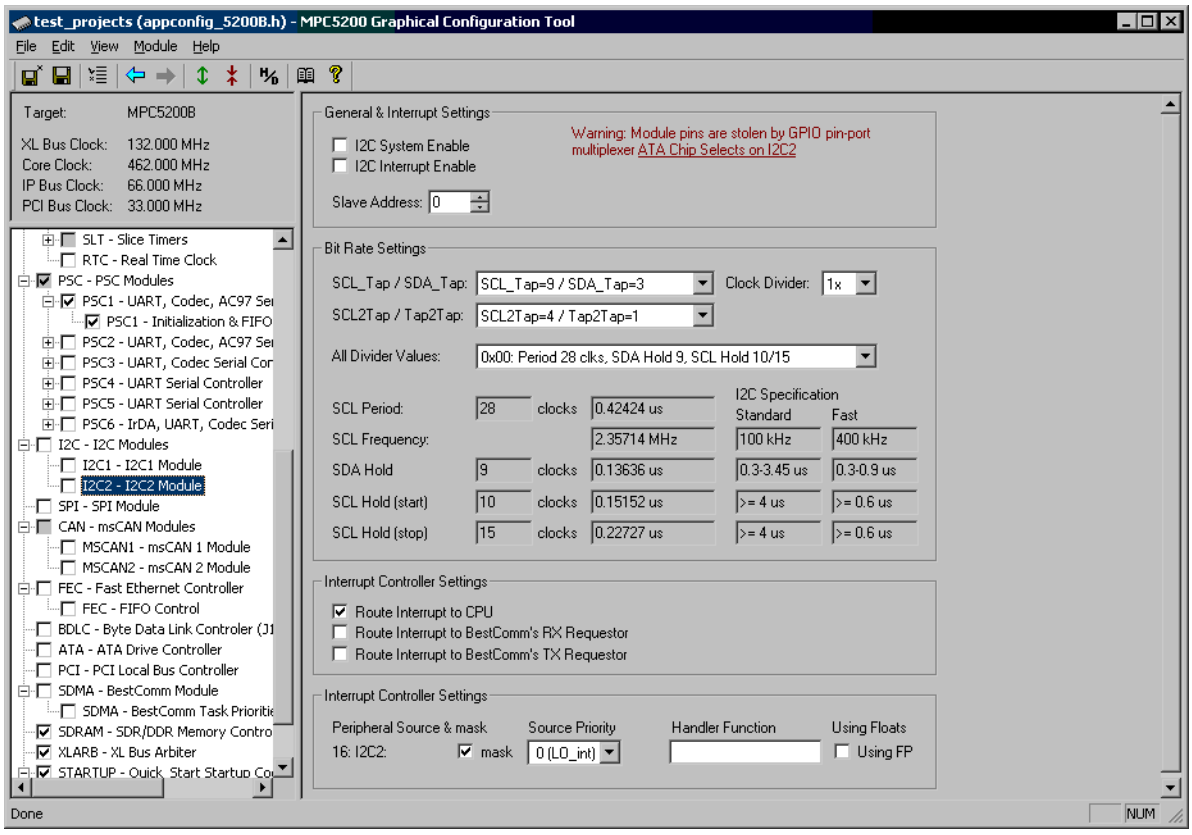

Figure 35. I²C Controller Control Page (I²C)

6.4.13 Serial Peripheral Interface (SPI)

In addition to Programmable Serial Controller modules which do support the Serial Peripheral Interface (SPI) mode, there is also one dedicated SPI module on the MPC5200/B. The SPI interface uses at most four pins which must be properly configured by both the GPIO port multiplexer and the SPI module itself. In addition to the GPIO port settings, each pin of the SPI should also be configured for input or output directly in the SPI module.

The GCT displays all four SPI pins together with their operation assigned by the SPI. In the case of any conflict (some SPI operation modes require pins to be in certain mode), a red warning message is displayed together with a hint of how to fix the problem (such as set pin as output).

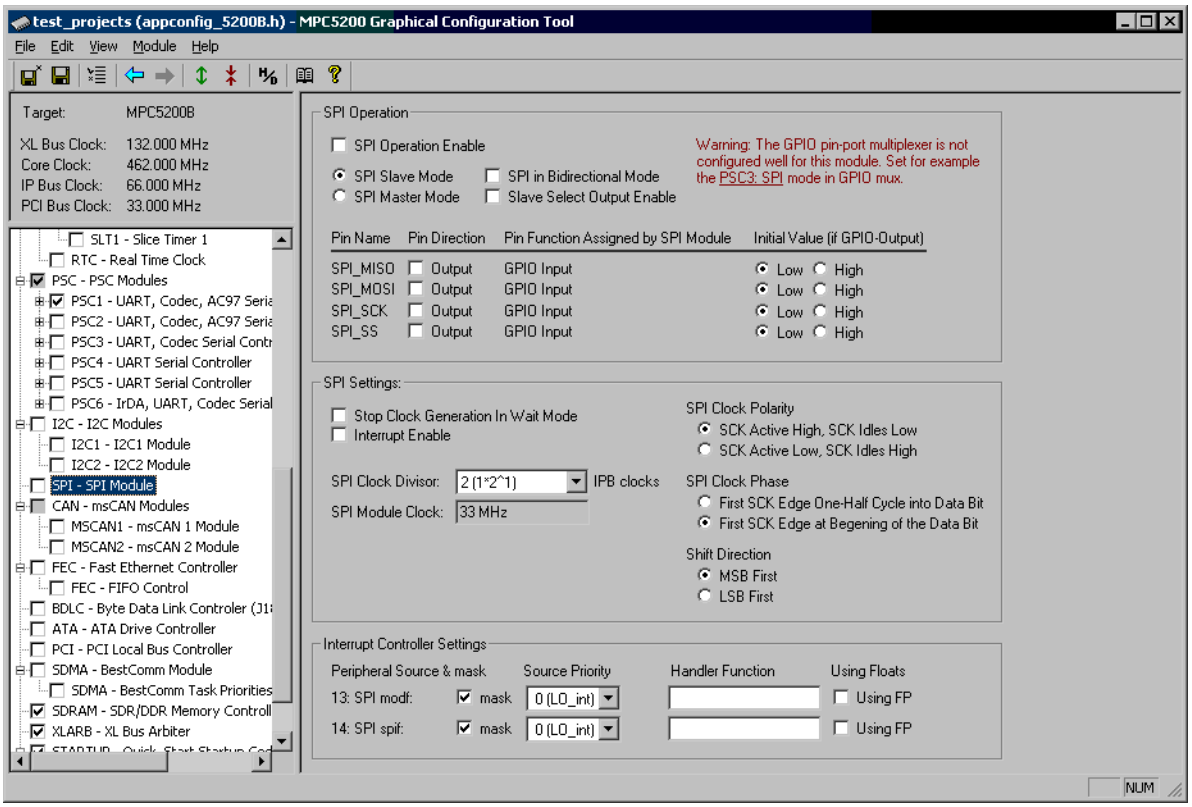

Figure 36. Serial Peripheral Controller Control Page (SPI)

6.4.14 CAN Controller (MSCAN)

There are two CAN interfaces (MSCAN) on the MPC5200/B, compatible with a standard Controller Area Network 2A and 2B protocol.

The GCT can help especially with setting communication speed and bit timing parameters of the CAN interface. The user is able to specify a desired communication speed directly in bits-per-second units. Pressing the Calculate Parameters button builds a list of all timing parameter combinations suitable for the requested speed. All important time-quanta portions and sampling points are also drawn on a time-graph which is displayed together with the numeric parameter values.

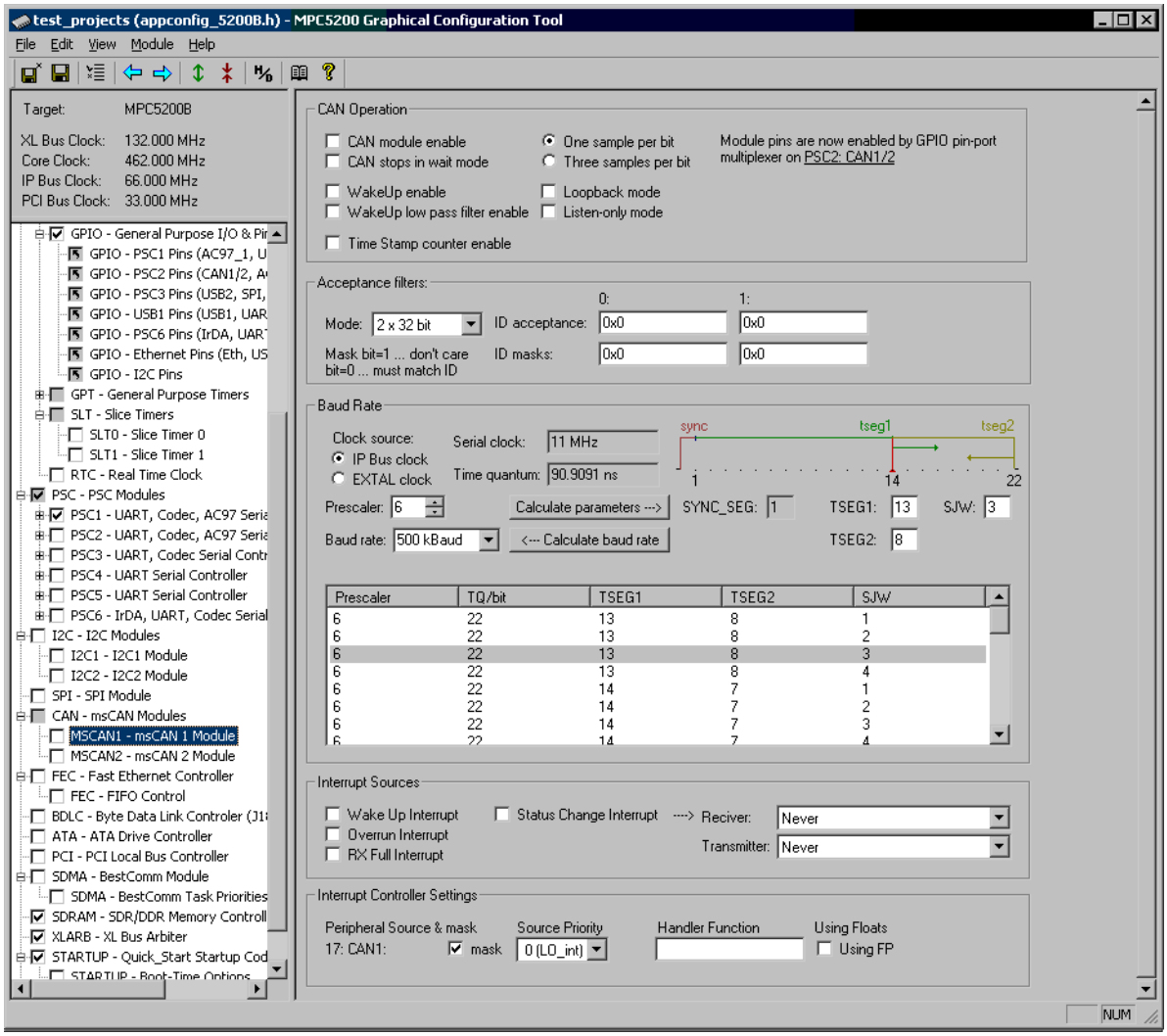

Figure 37. Controller Area Network Control Page (CAN)

6.4.15 Fast Ethernet Controller (FEC)

The Fast Ethernet Controller of the MPC5200/B implements an interface to the standard 10/100 MB IEEE 802.3 ethernet network. For its operation, it requires an external ethernet transceiver (PHY) with which the MPC5200/B communicates either directly (AMD industry standard interface) or over the Media Independent Interface (MII) defined by IEEE 802.3 standard.

The GCT can be used to define initial configuration of the FEC controller as well as to specify parameters which are to be downloaded into the ethernet transceiver over the MII. Two key PHY registers can be configured graphically in the GCT (the control and auto-negotiation registers). There is also a way to specify initial values of the custom PHY registers not defined by the IEEE standard, for example, the LED control registers of the Intel LXT971 used on the Lite5200 board.

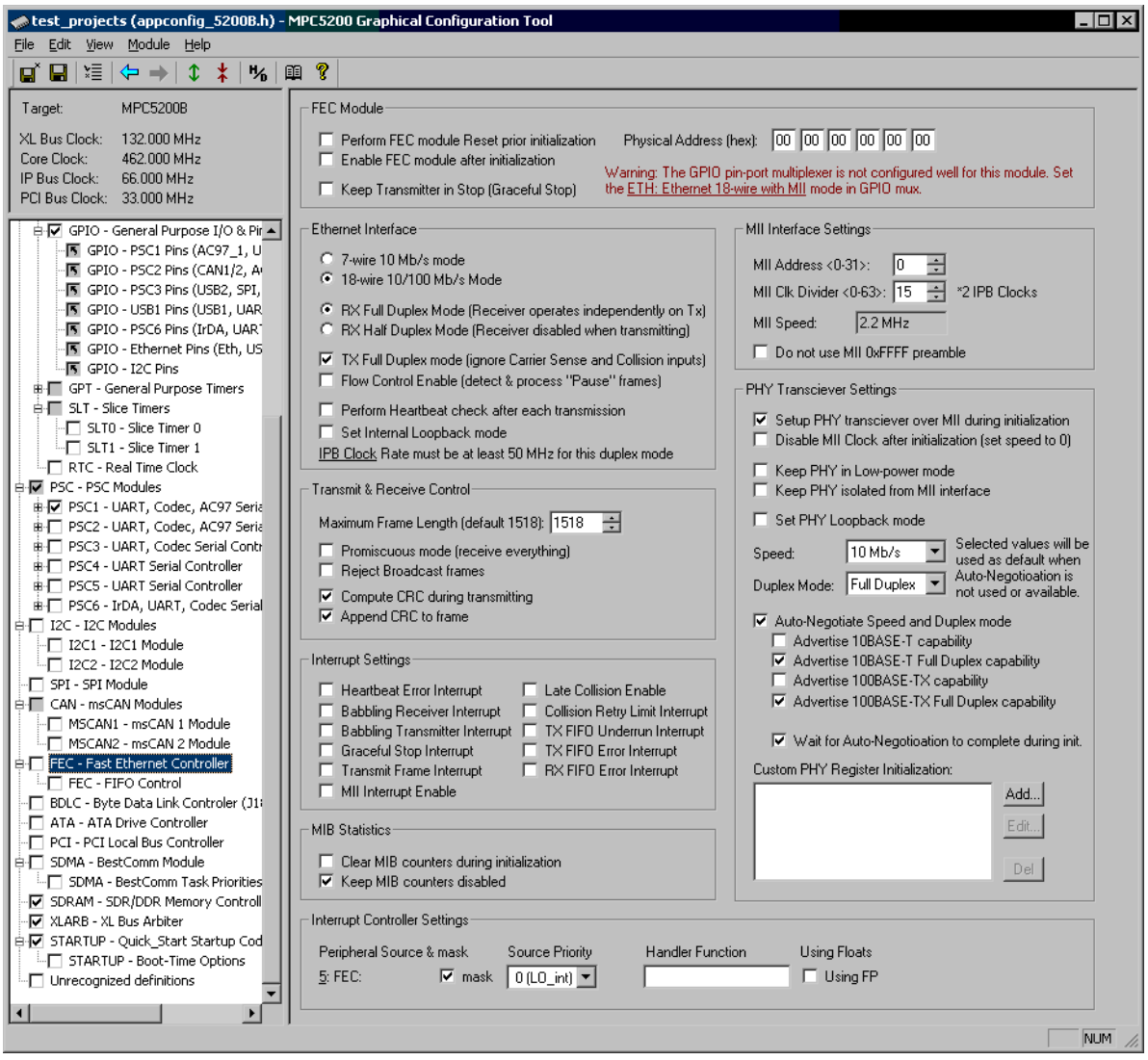

Figure 38. Fast Ethernet Controller Control Page (FEC)

6.4.16 ATA Hard Drive Controller (ATA)

The ATA Interface of the MPC5200 enables connection of the standard ATAPI-4 hard drive to the MPC5200. Although the most of the hard drive operations are performed in run-time (where GCT cannot help), there are several timing control registers which need to be configured before accessing the ATA bus. GCT automatically calculates timing parameters to be compliant with the ATAPI-4 standard. The only input to the calculations is current MPC5200 peripheral bus (IPB) frequency, so there are only few options to be configured on the ATA control page. The timing parameters for all PIO, MDMA and UDMA modes are saved to the `appconfig.h` file so the user does not need to calculate these values in run-time.

With the support of GCT-generated values, the MPC5200_Quick_Start ATA initialization code is capable to detect the ATA Hard Drives and to initialize automatically the best timing parameters suitable for both drives (or single drive) connected.

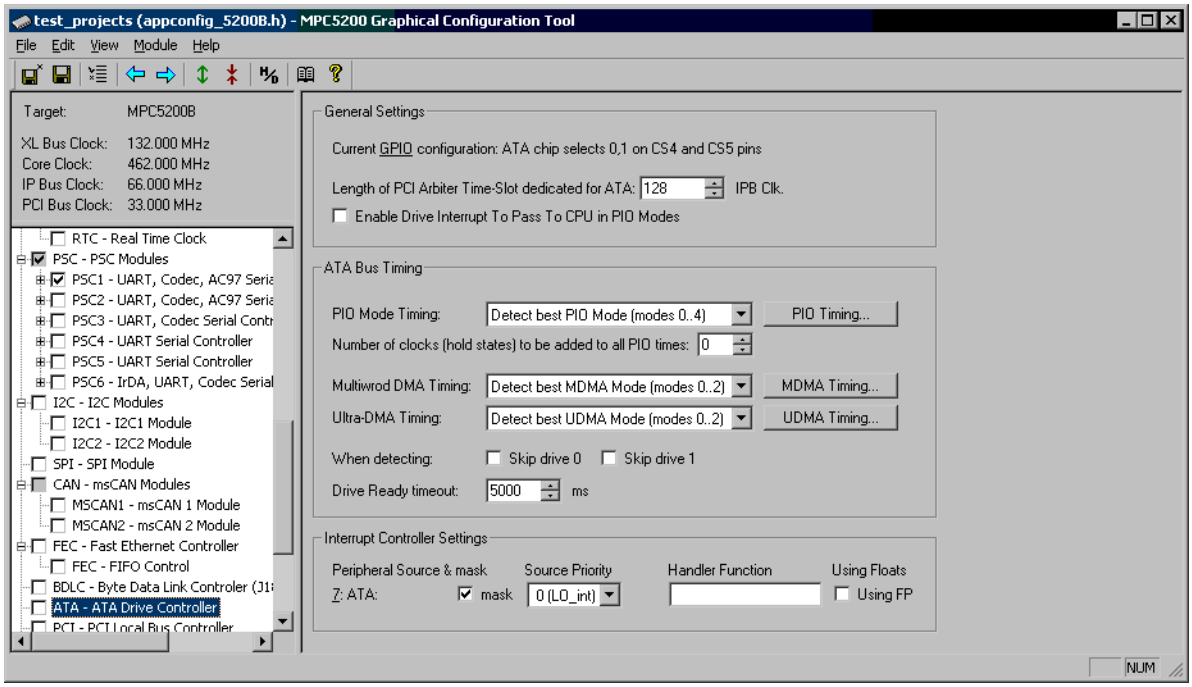

Figure 39. ATA Hard Drive Interface Control Page (ATA)

6.4.17 PCI Local Bus Controller (PCI)

There is a PCI Local Bus Interface bridge on the MPC5200/B. The GCT Control Page can be used to configure the standard PCI Configuration Space of the MPC5200/B, accessible by other devices on the PCI bus. Also, the page contains the XLB-to-PCI memory mapping windows and other parameters used when MPC5200/B accesses the PCI bus as an initiator.

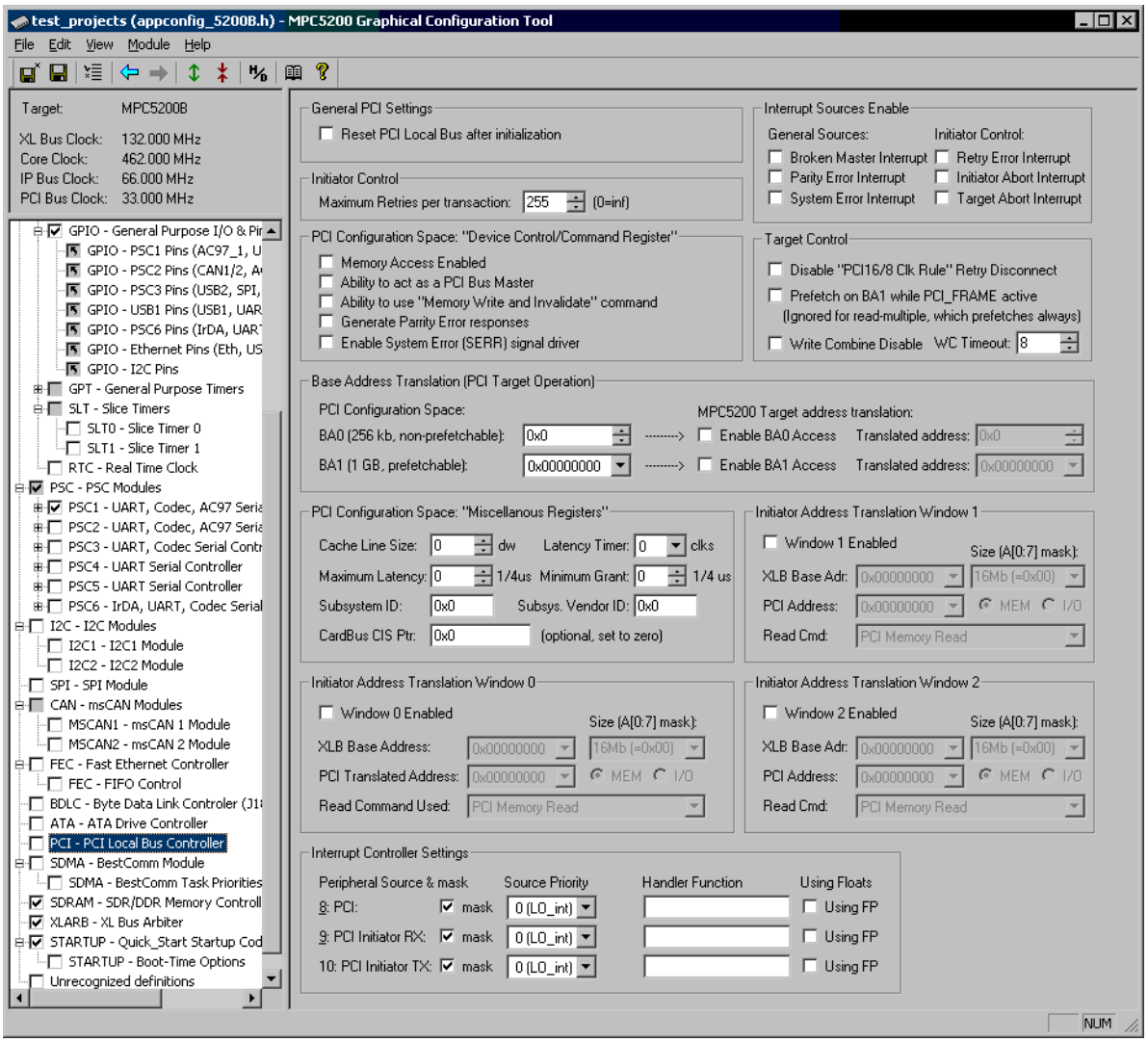

Figure 40. PCI Local Bus Control Page (PCI)

6.4.18 BestComm DMA Module (SDMA)

The BestComm DMA module is the only MPC5200 module for which the MPC5200_Quick_Start initialization code uses an external embedded-side code. There is a complete BestComm support in the BSP and BestComm API (BAPI) package; both the GCT and MPC5200_Quick_Start code re-use this implementation.

The latest BSP code as well as the standard RTOS DMA images are included in the MPC5200_Quick_Start distribution, so there are no external resources needed to implement DMA operations in Quick Start applications. See Section 9, "MPC5200 BSP" for more details about BSP package.

The GCT support for the BestComm DMA module is narrowed down to configuration of BestComm control registers and the BSP initialization sequence. There are two control pages dedicated to the BestComm module in the GCT: One for general BestComm and BAPI initialization (Figure 41), and one for selecting individual BestComm task priorities.

Except other settings, the main BestComm control page contains a button to run the BestComm GCT for a current project, provided it is based on the DMA_Custom template. The BestComm GCT can be used to build custom BestComm microcode images which are rather complicated and tricky. For a vast majority of applications, the RTOS Image-based projects provide enough built-in DMA functionality within the pre-compiled BestComm images. In this case, the BestComm GCT is not needed at all. BestComm GCT is not part of the MPC5200_Quick_Start installation.

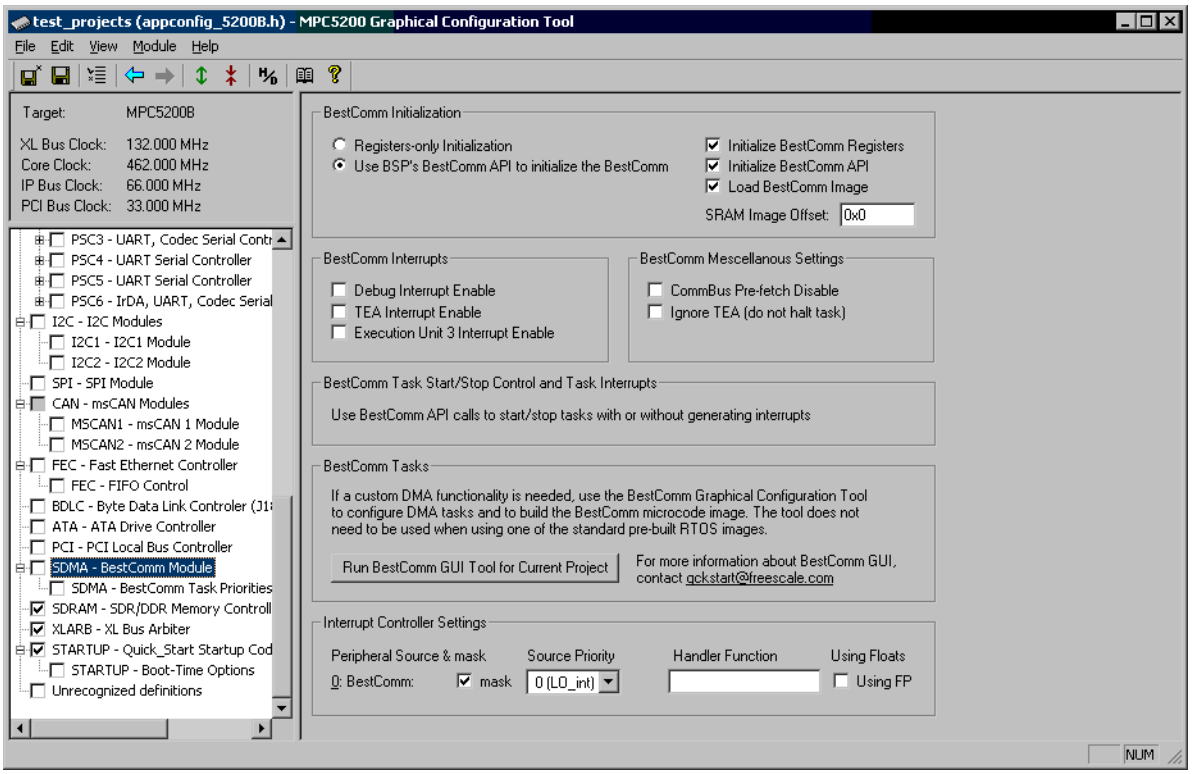

Figure 41. BestComm Control Page (SDMA)

6.4.19 SDRAM Memory Controller (SDRAM)

The SDRAM controller must be configured properly in the MPC5200 system to enable RAM operation, which is crucial for vast majority of applications. As it is not always easy to determine the proper SDRAM controller settings for a given memory device, the GCT contains a database of valid settings for the most commonly used SDRAM and DDRAM devices.

The central portion of the SDRAM GCT page displays the SDRAM settings for the selected memory device and a clock speed as retrieved from the database. The user is not allowed to modify the settings unless he purposely enables them using the check box at the bottom side of the page. When modified, the database can be updated with new values. A database is saved as a standard INI file format in the `sdram.ini` file located in the `sdram` sub-directory of the GCT application folder. An advanced user can use a text editor to modify or to add new entries to the database.

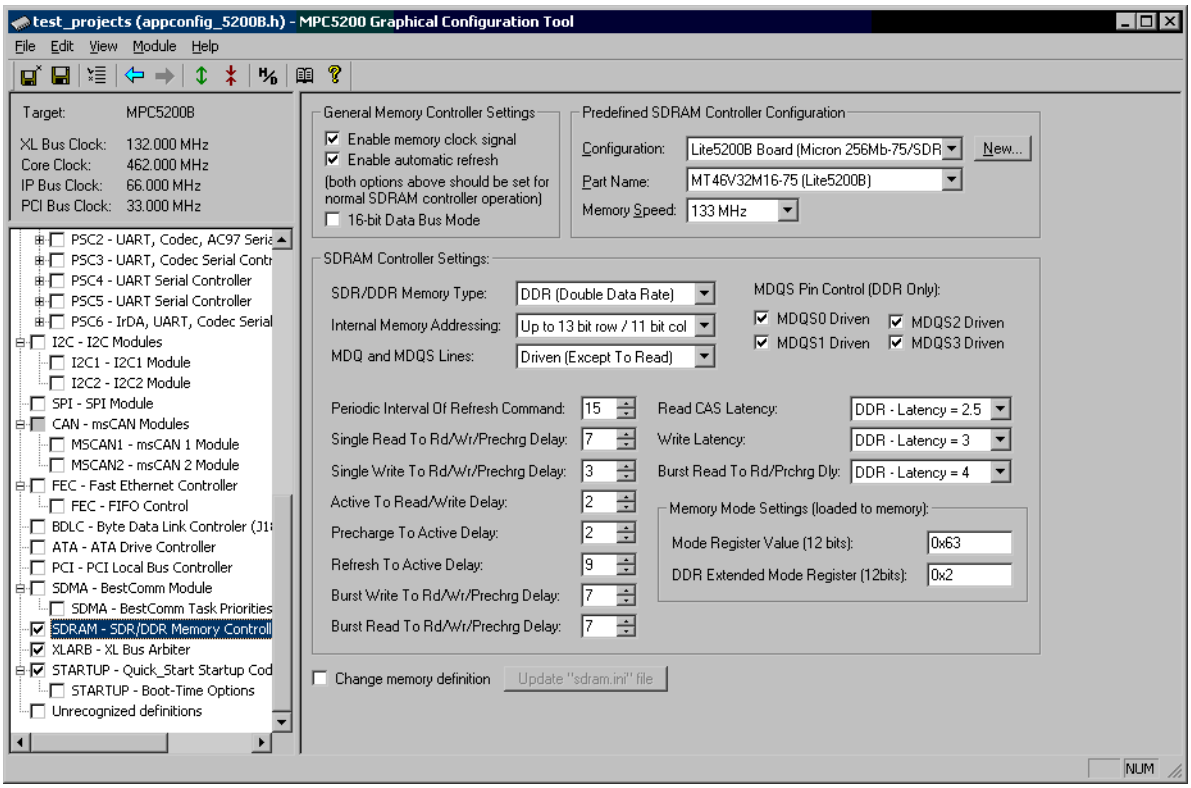

Figure 42. SDRAM Controller Control Page (SDRAM)

In addition to the direct modification of the memory database file, a new memory device can be defined by specifying the memory timing parameters from the data sheet. When the New button is pressed on the SDRAM GCT page, the dialog window appears (Figure 43), and the user is able to specify a data-sheet timing parameter for any clock speed which is to be supported.

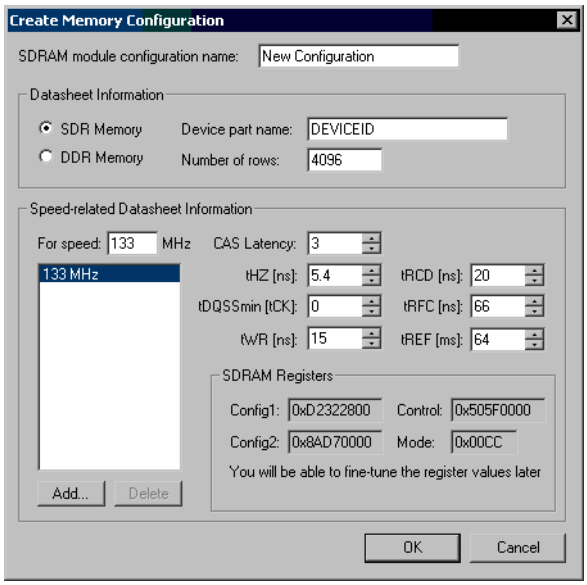

Figure 43. SDRAM Memory Configuration

The memory configuration created this way can be fine-tuned in the GCT SDRAM page and later written to the database file. There is no way to rename an existing memory configuration or to add a new clock speed entry to the list of supported speeds once the configuration is created. The only way to manage the database is by editing the database `sdram.ini` file.

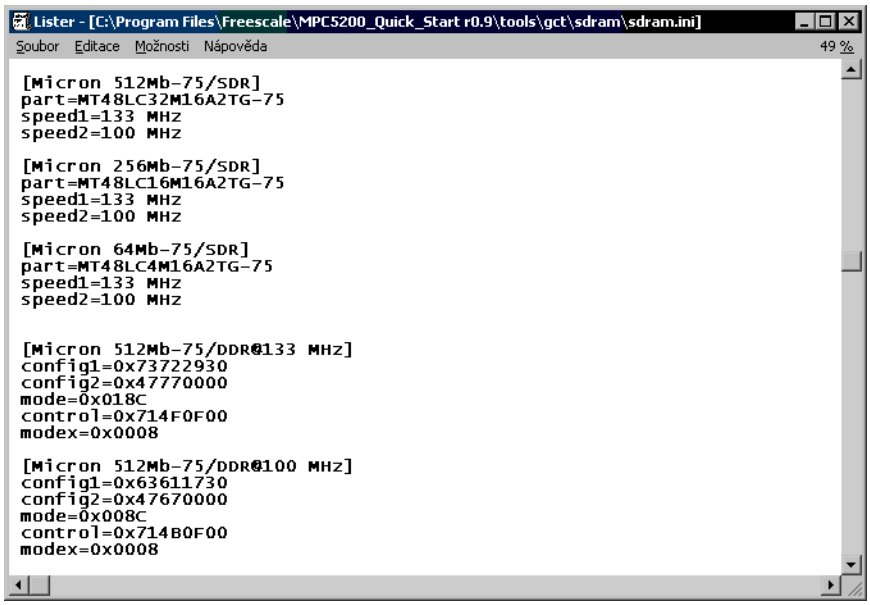

Figure 44. Example of sdram.ini Memory Definition File

6.4.20 XLB Bus Arbiter (XLARB)

The XL Bus Arbiter module is completely supported by the GCT.

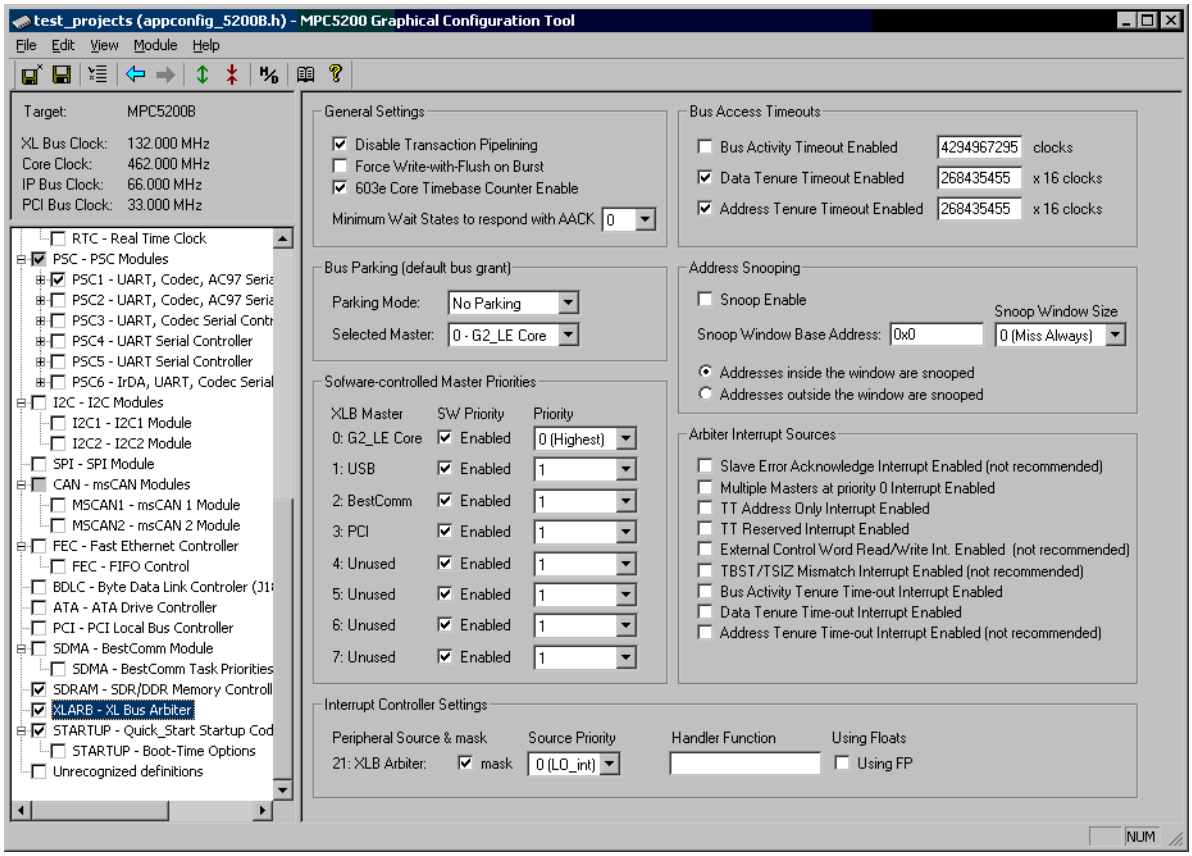

Figure 45. XLB Bus Arbiter Control Page (XLARB)

6.4.21 Startup Code Control Page (STARTUP)

As it was already described in Section 5.3, "Startup Code," the MPC5200_Quick_Start startup code invokes the `__pre_main()` function right before the user's `main()` function is called. The default implementation of the `__pre_main()` function in the `appconfig.c` file calls the initialization functions of all MPC5200 modules selected on the STARTUP page of the GCT. The GCT saves the information about which modules are to be automatically initialized in several special-purpose macro values in the `appconfig.h` file.

The `__pre_main()` code does not test whether there is a valid `appconfig.h` configuration of the modules to be initialized. For the selected modules, the `__pre_main()` code simply invokes the `ioctl()` initialization call. Thanks to a conditional compilation, the initialization functions are empty for modules not configured in the GCT.

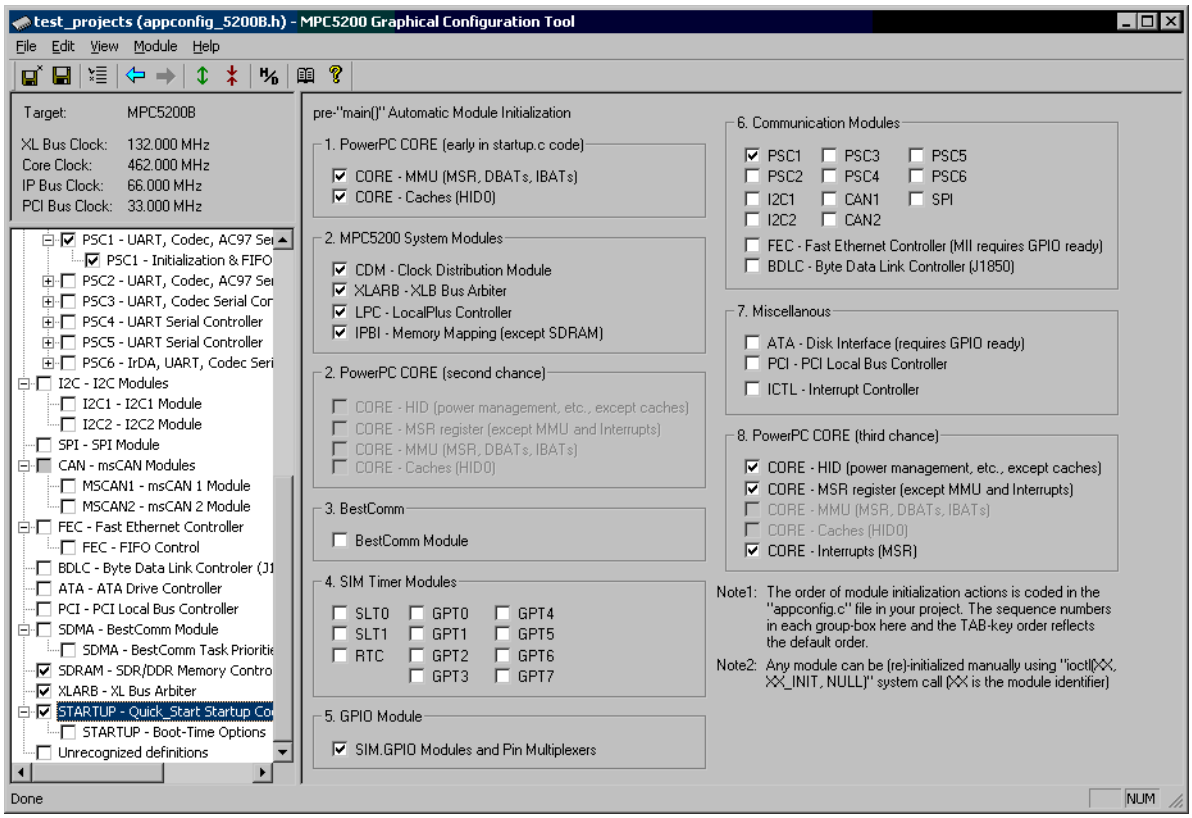

Figure 46. Startup Code Control Page (STARTUP)

The order in which the MPC5200 modules are initialized is encoded in the `__pre_main()` function in the `appconfig.c` file. By default, the order of initialization calls is the same as the order of check box buttons on the STARTUP control page. Except a simple prerequisite that the GPIO module and port multiplexer should be initialized before the FEC and ATA modules, the user can modify the `__pre_main()` code and set the order of module initialization calls according to his specific needs.

The rest of MPC5200/B modules not initialized automatically in the `__pre_main()` can still be initialized manually any time in the `main()` function or other part of a user code. The initialization calls are described in the Section 7, "Module Initialization Code."

6.5 Side-Bar Views

6.5.1 Register Values View

Any time when using the GCT, a Register View toolbar-like window can be shown to display the immediate register values as they are to be written to the `appconfig.h` file. For each module, all the registers bound to the graphical controls on the page are displayed. When a module configuration is modified, all affected registers are red-highlighted in the Register View (see Figure 47 below).

There is also a possibility to modify the register values directly in the Register View window (press Enter to accept a new value), causing the graphical controls to be redrawn accordingly; however, there are often other run-time bits in the control registers which are not supposed to be set during an initialization. Modifying the register values without paying high attention to each individual bit or bit-field of the register may cause the module settings to be invalid, even if the configuration looks good in the GCT.

The Register View bar can be activated or deactivated by a menu View / Register Summary.

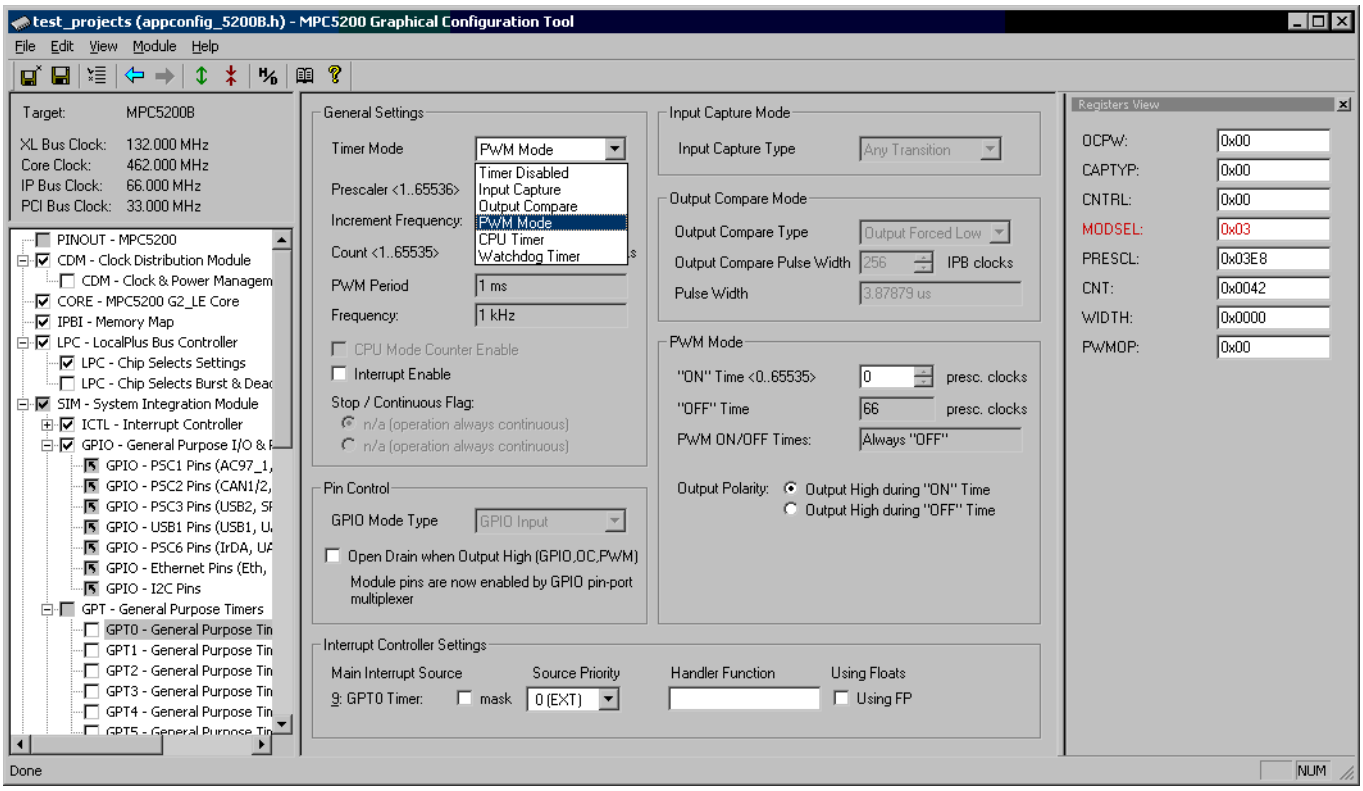

Figure 47. Register Summary View

6.5.2 Warnings View

Similarly, as the Registers View described above, the Warnings View bar can be shown or hidden any time when working with the GCT. Warnings View shows a list of warnings collected from across all the control pages in a GCT project. By default, the Warnings View displays only the warnings from modules enabled

for `appconfig.h` output (those with a checkmark sign in the project tree). All other warnings can be displayed if required.

A double-click on a warning item in the list activates the control page where the potential conflict exists, and a balloon-like hint is shown as a notification (see Figure 48 below).

The Warnings View bar can be activated or deactivated by a menu View / Warnings Summary.

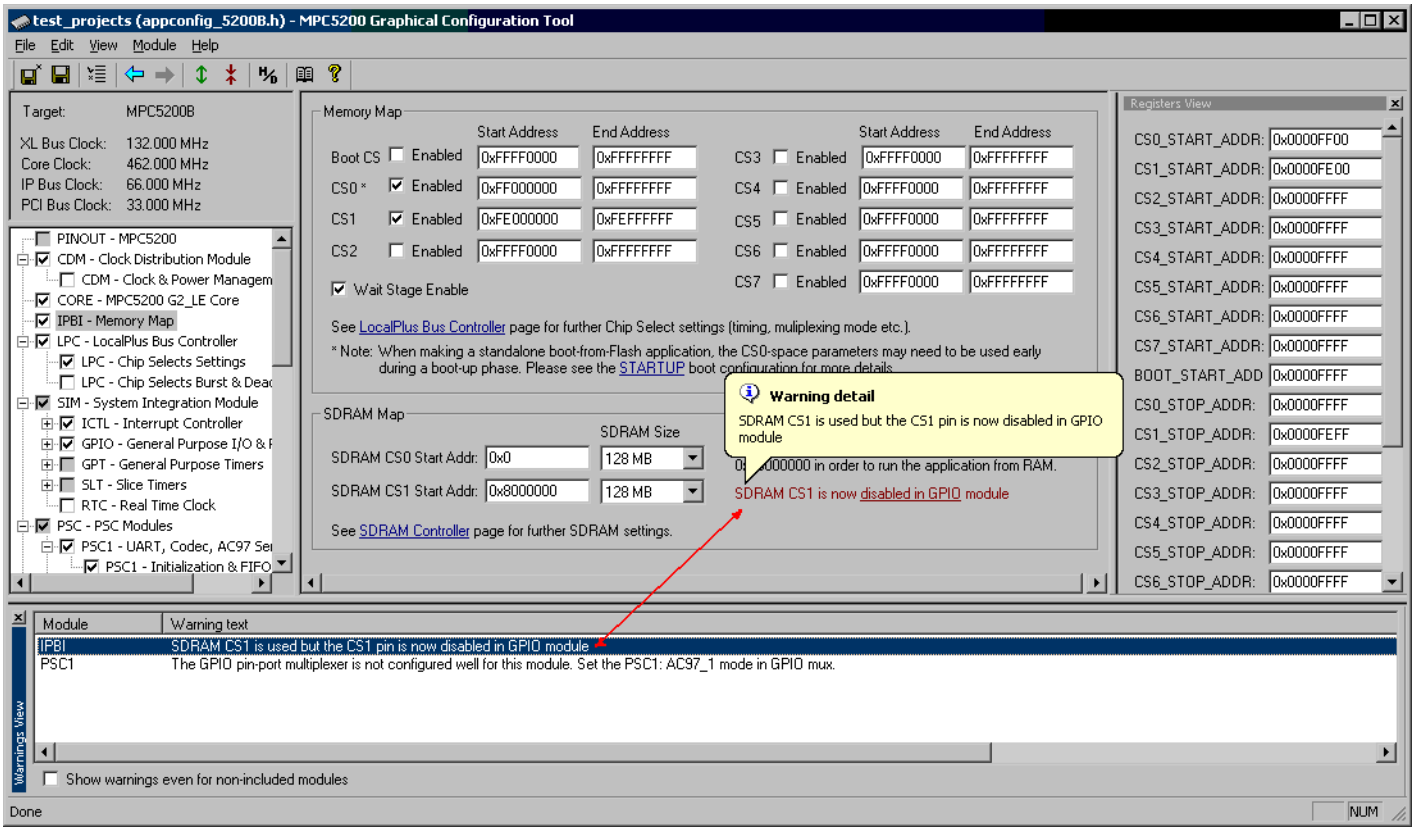

Figure 48. Warnings View

7 Module Initialization Code

The use of MPC5200_Quick_Start tool and the GCT brings a standard way of peripheral module initialization. Similarly, as with Quick Start tools for other Freescale microprocessor platforms (MPC500, MPC5500, 56F800/E) there is a special system call used to access the device's peripheral modules. A general format of the Quick Start system call is:

```
ioctl (MODULE, MODULE_COMMAND, parameter);
```

Where the `MODULE` is an unique identifier of the peripheral module (such as PSC1, SPI, CAN, etc.) and the `MODULE_COMMAND` together with a `parameter` specify an action to be performed on a module (such as `PSC_INIT`, `CAN_TRANSMIT`, `SPI_SET_BIT_RATE`, etc.).

Unlike the other Quick Start platforms, the MPC5200_Quick_Start does not implement any commands except the ones used for the module initialization (`PSC_INIT`, `CAN_INIT`, etc.). One exception is the CAN

module, for which there is a complete low-level driver implemented using the `ioctl()` system call as well as the sample application demonstrating its use. The CAN low-level driver is compatible with other MSCAN drivers of other Quick Start platforms (for example, 56F800).

The list of initialization commands to be used with the `ioctl()` system call can be found in Table 3 below. No initialization command requires the `parameter` value, so the `NULL` value can be used.

Table 3. Module Initialization Commands

Module Identifier	Initialization Command	Description
ATA	ATA_INIT	Initializes the ATA Controller (GPIO_INIT must be called before this command)
CORE	CORE_INIT_MSR	Initializes the PowerPC MSR register content, except the MMU and Interrupt-related bits
CORE	CORE_INIT_INT	Initializes the exceptions-related bits in the MSR core register
CORE	CORE_INIT_MMU	Initializes the Memory Management Unit registers (such as MMU-related bits in the MSR and HID2 core registers, IBATx and DBATx core registers).
CORE	CORE_INIT_HID	Initializes the bits of HID0 and HID2 core registers not related to the CACHE or MMU operation
CORE	CORE_INIT_CACHE	Initializes the CACHE-related bits in the HID0 core register
CDM	CDM_INIT	Initializes the Clock Distribution Module
FEC	FEC_INIT	Initializes the Fast Ethernet Controller module
GPIO	GPIO_INIT	Initializes the SIM.GPIO module
GPT0,..., GPT7	GPT_INIT	Initializes the General Purpose Timer module specified by the module identifier in the ioctl() call
I2C1, I2C2	I2C_INIT	Initializes the I2C Controller specified by the module identifier in the ioctl() call
ICTL	ICTL_INIT	Initializes the Interrupt Controller module and the Quick Start Interrupt Dispatcher infrastructure
IPBI	IPBI_INIT	Initializes the Memory Map module
LPC	LPC_INIT	Initializes the LocalPlus Bus module
CAN1, CAN2	CAN_INIT	Initializes the MSCAN module specified by the module identifier in the ioctl() call
PCI	PCI_INIT	Initializes the PCI Local Bus Controller
PSC1,..., PSC6	PSC_INIT	Initializes the Programmable Serial Controller module specified by the module identifier in the ioctl() call
RTC	RTC_INIT	Initializes the Real Time Clock module
SDMA	SDMA_INIT	Initializes the BestComm module and optionally also loads the DMA microcode image using a BSP calls
SDRAM	SDRAM_INIT	In the most cases the SDRAM is initialized automatically during startup. SDRAM_INIT is rarely used.

Table 3. Module Initialization Commands

Module Identifier	Initialization Command	Description
SLT0, SLT1	SLT_INIT	Initializes the Slice Timer module specified by the module identifier in the ioctl() call
SPI	SPI_INIT	Initializes the Serial Peripheral Interface module
USB	USB_INIT	*Not implemented.* There is no support for the USB module in current version of MPC5200_Quick_Start
XLARB	XLARB_INIT	Initializes the XL Bus Arbiter module

8 Sample Applications

To demonstrate use of the GCT and a basic access to the peripheral modules, several sample applications are included in the MPC5200_Quick_Start installation. Each application focuses on a single peripheral module of the MPC5200. The `appconfig.h` file contains valid configuration values for the module being demonstrated and also for the other modules needed to run the application (CDM, SDRAM and PSC1 for a console). In most of the cases the STARTUP configuration is set up for an automatic `__pre_main()` initialization of all modules used in the application.

All sample applications are located in the `sample_applications` sub-directory of the MPC5200_Quick_Start installation folder. Applications are organized in folders by compiler and board for which they are tested.

Except the system files, which match exactly the ones in the Project Stationery (see Section 4.1, "Project Stationery and Templates"), there is always only a single source file in each sample application—the `main.c` file. This file contains detailed comment block describing the application functionality plus all the application source code. The table below describes briefly the sample applications included with the current version of the MPC5200_Quick_Start.

Table 4. Sample Applications

Sample Application	Description
ata_demo	Demonstrates an automatic detection of ATA hard drive(s) timing modes (PIO, MDMA, UDMA) performed in the initialization code.
fec_demo	Demonstrates a use of the Fast Ethernet Controller module. Except FEC module initialization, this application also uses two standard DMA tasks to demonstrate basic receive and transmit functions.
gpt_slt_demo	Using a GPT and SLT timer interrupt sources, this application demonstrates a use of the Interrupt Dispatcher and interrupt service routines.
mscan_demo	Demonstrates a use of MSCAN module and MPC5200_Quick_Start low-level CAN driver. A CAN transmit and receive operations can be demonstrated using either a CAN link to PC or using two Lite5200 boards connected together.
pci_demo1	Demonstrates a use of the PCI Configuration Space read operations. Displays an information about devices in the Lite5200 PCI slots (including the MPC5200 PCI bridge itself).

Table 4. Sample Applications

Sample Application	Description
pwm_demo	Demonstrates a use of the PWM mode of the GPT timer module to control the LED light intensity.
rtc_demo	Demonstrates a use of interrupts on the RTC module.
spi_demo1	Requires an externally connected MAX5152 DA converter to demonstrate a use of the SPI mode of the PSC2 module. The voltage on the four DAC channels can be set using the console commands.
spi_demo2	Requires an externally connected ST95020 EEPROM device to demonstrate a use of the interrupt-driven SPI module operations.
uart_demo	Demonstrates a use of UART mode of two PSC modules.

9 MPC5200 BSP

MPC5200_Quick_Start is built on top of the software support package called Board Support Package (BSP), distributed with the Lite5200 systems. The latest version of the BSP package is included in the Quick Start distribution, so there is no need to retrieve and install it separately. The full BSP source code is also included in all MPC5200_Quick_Start project templates, so BSP functions are immediately ready to be used in user applications.

The following table summarizes the features of the BSP and shows the items reused by MPC5200_Quick_Start applications. All BSP code is installed in the `support\bsp` sub-directory of the MPC5200_Quick_Start `src` folder.

Table 5. BSP Reuse in MPC5200_Quick_Start

BSP Item	Quick Start Reuse	Description
MPC5200 and MPC5200B Header Files	Fully reused	Quick Start uses its own structure types for mapping of MPC5200 peripheral registers. However in most of the cases the Quick Start types are just a typedefs equivalent to the original BSP types. The Quick Start header files take care about including the original BSP header files so the re-use is fully transparent to the user.
PowerPC basic types	Re-implemented	Quick Start uses the same basic types as in the BSP (uint32, uint16,...). The *ppctypes.h* header file is duplicated in the Quick Start *include* folder.
PSC_UART and console code	Partially reused	The PSC UART interface to the system `stdio` calls (printf, puts, gets,...) is reused in Quick Start applications. The BSP console initialization code is disabled so the GCT configuration of the PSC1 applies for the console.
“BestComm code and DMA RTOS images	Fully reused	The BSP contains the official BestComm support from Freescale so it is fully reused in Quick Start. The BestComm DMA images are integrated in the Quick Start project templates DMA_ImageRtos1, DMA_ImageRtos2 and DMA_ImageRtos3.

Table 5. BSP Reuse in MPC5200_Quick_Start (continued)

BSP Item	Quick Start Reuse	Description
“Exceptions BSP code	Not used / Not available	The Quick Start implements its own exception handling mechanisms in Interrupt Dispatcher.
“Time, Sleep, RTClock, Core and Frequency BSP code	Not used / Available	The code is not used by the Quick Start. However, the applicable BSP source files are included in Quick Start projects to maintain compatibility with older code.

10 Conclusion

The MPC5200_Quick_Start development environment can help users become familiar with the powerful and rather complex devices of the MPC5200 family. With support for creating non-operating system applications, the user is capable of writing a fully functional code with a complete low-level access to all peripheral modules of the processor. Using the GCT, the vast majority of processor features can be explored quickly and more effectively than by going through the Users Guide. By looking at how the control registers changes in the GCT, the user can also better understand the meaning of individual control bits and bit fields as they are described in the Users Guide.

The integration with, and the support of, the CodeWarrior IDE significantly reduces the code-debug-deploy loop as compared with other environments. On the other hand, it is still a subject of future MPC5200_Quick_Start development to widen the set of supported tools and platforms.

11 Revision History

Table 6 provides revision history details for this document, beginning with Revision 3.

Table 6. Revision History

Rev. No.	Substantive Change(s)
3	Added PowerPC trademarking information in first paragraph on page 1 and the back page, removed references to Metrowerks (now Freescale), and added a revision history table to record future changes. Minor editorial changes also made.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2757
Rev. 3, 06/2006

freescale™
semiconductor