M5251C3 Evaluation Board

Users Guide

Document Number: M5251C3UG
Rev. 0
05/2006

Contents

Chapter 1
M5251C3 Evaluation Board

The 5251 is a versatile single-board computer based on the MCF5251 ColdFire® Processor. It may be used as a powerful microprocessor based controller in a variety of applications. It serves as a complete microcomputer system for reference design, development/evaluation, training, and educational use. The user need only connect an RS-232 compatible terminal (or a personal computer with terminal emulation software) and a power supply to have a fully functional system.

CAUTION

This board generates, uses, and can radiate radio frequency energy and, if not installed properly, may cause interference to radio communications. Operation of this product in a residential area is likely to cause interference, in which case, the user, at his/her own expense, will be required to correct the interference.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

1.1 General Hardware Description

The M5251C3 board provides SDRAM, Flash ROM, and RS-232 in addition to the built-in I/O functions of the MCF5251 device for programming and evaluating the attributes of the microprocessor. In addition, there is an IDE and ATA interface for connection to things like an external HDD. There is also an SD card interface, CAN interface, and both analog and digital audio I/O connections. The board is driven by the MCF5251 device, which is a member of the ColdFire® family of processors. It is a 32-bit processor with a 24-bit address bus and 16 lines of data. The processor has eight 32-bit data registers, eight 32-bit address registers, a 32-bit program counter, and a 16-bit status register.

The MCF5251 processor has a System Integration Module referred to as the SIM. This module incorporates many of the functions needed for system design. These include programmable chip-select logic, system protection logic, general purpose I/O and interrupt controller logic. The chip-select logic can select up to five memory banks and peripherals in addition to one bank of DRAM. The chip-select logic also allows the insertion of a programmable number of wait states to allow slower memory or memory-mapped peripherals to be used. (Refer to M5251C3 Evaluation Board by Freescale for detailed information about the chip selects.) One of the chip selects ($\overline{\text{CS0}}$) is used to access the on-board Flash ROM; the other chip selects are user-programmable. The DRAM controller is used to control one SDRAM device providing 8 MB of SDRAM memory configured as 4 MB x 16 words. All other functions of the SIM are available to the user.

Figure 1-1 shows the M5251C3 block diagram.

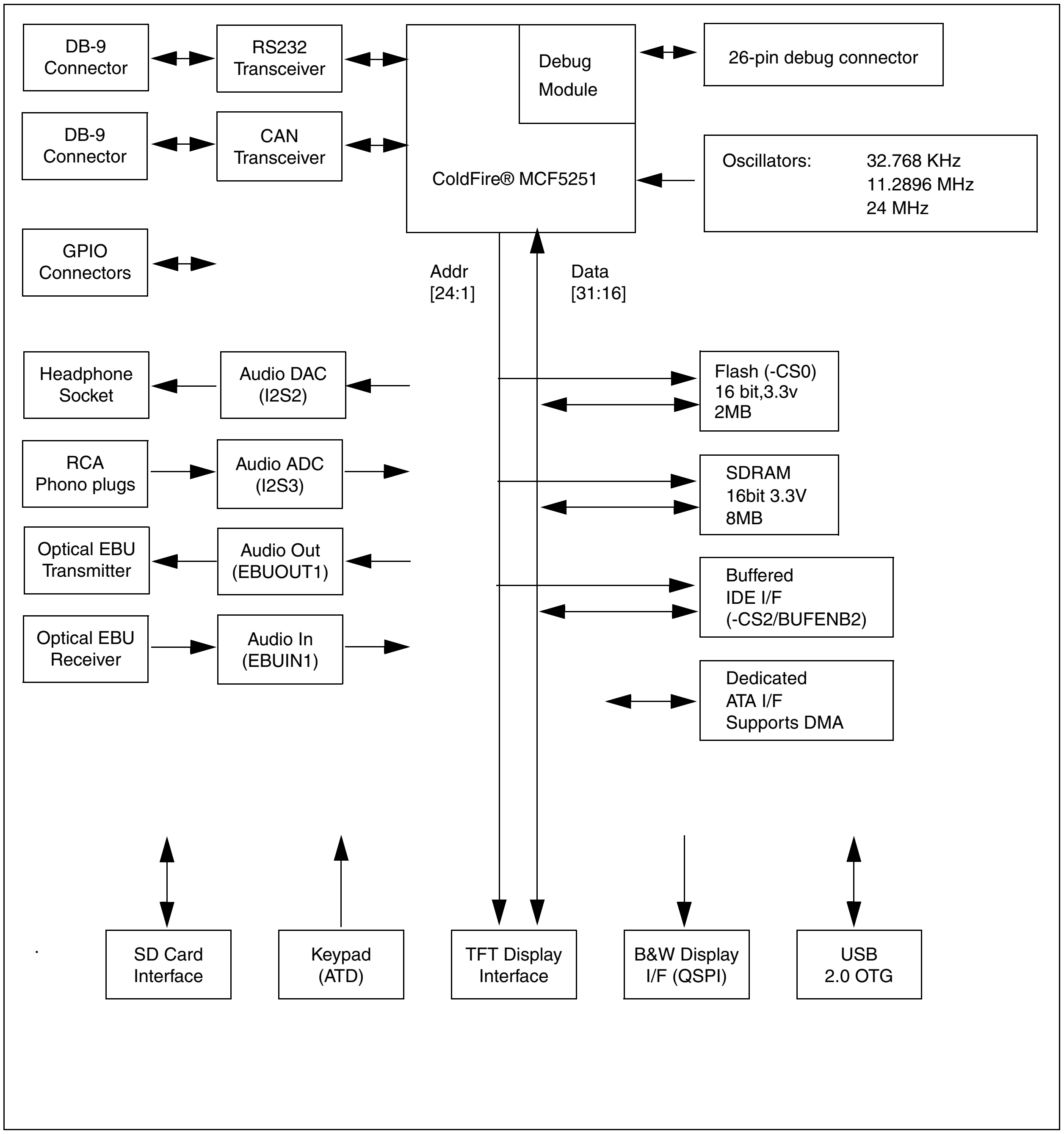


Figure 1-1 M5251C3 Block Diagram

1.2 System Memory

One on-board Flash ROM (U11) is used in the system. The Am29LV160DB-XX device contains 16 Mbits of non-volatile storage (1 Mbyte x 16), giving a total of 2 Mbytes of Flash memory. The lower 256 Kbytes are used to store the M5251C3 dBUG debugger/monitor firmware that is pre-programmed into the Flash during factory testing.

The MCF5251 processor has 128 Kbytes of internal SRAM organized as 2 banks of 64 Kbytes. The SRAM can be used for either data or instruction space.

There is one SDRAM (U12) device on the PCB. The system ships with 1 x 4 Mbytes x 16 of SDRAM totalling 8 Mbytes of volatile memory.

The internal cache of the MCF5251 is non-blocking. The instruction cache is 8 Kbytes with a 16-byte line size. The ROM Monitor currently does not utilize the cache, but programs downloaded with the ROM Monitor can initialize and use the cache.

1.3 Serial Communication Channels

The MCF5251 processor has 3 built-in UARTs with independent baud rate generators. The signals of all channels can be passed through the external transceiver to make the channels RS-232 compatible (P4). An RS-232 serial cable with DB9 connectors is included with the board. UART0 channel is the “TERMINAL” channel used by dBUG for communication with an external terminal/PC. The “TERMINAL” baud rate defaults to 19200.

1.4 Parallel I/O Ports

The MCF5251 offers up to 60 lines of general-purpose I/O, of which 6 are dedicated inputs and 3 are dedicated outputs. Seven of the GPIO lines are also available as edge-sensitive interrupt inputs. In addition, there is one dedicated input for wake-up from low-power mode.

1.5 System Configuration

The M5251C3 board requires the following items for minimum system configuration:

- The M5251C3 board (provided)
- Power supply, +7 V to 14 V DC with minimum of 1.0 amp
- RS232C-compatible terminal or a PC with terminal emulation software
- RS232 communication cable (provided)

Refer to Section 2.2.2, “System Initialization” for initial system setup.

Figure 1-2 displays the minimum system configuration.

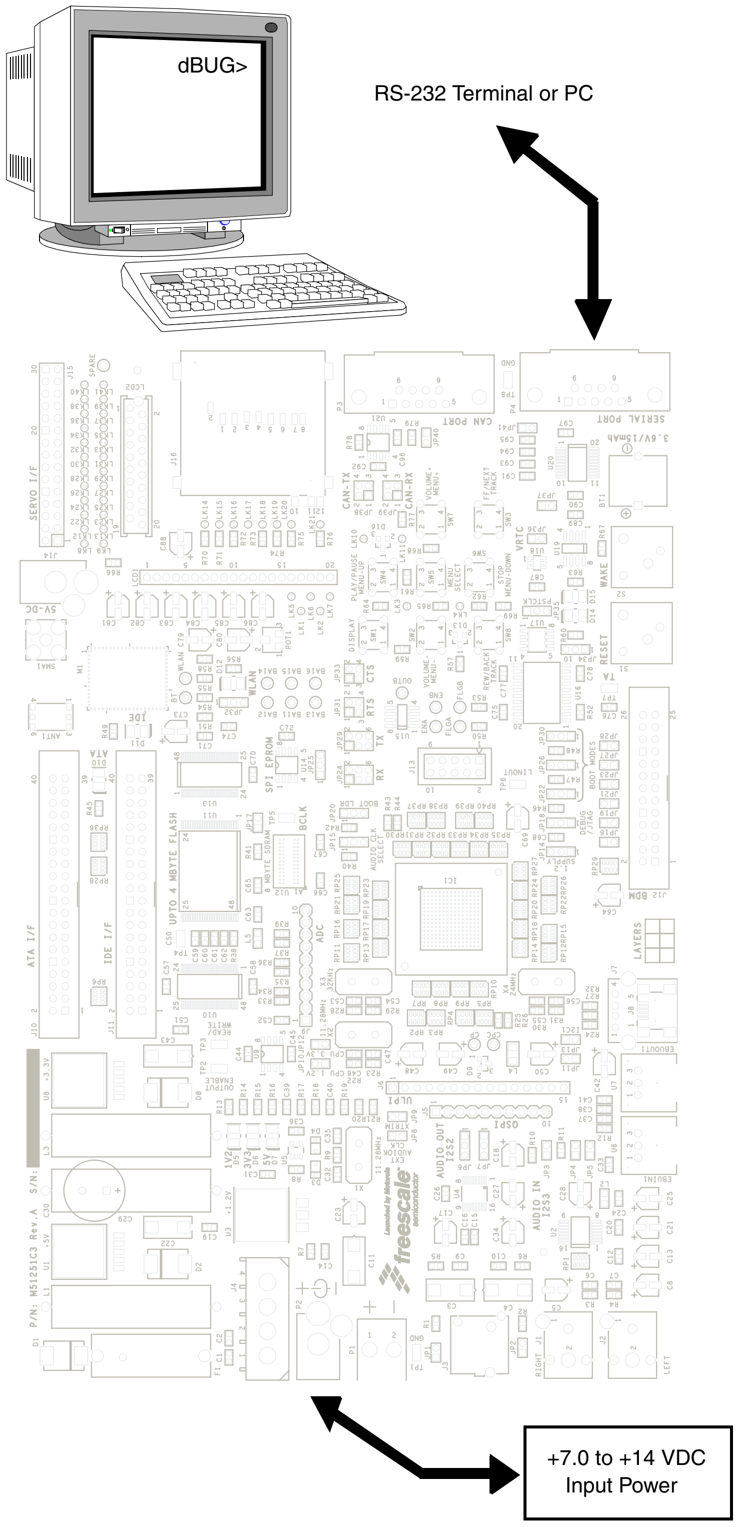


Figure 1-2 Minimum System Configuration

1.6 Installation and Setup

This section discusses all the steps needed to prepare the board for operation. Read all the sections carefully before using the board. When you are preparing the board for the first time, be sure to check that all jumpers are in the default locations.

1.6.1 Unpacking

Unpack the computer board from its shipping box. Save the box for storing or reshipping. Refer to the following list and verify that all the items are present. You should have received:

- M5251C3 Single Board Computer (CE certified)
- Display: 2.2” TFT with backlight, FPC connector, and PCBA
- Sceptre 9.0 Volt, 2.7 A power supply with WS-047 and WS-048 adapters
- Cable: DB9 M/F: RS-232 Parallel Port Interface Extension Cable
- Wind River USB Probe with ColdFire adapter
- Wind River Workbench 2.4 OCD Edition
- Wind River literature CD
- *EVB Quickstart Guide* (hardcopy)
- Warranty card - 920-75133
- Technical Information Center Worldwide Contact List
- Freescale Documentation (www.freescale.com/digitalaudio)
  - M5251C3 Evaluation Board (this document)
  - *MCF5251UM ColdFire Reference Manual*
  - ColdFire® Programmers Reference Manual
  - CF Flasher Tool
  - TRIO HDD Demonstration Code
  - dBUG Firmware
  - ColdFire Init

CAUTION

Avoid touching the MOS devices. Static discharge can and will damage these devices.

Once you have verified that all the items are present, remove the board from its protective jacket and anti-static bag. Check the board for any visible damage. Ensure that there are no broken, damaged, or missing parts. If you have not received all the items listed above or they are damaged, contact Rapid Design immediately. For contact details, see the front of this manual.

1.6.2 Preparing the Board for Use

The board, as shipped, is ready to be connected to a terminal and power supply without any need for modification. Figure 1-4 shows the position of the jumpers and connectors.

1.6.3 Providing Power to the Board

The board accepts three means of power supply connection—P1, P2, or J4. Connector P1 is a 2.1 mm power jack, P2 is a lever-actuated connector, and J4 is a PC disk drive-type power connector. The board accepts +7 V to +14 V DC at 1.0 amp via either of the connectors. Table 1-2 lists power supply connections on P2.

Table 1-1 Power Supply Connections on P2

Contact Number	Voltage
1	+7 V to +14 V DC
2	Ground

1.6.4 Selecting Terminal Baud Rate

The serial channel UART0 of the MCF5251 is used for serial communication and has a built in timer. This timer is used by the dBUG ROM monitor to generate the baud rate used to communicate with a serial terminal. A number of baud rates can be programmed. On power-up or manual RESET, the dBUG ROM monitor firmware configures the channel for 19200 baud. Once the dBUG ROM monitor is running, a SET command may be issued to select any baud rate supported by the ROM monitor. See Section 2.1, "What Is dBUG?" for the discussion of this command.

1.6.5 Terminal Character Format

The character format of the communication channel is fixed at power-up or RESET. The default character format is 8 bits per character, no parity and one stop bit with no flow control. It is necessary to ensure that the terminal or PC is set to this format.

1.6.6 Connecting the Terminal

The board is now ready to be connected to a PC/terminal. Use the RS232 serial cable to connect the PC/terminal to the M5251C3 PCB. The cable has a 9-pin female D-sub terminal connector at one end and a 9-pin male D-sub connector at the other end. Connect the 9-pin male connector to connector P4 on the M5251C3 board. Connect the 9-pin female connector to one of the available serial communication channels normally referred to as COM1 (COM2, etc.) on the PC running terminal emulation software. The connector on the PC/terminal may be either male 25-pin or 9-pin. It may be necessary to obtain a 25pin-to-9pin adapter to make this connection. If an adapter is required, refer to Figure 1-3, which shows the pin assignment for the 9-pin connector on the board.

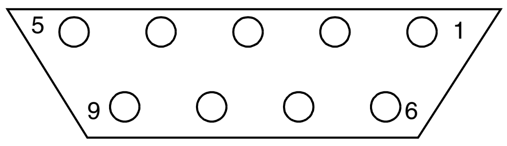


Figure 1-3 Pin Assignment for Female (Terminal) Connector

1.6.7 Using a Personal Computer as a Terminal

A personal computer may be used as a terminal provided a terminal emulation software package is available. Examples of this software are PROCOMM, KERMIT, QMODEM, Windows 95/98/2000/XP Hyper Terminal or similar packages. The board should then be connected as described in Section 1.6.6, "Connecting the Terminal."

Once the connection to the PC is made, power may be applied to the PC and the terminal emulation software can be run. In terminal mode, it is necessary to select the baud rate and character format for the channel. Most terminal emulation software packages provide a command known as "Alt-p" (press the p key while pressing the Alt key) to choose the baud rate and character format. The character format should be 8 bits, no parity, one stop bit. (See Section 1.6.5, "Terminal Character Format.") The baud rate should be set to 19200. Power can now be applied to the board.

Pin assignments are as follows:

1. Data Carrier Detect—Output (shorted to pins 4 and 6)
2. Receive Data—Output from board (Receive refers to terminal side.)
3. Transmit Data—Input to board (Transmit refers to terminal side.)
4. Data Terminal Ready—Input (shorted to pin 1 and 6)
5. Signal Ground
6. Data Set Ready—Output (shorted to pins 1 and 4)
7. Request to Send—Input
8. Clear to Send—Output
9. Not connected

Figure 1-4 on the next page shows the default jumper locations for the board.

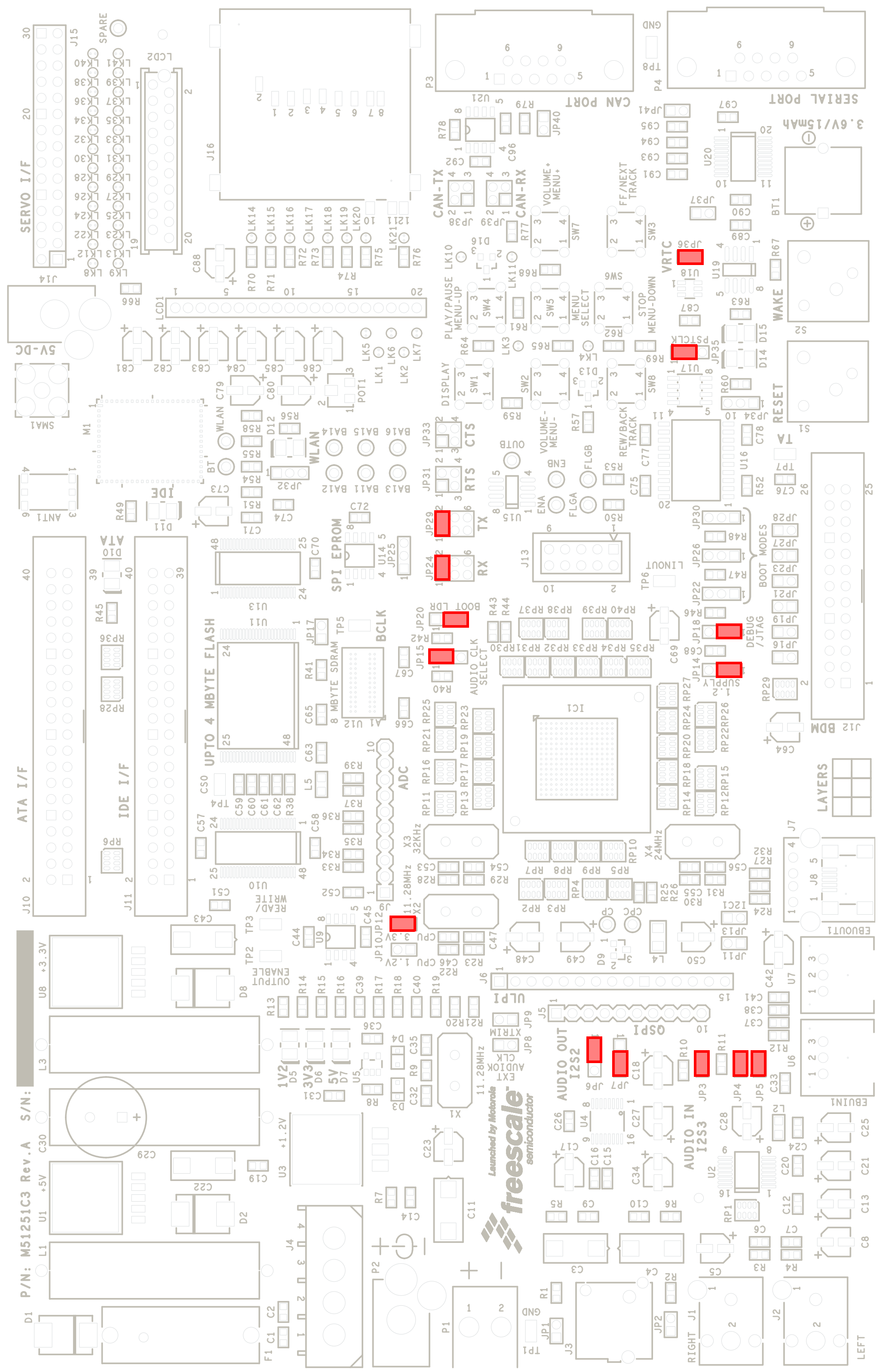


Figure 1-4 Default Jumper Locations

1.7 System Power-up and Initial Operation

When all of the cables are connected to the board, power may be applied. The dBUG ROM Monitor initializes the board and then displays a power-up message on the terminal, which includes the amount of memory present on the board.

```
Hard Reset
DRAM Size: 8M

Copyright 1995-2005 Freescale Semiconductor, Inc.  All Rights Reserved.
ColdFire MCF5251 EVS Firmware v3b.00.1a (Build XXX on XXX  XX 20XX
xx:xx:xx)

Enter 'help' for help.

dBUG>
```

The board is now ready for operation under the control of the debugger as described in Section 2.2, "Operational Procedure." If you do not get the above response, perform the following checks:

1. Make sure that the power supply is properly configured for polarity, voltage level and current capability (~1A) and is connected to the board.
2. Check that the terminal and board are set for the same character format and baud.
3. Press the RESET button to insure that the board has been initialized properly.

If you still are not receiving the proper response, your board may have been damaged. Contact Rapid PCB for further instructions. For contact details, see the front of this manual.

1.8 M5251C3 Jumper Setup

Jumper settings are as follows:

NOTE

"*" is used to indicate that default setting.
"**" is used to indicate mandatory setting for proper operation.

Table 1-2 Jumper Settings

Jumper Setting		Function
JP1,JP2	Not fitted	Audio DAC AK4366VT U4 left and right audio channel load
JP3,JP4,JP5	* Fitted	Audio ADC AK5353VT U2 I2S3 isolation (SCLK,LRCK,SDATAI)
JP6	* 1-2	Audio DAC AK4366VT U4 audio format I2S
	2-3	Audio DAC AK4366VT U4 audio format 24-bit MSB justified
JP7	1-2	Audio DAC AK4366VT U4 de-emphasis ON
	* 2-3	Audio DAC AK4366VT U4 de-emphasis OFF
JP8	Not fitted	Enable external audio clock source
JP9	Not fitted	Enable XTRIM hardware feature

Table 1-2 Jumper Settings (Continued)

Jumper Setting		Function
JP10	Not fitted	External 1.2V (core) supply current measurement
JP11,JP13	Not fitted	I2S0 5V pull-up (SDA0 and SCL0)
JP12	** Fitted	External 3.3V (pad) supply current measurement
JP14	** 1-2	1.2V (core) supply from internal regulator
	2-3	1.2V (core) supply from external regulator
JP15	* 1-2	Audio clock source taken from CRIN
	2-3	Audio clock source taken from external pin (LRCK3)
JP16,JP19	Not fitted	I2S1 5V pull-up (SDA1 and SCL1)
JP17	Not fitted	SPI EEPROM M25P40 U14 CS enable (QSPICS0)
JP18	** 1-2	Select BDM debug mode
	2-3	Select JTAG debug mode
JP20	1-2	Boot mode select, from internal ROM
	* 2-3	Boot mode select, from external memory (CS0)
JP21,JP23,JP27,JP28	Not fitted	DDATA signal isolation from BDM interface
JP22,JP26,JP30	Not fitted	Internal boot ROM mode select
JP24	* 1-2 (UART0)	RS232 transceiver RX select
JP25	1-2	SPI EEPROM M25P40 U14 write protected (WP)
	2-3	SPI EEPROM M25P40 U14 write enabled
JP29	* 1-2 (UART0)	RS232 transceiver TX select
JP31	Not fitted	RS232 transceiver RTS select
JP32	1-2	Wireless module M1 WAIT signal connected to IDEIORDY
	2-3	Wireless module M1 WAIT signal connected to TA
JP33	Not fitted	RS232 transceiver CTS select
JP34	1-2	Push button S2 connected to interrupt capable GPIO1
	2-3	Push button S2 connected to WAKEUP pin
JP36	* Fitted	Real Time Clock (RTC) supply isolation / current measurement
JP37	Not fitted	RS232 transceiver INVALID signal connected to GPIO34
JP38	Not fitted	CAN transceiver TX select
JP39	Not fitted	CAN transceiver RX select
JP40	Not fitted	Connect CAN bus termination resistor (120R)
JP41	Not fitted	RS232 transceiver READY signal connected to GPIO8

1.9 Using the BDM Port

The MCF5251 microprocessor has a built in debug module referred to as BDM (background debug module). In order to use the BDM, simply connect the 26-pin debug connector on the board (J12) to the P&E BDM wiggler cable provided in the kit. No special setting is needed. Refer to the *ColdFire® User's Guide* BDM section for additional instructions.

NOTE

BDM functionality and use are supported via third party developer software tools. Details may be found on CD-ROM included in this kit.

Chapter 2
Using the Monitor/Debug Firmware

The M5251C3 single board computer has a resident firmware package that provides a self-contained programming and operating environment. The firmware, named dBUG, provides the user with monitor/debug interface, in-line assembler and disassembly, program download, register and memory manipulation, and I/O control functions. This chapter is a how-to-use description of the dBUG package, including the user interface and command structure.

NOTE

The current M5251C3 dBUG version does not support the network commands that require an ethernet connection, these commands are documented to support future firmware releases.

2.1 What Is dBUG?

dBUG is a traditional ROM monitor/debugger that offers a comfortable and intuitive command line interface that can be used to download and execute code. It contains all the primary features needed in a debugger to create a useful debugging environment.

dBUG is a resident firmware package for the ColdFire® family single board computers. The firmware (stored in one 1Mx16 Flash ROM device) provides a self-contained programming and operating environment. dBUG interacts with the user through pre-defined commands that are entered via the terminal. These commands are defined in Section 2.4, “Commands.”

The user interface to dBUG is the command line. A number of features have been implemented to achieve an easy and intuitive command line interface.

dBUG assumes that an 80 x 24 character dumb-terminal is utilized to connect to the debugger. For serial communications, dBUG requires eight data bits, no parity, and one stop bit, 8N1 with no flow control. The default baud rate is 19200 but can be changed after the power-up.

The command line prompt is “dBUG>”. Any dBUG command may be entered from this prompt. dBUG does not allow command lines to exceed 80 characters. Wherever possible, dBUG displays data in 80 columns or less. dBUG echoes each character as it is typed, eliminating the need for any “local echo” on the terminal side.

In general, dBUG is not case sensitive. Commands may be entered either in upper or lower case, depending upon the user’s equipment and preference. Only symbol names require that the exact case be used.

Most commands can be recognized by using an abbreviated name. For instance, entering “h” is the same as entering “help”. Thus, it is not necessary to type the entire command name.

The commands DI, GO, MD, STEP, and TRACE are used repeatedly when debugging. dBUG recognizes this and allows for repeated execution of these commands with minimal typing. After a command is entered, simply press <RETURN> or <ENTER> to invoke the command again. The command is executed as if no command line parameters were provided.

An additional function called the "TRAP 15 handler" allows the user program to utilize various routines within dBUG. The TRAP #15 handler is discussed in Section 2.5, "TRAP #15 Functions" on page 2-22.

The operational mode of dBUG is shown in Figure 2-1. After the system initialization, the board waits for a command-line input from the user terminal. When a proper command is entered, the operation continues in one of the two basic modes. If the command causes execution of the user program, the dBUG firmware may or may not be re-entered, at the discretion of the user's program. For the alternate case, the command will be executed under control of the dBUG firmware, and after command completion, the system returns to command entry mode.

During command execution, additional user input may be required depending on the command function.

For commands that accept an optional <width> to modify the memory access size, the valid values are:

- B 8-bit (byte) access
- W 16-bit (word) access
- L 32-bit (long) access

When no <width> option is provided, the default width is .W, 16-bit.

The core ColdFire® register set is maintained by dBUG. These are listed below:

- A0-A7
- D0-D7
- PC
- SR

All control registers on ColdFire® are not readable by the supervisor-programming model, and thus not accessible via dBUG. User code may change these registers, but caution must be exercised as changes may render dBUG inoperable.

A reference to "SP" (stack pointer) actually refers to general purpose address register seven, "A7."

2.2 Operational Procedure

System power-up and initial operation are described in detail in Section 1.7, "System Power-up and Initial Operation."

2.2.1 System Power-up

1. Be sure the power supply is connected properly prior to power-up.
2. Make sure the terminal is connected to TERMINAL (P4) connector.
3. Turn power on to the board.

Figure 2-1 shows the dBUG operational mode.

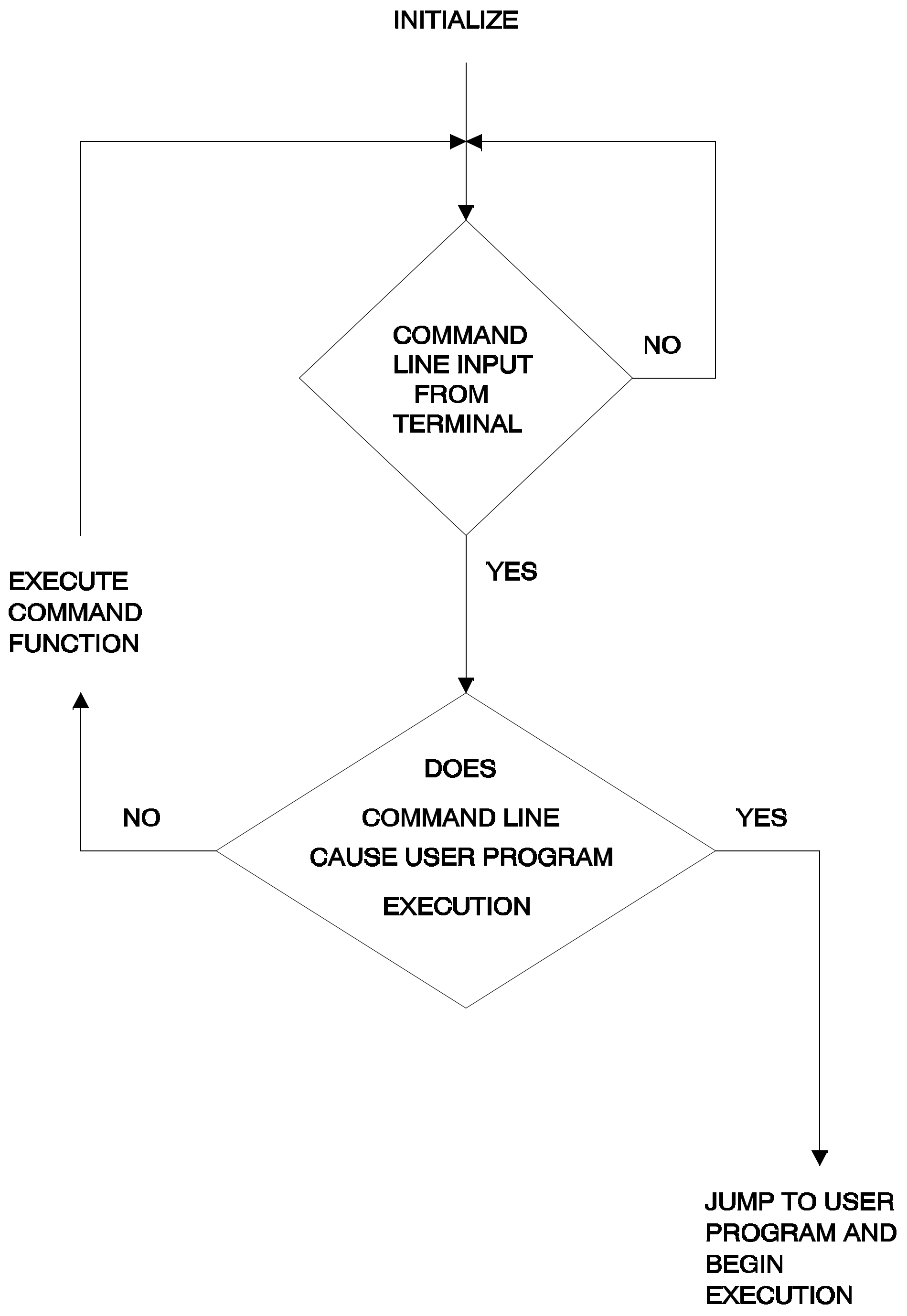

INITIALIZE
COMMAND LINE INPUT FROM TERMINAL
NO
YES
EXECUTE COMMAND FUNCTION
DOES COMMAND LINE CAUSE USER PROGRAM EXECUTION
NO
YES
JUMP TO USER PROGRAM AND BEGIN EXECUTION

Figure 2-1 Flow Diagram of dBUG Operational Mode

2.2.2 System Initialization

The act of powering up the board will initialize the system. The processor is reset and dBUG is invoked.

dBUG performs the following configurations of internal resources during the initialization. The instruction cache is invalidated and disabled. The Vector Base Register (VBR) points to the Flash. However, a copy of the exception table is made at address $00000000 in SDRAM. To take over an exception vector, the user places the address of the exception handler in the appropriate vector in the vector table located at 0x00000000, and then points the VBR to 0x00000000.

The Software Watchdog Timer is disabled and internal timers are placed in a stop condition. Interrupt controller registers initialized with unique interrupt level/priority pairs. Refer to the dBUG source files on the ColdFire website (www.freescale.com/coldfire) for the complete initialization code sequence.

After initialization, the terminal displays the following:

```
Hard Reset
DRAM Size: 8M

Copyright 1995-2005 Freescale Semiconductor, Inc.  All Rights Reserved.
ColdFire MCF5251 EVS Firmware v3b.00.1a (Build XXX on XXX)

Enter 'help' for help.

dBUG>
```

If you did not get this response check the setup, refer to Section 1.7, "System Power-up and Initial Operation."

Other means can be used to re-initialize the M5251C3 Computer Board firmware. These means are discussed in Section 2.2.2.1, "Hard RESET Button," Section 2.2.2.2, "ABORT Button," and Section 2.2.2.3, "Software Reset Command."

2.2.2.1 Hard RESET Button

Hard RESET (S1) is the red button. Pressing this button causes all processes to terminate, resets the MCF5251 processor and board logic, and restarts the dBUG firmware. Pressing the RESET button would be the appropriate action if all else fails.

2.2.2.2 ABORT Button

ABORT (S2) is the button located next to the RESET button. The abort function causes an interrupt of the present processing (a level 7 interrupt on MCF5251) and gives control to the dBUG firmware. This action differs from RESET in that no processor register or memory contents are changed, the processor and peripherals are not reset, and dBUG is not restarted. Also, in response to depressing the ABORT button, the contents of the MCF5251 core internal registers are displayed.

The abort function is most appropriate when software is being debugged. The user can interrupt the processor without destroying the present state of the system. This is accomplished by forcing a non-maskable interrupt that will call a dBUG routine that will save the current state of the registers to shadow registers in the monitor for display to the user. The user will be returned to the ROM monitor prompt after exception handling.

2.2.2.3 Software Reset Command

dBUG's command—"RESET"—causes the dBUG to restart as if a hardware reset was invoked.

2.3 Command Line Usage

The user interface to dBUG is the command line. A number of features have been implemented to achieve an easy and intuitive command line interface.

dBUG assumes that an 80 x 24 ASCII-character dumb terminal is used to connect to the debugger. For serial communications, dBUG requires eight data bits, no parity, and one stop bit (8N1). The baud rate default is 19200 bps—a speed commonly available from workstations, personal computers, and dedicated terminals.

The command line prompt is: dBUG>

Any dBUG command may be entered from this prompt. dBUG does not allow command lines to exceed 80 characters. Wherever possible, dBUG displays data in 80 columns or fewer. dBUG echoes each character as it is typed, eliminating the need for any local echo on the terminal side.

The <Backspace> and <Delete> keys are recognized as rub-out keys for correcting typographical mistakes.

Command lines may be recalled using the <Control> U, <Control> D and <Control> R key sequences. <Control> U and <Control> D cycle up and down through previous command lines. <Control> R recalls and executes the last command line.

dBUG is not case-sensitive. Commands may be entered either in uppercase or lowercase, depending upon the user's equipment and preference. Only symbol names require that the exact case be used.

Most commands can be recognized by using an abbreviated name. For instance, entering h is the same as entering help. Thus, it is not necessary to type the entire command name.

The commands DI, GO, MD, STEP, and TRACE are used repeatedly when debugging. dBUG recognizes this and allows for repeated execution of these commands with minimal typing. After a command is entered, press the <Return> or <Enter> key to invoke the command again. The command is executed as if no command line parameters were provided.

2.4 Commands

This section lists the commands that are available with all versions of dBUG. (Some board or CPU combinations may use additional commands not listed in Table 2-1.)

Table 2-1 dBUG Command Summary

Mnemonic	Syntax	Description
ASM	asm <<addr> stmt>	Assemble
BC	bc addr1 addr2 length	Block Compare
BF	bf <width> begin end data <inc>	Block Fill
BM	bm begin end dest	Block Move
BR	br addr <-r> <-c count> <-t trigger>	Breakpoint
BS	bs <width> begin end data	Block Search

Table 2-1 dBUG Command Summary (Continued)

Mnemonic	Syntax	Description
DC	dc value	Data Convert
DI	di<addr>	Disassemble
DL	dl <offset>	Download Serial
DN	dn <-c> <-e> <-i> <-s <-o offset>> <filename>	Download Network
GO	go <addr>	Execute
GT	gt addr	Execute To
HELP	help <command>	Help
IRD	ird <module.register>	Internal Register Display
IRM	irm module.register data	Internal Register Modify
LR	lr <width> addr	Loop Read
LW	lw <width> addr data	Loop Write
MD	md <width> <begin> <end>	Memory Display
MM	mm <width> <addr> <data>	Memory Modify
MMAP	mmap	Memory Map Display
RD	rd <reg>	Register Display
RM	rm reg data	Register Modify
RESET	reset	Reset
SD	sd	Stack Dump
SET	set <option value>	Set Configurations
SHOW	show <option>	Show Configurations
STEP	step	Step (Over)
SYMBOL	symbol <symb> <-a symb value> <-r symb> <-C|l|s>	Symbol Management
TRACE	trace <num>	Trace (Into)
UPDBUG	updbug	Update dBUG
UPUSER	upuser <bytes>	Update User Flash
VERSION	version	Show Version

2.4.1 ASM (Assembler)

Usage: ASM <<addr> stmt>

The ASM command is a primitive assembler. The <stmt> is assembled and the resulting code placed at <addr>. This command has an interactive and non-interactive mode of operation.

The value for address <addr> may be an absolute address specified as a hexadecimal value, or a symbol name. The value for stmt must be valid assembler mnemonics for the CPU.

For the interactive mode, the user enters the command and the optional <addr>. If the address is not specified, then the last address is used. The memory contents at the address are disassembled, and the user prompted for the new assembly. If valid, the new assembly is placed into memory, and the address incremented accordingly. If the assembly is not valid, then memory is not modified, and an error message produced. In either case, memory is disassembled and the process repeats.

The user may press the <Enter> or <Return> key to accept the current memory contents and skip to the next instruction, or a enter period to quit the interactive mode.

In the non-interactive mode, the user specifies the address and the assembly statement on the command line. The statement is the assembled, and if valid, placed into memory, otherwise an error message is produced.

Examples:

To place a NOP instruction at address 0x00010000, the command is:

```
asm 10000 nop
```

To interactively assembly memory at address 0x00400000, the command is:

```
asm 400000
```

2.4.2 BC (Block Compare)

Usage: BC addr1 addr2 length

The BC command compares two contiguous blocks of memory on a byte by byte basis. The first block starts at address addr1 and the second starts at address addr2, both of length bytes.

If the blocks are not identical, the address of the first mismatch is displayed. The value for addresses addr1 and addr2 may be an absolute address specified as a hexadecimal value or a symbol name. The value for length may be a symbol name or a number converted according to the user defined radix (hexadecimal by default).

Example:

To verify that the data starting at 0x20000 and ending at 0x30000 is identical to the data starting at 0x80000, the command is:

```
bc 20000 80000 10000
```

2.4.3 BF (Block Fill)

Usage: BF <width> begin end data <inc>

The BF command fills a contiguous block of memory starting at address begin, stopping at address end, with the value data. <Width> modifies the size of the data that is written. If no <width> is specified, the default of word sized data is used.

The value for addresses begin and end may be an absolute address specified as a hexadecimal value, or a symbol name. The value for data may be a symbol name, or a number converted according to the user-defined radix, normally hexadecimal.

The optional value <inc> can be used to increment (or decrement) the data value during the fill.

This command first aligns the starting address for the data access size, and then increments the address accordingly during the operation. Thus, for the duration of the operation, this command performs properly-aligned memory accesses.

Examples:

To fill a memory block starting at 0x00020000 and ending at 0x00040000 with the value 0x1234, the command is:

```
bf 20000 40000 1234
```

To fill a block of memory starting at 0x00020000 and ending at 0x0004000 with a byte value of 0xAB, the command is:

```
bf.b 20000 40000 AB
```

To zero out the BSS section of the target code (defined by the symbols bss_start and bss_end), the command is:

```
bf bss_start bss_end 0
```

To fill a block of memory starting at 0x00020000 and ending at 0x00040000 with data that increments by 2 for each <width>, the command is:

```
bf 20000 40000 0 2
```

2.4.4 BM (Block Move)

Usage: BM begin end dest

The BM command moves a contiguous block of memory starting at address begin and stopping at address end to the new address dest. The BM command copies memory as a series of bytes, and does not alter the original block.

The values for addresses begin, end, and dest may be absolute addresses specified as hexadecimal values, or symbol names. If the destination address overlaps the block defined by begin and end, an error message is produced and the command exits.

Examples:

To copy a block of memory starting at 0x00040000 and ending at 0x00080000 to the location 0x00200000, the command is:

```
bm 40000 80000 200000
```

To copy the target code’s data section (defined by the symbols data_start and data_end) to 0x00200000, the command is:

```
bm data_start data_end 200000
```

NOTE

Refer to “upuser” command for copying code/data into Flash memory.

2.4.5 BR (Breakpoints)

Usage: BR addr <-r> <-c count> <-t trigger>

The BR command inserts or removes breakpoints at address addr. The value for addr may be an absolute address specified as a hexadecimal value, or a symbol name. Count and trigger are numbers converted according to the user-defined radix, normally hexadecimal.

If no argument is provided to the BR command, a listing of all defined breakpoints is displayed.

The -r option to the BR command removes a breakpoint defined at address addr. If no address is specified in conjunction with the -r option, then all breakpoints are removed.

Each time a breakpoint is encountered during the execution of target code, its count value is incremented by one. By default, the initial count value for a breakpoint is zero, but the -c option allows setting the initial count for the breakpoint.

Each time a breakpoint is encountered during the execution of target code, the count value is compared against the trigger value. If the count value is equal to or greater than the trigger value, a breakpoint is encountered and control returned to dBUG. By default, the initial trigger value for a breakpoint is one, but the -t option allows setting the initial trigger for the breakpoint.

If no address is specified in conjunction with the -c or -t options, then all breakpoints are initialized to the values specified by the -c or -t option.

Examples:

To set a breakpoint at the C function main() (symbol _main; see “symbol” command), the command is:

```
br _main
```

When the target code is executed and the processor reaches main(), control will be returned to dBUG.

To set a breakpoint at the C function bench() and set its trigger value to 3, the command is:

```
br _bench -t 3
```

When the target code is executed, the processor must attempt to execute the function bench() a third time before returning control back to dBUG.

To remove all breakpoints, the command is:

```
br -r
```

2.4.6 BS (Block Search)

Usage: BS <width> begin end data

The BS command searches a contiguous block of memory starting at address begin, stopping at address end, for the value data. <Width> modifies the size of the data that is compared during the search. If no <width> is specified, the default of word sized data is used.

The values for addresses begin and end may be absolute addresses specified as hexadecimal values, or symbol names. The value for data may be a symbol name or a number converted according to the user-defined radix, normally hexadecimal.

This command first aligns the starting address for the data access size, and then increments the address accordingly during the operation. Thus, for the duration of the operation, this command performs properly-aligned memory accesses.

Examples:

To search for the 16-bit value 0x1234 in the memory block starting at 0x00040000 and ending at 0x00080000:

```
bs 40000 80000 1234
```

This reads the 16-bit word located at 0x00040000 and compares it against the 16-bit value 0x1234. If no match is found, then the address is incremented to 0x00040002 and the next 16-bit value is read and compared.

To search for the 32-bit value 0xABCD in the memory block starting at 0x00040000 and ending at 0x00080000:

```
bs.l 40000 80000 ABCD
```

This reads the 32-bit word located at 0x00040000 and compares it against the 32-bit value 0x0000ABCD. If no match is found, then the address is incremented to 0x00040004 and the next 32-bit value is read and compared.

2.4.7 DC (Data Conversion)

Usage: DC data

The DC command displays the hexadecimal or decimal value data in hexadecimal, binary, and decimal notation.

The value for data may be a symbol name or an absolute value. If an absolute value passed into the DC command is prefixed by '0x', then data is interpreted as a hexadecimal value. Otherwise, data is interpreted as a decimal value.

All values are treated as 32-bit quantities.

Examples:

To display the decimal and binary equivalent of 0x1234, the command is:

```
dc 0x1234
```

To display the hexadecimal and binary equivalent of 1234, the command is:

```
dc 1234
```

2.4.8 DI (Disassemble)

Usage: DI <addr>

The DI command disassembles target code pointed to by addr. The value for addr may be an absolute address specified as a hexadecimal value, or a symbol name.

Wherever possible, the disassembler will use information from the symbol table to produce a more meaningful disassembly. This is especially useful for branch target addresses and subroutine calls.

The DI command attempts to track the address of the last disassembled opcode. If no address is provided to the DI command, then the DI command uses the address of the last opcode that was disassembled.

The DI command is repeatable.

Examples:

To disassemble code that starts at 0x00040000, the command is:

```
di 40000
```

To disassemble code of the C function main(), the command is:

```
di _main
```

2.4.9 DL (Download Console)

Usage: DL <offset>

The DL command performs an S-record download of data obtained from the console, typically a serial port. The value for offset is converted according to the user-defined radix, normally hexadecimal. Please reference the ColdFire Microprocessor Family Programmer’s Reference Manual for details on the S-Record format.

If offset is provided, then the destination address of each S-record is adjusted by offset.

The DL command checks the destination download address for validity. If the destination is an address outside the defined user space, then an error message is displayed and downloading aborted.

If the S-record file contains the entry point address, then the program counter is set to reflect this address.

Examples:

To download an S-record file through the serial port, the command is:

```
dl
```

To download an S-record file through the serial port, and add an offset to the destination address of 0x40, the command is:

```
dl 0x40
```

2.4.10 DN (Download Network)

Usage: DN <-c> <-e> <-i> <-s> <-o offset> <filename>

The DN command downloads code from the network. The DN command handle files which are either S-record, COFF, ELF or Image formats. The DN command uses Trivial File Transfer Protocol (TFTP) to transfer files from a network host.

In general, the type of file to be downloaded and the name of the file must be specified to the DN command. The -c option indicates a COFF download, the -e option indicates an ELF download, the -i option indicates an Image download, and the -s indicates an S-record download. The -o option works only in conjunction with the -s option to indicate an optional offset for S-record download. The filename is passed directly to the TFTP server and therefore must be a valid filename on the server.

If neither of the -c, -e, -i, -s or filename options are specified, then a default filename and filetype will be used. Default filename and filetype parameters are manipulated using the SET and SHOW commands.

The DN command checks the destination download address for validity. If the destination is an address outside the defined user space, then an error message is displayed and downloading aborted.

For ELF and COFF files which contain symbolic debug information, the symbol tables are extracted from the file during download and used by dBUG. Only global symbols are kept in dBUG. The dBUG symbol table is not cleared prior to downloading, so it is the user's responsibility to clear the symbol table as necessary prior to downloading.

If an entry point address is specified in the S-record, COFF or ELF file, the program counter is set accordingly.

Examples:

To download an S-record file with the name "srec.out", the command is:

```
dn -s srec.out
```

To download a COFF file with the name "coff.out", the command is:

```
dn -c coff.out
```

To download a file using the default filetype with the name "bench.out", the command is:

```
dn bench.out
```

To download a file using the default filename and filetype, the command is:

```
dn
```

2.4.11 GO (Execute)

Usage: GO <addr>

The GO command executes target code starting at address addr. The value for addr may be an absolute address specified as a hexadecimal value, or a symbol name.

If no argument is provided, the GO command begins executing instructions at the current program counter.

When the GO command is executed, all user-defined breakpoints are inserted into the target code, and the context is switched to the target program. Control is only regained when the target code encounters a breakpoint, illegal instruction, trap #15 exception, or other exception which causes control to be handed back to dBUG.

The GO command is repeatable.

Examples:

To execute code at the current program counter, the command is:

```
go
```

To execute code at the C function main(), the command is:

```
go _main
```

To execute code at the address 0x00040000, the command is:

```
go 40000
```

2.4.12 GT (Execute To)

Usage: GT addr

The GT command inserts a temporary breakpoint at addr and then executes target code starting at the current program counter. The value for addr may be an absolute address specified as a hexadecimal value, or a symbol name.

When the GT command is executed, all breakpoints are inserted into the target code, and the context is switched to the target program. Control is only regained when the target code encounters a breakpoint, illegal instruction, or other exception which causes control to be handed back to dBUG.

Examples:

To execute code up to the C function bench(), the command is:

```
gt _bench
```

2.4.13 IRD (Internal Register Display)

Usage: IRD <module.register>

This command displays the internal registers of different modules inside the MCF5xxx. In the command line, module refers to the module name where the register is located and register refers to the specific register to display.

The registers are organized according to the module to which they belong. The available modules on the MCF5xxx are CS, DMA0, DMA1, DMA2, DMA3, DRAMC, PP, MBUS, SIM, TIMER1, TIMER2, UART0 and UART1. Refer to the *MCF5407 Reference Manual* for more information on these modules and the registers they contain.

Example:

```
ird sim.rsr
```

2.4.14 IRM (Internal Register Modify)

Usage: IRM module.register data

This command modifies the contents of the internal registers of different modules inside the MCF5xxx. In the command line, module refers to the module name where the register is located and register refers to the specific register to modify. The data parameter specifies the new value to be written into the register.

The registers are organized according to the module to which they belong. The available modules on the MCF5xxx are CS, DMA0, DMA1, DMA2, DMA3, DRAMC, PP, MBUS, SIM, TIMER1, TIMER2, UART0, and UART1. Refer to the *MCF5407 Reference Manual* for more information on these modules and the registers they contain.

Example:

To modify the TMR register of the first Timer module to the value 0x0021, the command is:

```
irm timer1.tmr 0021
```

2.4.15 HELP (Help)

Usage: HELP <command>

The HELP command displays a brief syntax of the commands available within dBUG. In addition, the address of where user code may start is given. If command is provided, then a brief listing of the syntax of the specified command is displayed.

Examples:

To obtain a listing of all the commands available within dBUG, the command is:

```
help
```

To obtain help on the breakpoint command, the command is:

```
help br
```

2.4.16 LR (Loop Read)

Usage: LR <width> addr

The LR command continually reads the data at addr until a key is pressed. The optional <width> specifies the size of the data to be read. If no <width> is specified, the command defaults to reading word sized data.

Example:

To continually read the longword data from address 0x20000, the command is:

```
lr.l 20000
```

2.4.17 LW (Loop Write)

Usage: LW <width> addr data

The LW command continually writes data to addr. The optional width specifies the size of the access to memory. The default access size is a word.

Examples:

To continually write the longword data 0x12345678 to address 0x20000, the command is:

```
lw.l 20000 12345678
```

Note that the following command writes 0x78 into memory:

```
lw.b 20000 12345678
```

2.4.18 MD (Memory Display)

Usage: MD <width> <begin> <end>

The MD command displays a contiguous block of memory starting at address begin and stopping at address end. The values for addresses begin and end may be absolute addresses specified as hexadecimal values, or symbol names. Width modifies the size of the data that is displayed. If no <width> is specified, the default of word sized data is used.

Memory display starts at the address begin. If no beginning address is provided, the MD command uses the last address that was displayed. If no ending address is provided, then MD will display memory up to an address that is 128 beyond the starting address.

This command first aligns the starting address for the data access size, and then increments the address accordingly during the operation. Thus, for the duration of the operation, this command performs properly-aligned memory accesses.

Examples:

To display memory at address 0x00400000, the command is:

```
md 400000
```

To display memory in the data section (defined by the symbols data_start and data_end), the command is:

```
md data_start
```

To display a range of bytes from 0x00040000 to 0x00050000, the command is:

```
md.b 40000 50000
```

To display a range of 32-bit values starting at 0x00040000 and ending at 0x00050000:

```
md.l 40000 50000
```

2.4.19 MM (Memory Modify)

Usage: MM <width> addr <data>

The MM command modifies memory at the address addr. The value for addr may be an absolute address specified as a hexadecimal value, or a symbol name. Width specifies the size of the data that is modified. If no <width> is specified, the default of word sized data is used. The value for data may be a symbol name, or a number converted according to the user-defined radix, normally hexadecimal.

If a value for data is provided, then the MM command immediately sets the contents of addr to data. If no value for data is provided, then the MM command enters into a loop. The loop obtains a value for data, sets the contents of the current address to data, increments the address according to the data size, and repeats. The loop terminates when an invalid entry for the data value is entered; i.e., a period.

This command first aligns the starting address for the data access size, and then increments the address accordingly during the operation. Thus, for the duration of the operation, this command performs properly-aligned memory accesses.

Examples:

To set the byte at location 0x00010000 to be 0xFF, the command is:

```
mm.b 10000 FF
```

To interactively modify memory beginning at 0x00010000, the command is:

```
mm 10000
```

2.4.20 MMAP (Memory Map Display)

Usage: mmap

This command displays the memory map information for the M5251C3 evaluation board. The information displayed includes the type of memory, the start and end address of the memory, and the port size of the memory. The display also includes information on how the chip-selects are used on the board.

Here is an example of the output from this command:

```
      Type            Start          End      Port Size
----------------------------------------------------
SDRAM             0x00000000   0x003FFFFF   32-bit
Vector Table      0x00000000   0x000003FF   32-bit
USER SPACE        0x00020000   0x003FFFFF   32-bit
MBAR              0x10000000   0x100003FF   32-bit
Internal SRAM     0x20000000   0x20000FFF   32-bit
External SRAM     0x30000000   0x3007FFFF   32-bit
Flash              0xFFE00000   0xFFFFFFFF   16-bit
Chip Selects
----------------
CS0  Flash
CS1  Ethernet controller
CS2  not in use
CS3  not in use
```

2.4.21 RD (Register Display)

Usage: RD <reg>

The RD command displays the register set of the target. If no argument for reg is provided, then all registers are displayed. Otherwise, the value for reg is displayed.

dBUG preserves the registers by storing a copy of the register set in a buffer. The RD command displays register values from the register buffer.

Examples:

To display all the registers and their values, the command is:

```
rd
```

To display only the program counter:

```
rd pc
```

Here is an example of the output from this command:

```
PC: 00000000 SR: 2000 [t.Sm.000...xnzvc]
An: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 01000000
Dn: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

2.4.22 RM (Register Modify)

Usage: RM reg data

The RM command modifies the contents of the register reg to data. The value for reg is the name of the register, and the value for data may be a symbol name, or it is converted according to the user-defined radix, normally hexadecimal.

dBUG preserves the registers by storing a copy of the register set in a buffer. The RM command updates the copy of the register in the buffer. The actual value will not be written to the register until target code is executed.

Examples:

To change register D0 on MC68000 and ColdFire to contain the value 0x1234, the command is:

```
rm D0 1234
```

2.4.23 RESET (Reset the Board and dBUG)

Usage: RESET

The RESET command resets the board and dBUG to their initial power-on states.

The RESET command executes the same sequence of code that occurs at power-on. If the RESET command fails to reset the board adequately, cycle the power or press the reset button.

Examples:

To reset the board and clear the dBUG data structures, the command is:

```
reset
```

2.4.24 SET (Set Configurations)

Usage: SET <option value>

The SET command allows the setting of user-configurable options within dBUG. With no arguments, SET displays the options and values available. The SHOW command displays the settings in the appropriate format. The standard set of options is listed below.

- baud—This is the baud rate for the first serial port on the board. All communications between dBUG and the user occur using either 9600 or 19200 bps, eight data bits, no parity, and one stop bit, 8N1, with no flow control.
- base—This is the default radix for use in converting a number from its ASCII text representation to the internal quantity used by dBUG. The default is hexadecimal (base 16), and other choices are binary (base 2), octal (base 8), and decimal (base 10).

- client—This is the network Internet Protocol (IP) address of the board. For network communications, the client IP is required to be set to a unique value, usually assigned by your local network administrator.
- server—This is the network IP address of the machine which contains files accessible via TFTP. Your local network administrator will have this information and can assist in properly configuring a TFTP server if one does not exist.
- gateway—This is the network IP address of the gateway for your local subnetwork. If the client IP address and server IP address are not on the same subnetwork, then this option must be properly set. Your local network administrator will have this information.
- netmask—This is the network address mask to determine if use of a gateway is required. This field must be properly set. Your local network administrator will have this information.
- filename—This is the default filename to be used for network download if no name is provided to the DN command.
- filetype—This is the default file type to be used for network download if no type is provided to the DN command. Valid values are: “srecord”, “coff”, and “elf”.
- mac—This is the ethernet Media Access Control (MAC) address (a.k.a hardware address) for the evaluation board. This should be set to a unique value, and the most significant nibble should always be even.

Examples:

To set the baud rate of the board to be 19200, the command is:

```
set baud 19200
```

NOTE

See the SHOW command for a display containing the correct formatting of these options.

2.4.25 SHOW (Show Configurations)

Usage: SHOW <option>

The SHOW command displays the settings of the user-configurable options within dBUG. When no option is provided, SHOW displays all options and values.

Examples:

To display all options and settings, the command is:

```
show
```

To display the current baud rate of the board, the command is:

```
show baud
```

Here is an example of the output from a show command:

```
dBUG> show
        base: 16
        baud: 19200
      server: 192.0.0.1
```

```
  client: 192.0.0.2
 gateway: 0.0.0.0
 netmask: 255.255.255.0
filename: test.srec
filetype: S-Record
    mac: 00:CF:52:49:C3:01
```

2.4.26 STEP (Step Over)

Usage: STEP

The STEP command can be used to “step over” a subroutine call, rather than tracing every instruction in the subroutine. The ST command sets a temporary breakpoint one instruction beyond the current program counter and then executes the target code.

The STEP command can be used to “step over” BSR and JSR instructions.

The STEP command will work for other instructions as well, but note that if the STEP command is used with an instruction that will not return, i.e. BRA, then the temporary breakpoint may never be encountered and dBUG may never regain control.

Examples:

To pass over a subroutine call, the command is:

```
step
```

2.4.27 SYMBOL (Symbol Name Management)

Usage: SYMBOL <symb> <-a symb value> <-r symb> <-c|l|s>

The SYMBOL command adds or removes symbol names from the symbol table. If only a symbol name is provided to the SYMBOL command, then the symbol table is searched for a match on the symbol name and its information displayed.

The -a option adds a symbol name and its value into the symbol table. The -r option removes a symbol name from the table.

The -c option clears the entire symbol table, the -l option lists the contents of the symbol table, and the -s option displays usage information for the symbol table.

Symbol names contained in the symbol table are truncated to 31 characters. Any symbol table lookups, either by the SYMBOL command or by the disassembler, will only use the first 31 characters. Symbol names are case-sensitive.

Symbols can also be added to the symbol table via in-line assembly labels and ethernet downloads of ELF formatted files.

Examples:

To define the symbol “main” to have the value 0x00040000, the command is:

```
symbol -a main 40000
```

To remove the symbol “junk” from the table, the command is:

```
symbol -r junk
```

To see how full the symbol table is, the command is:

```
symbol -s
```

To display the symbol table, the command is:

```
symbol -l
```

2.4.28 TRACE (Trace Into)

Usage: TRACE <num>

The TRACE command allows single-instruction execution. If num is provided, then num instructions are executed before control is handed back to dBUG. The value for num is a decimal number.

The TRACE command sets bits in the processors’ supervisor registers to achieve single-instruction execution, and the target code executed. Control returns to dBUG after a single-instruction execution of the target code.

This command is repeatable.

Examples:

To trace one instruction at the program counter, the command is:

```
tr
```

To trace 20 instructions from the program counter, the command is:

```
tr 20
```

2.4.29 UPDBUG (Update dBUG)

Usage: UPDBUG

The UPDBUG command is used to update the dBUG image in Flash. When updates to the M5251C3 dBUG are available, the updated image is downloaded to address 0x00020000. The new image is placed into Flash using the UPDBUG command. The user is prompted for verification before performing the operation. Use this command with extreme caution, as any error can render dBUG useless!

2.4.30 UPUSER (Update User Flash)

Usage: UPUSER <bytes>

The UPUSER command places user code and data into space allocated for the user in Flash. The optional parameter bytes specifies the number of bytes to copy into the user portion of Flash.If the bytes parameter is omitted, then this command writes to the entire user space. There are seven sectors of 256K each available as user space. Users access this memory starting at address 0xFFE40000.

Examples:

To program all 7 sectors of user Flash, the command is:

```
upuser
```

To program only 1000 bytes into user Flash, the command is:

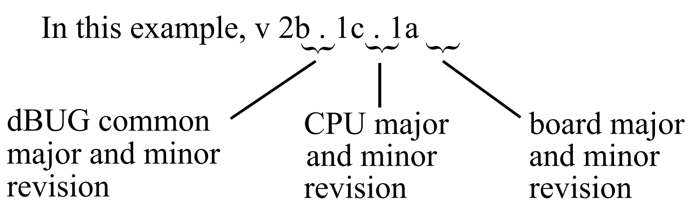
```
upuser 1000
```

2.4.31 VERSION (Display dBUG Version)

Usage: VERSION

The VERSION command displays the version information for dBUG. The dBUG version, build number and build date are all given.

The version number is separated by a decimal, for example, “v 2b.1c.1a”.


In this example, v 2b . 1c . 1a
dBUG common major and minor revision
CPU major and minor revision
board major and minor revision

The version date is the day and time at which the entire dBUG monitor was compiled and built.

Examples:

To display the version of the dBUG monitor, the command is:

```
version
```

2.5 TRAP #15 Functions

An additional utility within the dBUG firmware is a function called the TRAP 15 handler. This function can be called by the user program to utilize various routines within the dBUG, to perform a special task, and to return control to the dBUG. This section describes the TRAP 15 handler and how it is used.

There are four TRAP #15 functions. These are: OUT_CHAR, IN_CHAR, CHAR_PRESENT, and EXIT_TO_dBUG.

2.5.1 OUT_CHAR

This function (function code 0x0013) sends a character, which is in lower 8 bits of D1, to the terminal.

Assembly example:

```
        /* assume d1 contains the character */
        move.l          #$0013,d0         Selects the function
        TRAP            #15               The character in d1 is sent to terminal
```

C example:

```
void board_out_char (int ch)
{
        /* If your C compiler produces a LINK/UNLK pair for this routine,
         * then use the following code which takes this into account
        */
#if  1
        /* LINK a6,#0 -- produced by C compiler */
        asm (" move.l8(a6),d1");            /* put 'ch'into d1 */
        asm (" move.l#0x0013,d0");  /* select the function */
        asm (" trap#15");                   /* make the call */
        /* UNLK a6  -- produced by C compiler */
#else
        /*  If C compiler does not produce a LINK/UNLK pair, the use
         *  the following code.
        */
         asm (" move.l4(sp),d1");           /* put 'ch'into d1 */
        asm (" move.l#0x0013,d0");  /* select the function */
        asm (" trap#15");                   /* make the call */
#endif
}
```

2.5.2 IN_CHAR

This function (function code 0x0010) returns an input character (from the terminal) to the caller. The returned character is in D1.

Assembly example:

```
        move.l   #$0010,d0         Select the function
        trap     #15               Make the call, the input character is in d1.
```

C example:

```
int board_in_char (void)
{
        asm (" move.l#0x0010,d0");          /* select the function */
        asm (" trap#15");                   /* make the call */
        asm (" move.ld1,d0");               /* put the character in d0 */
}
```

2.5.3 CHAR_PRESENT

This function (function code 0x0014) checks if an input character is present to receive. A value of zero is returned in D0 when no character is present. A non-zero value in D0 means a character is present.

Assembly example:

```
        move.l   #$0014,d0         Select the function
        trap     #15       Make the call,  d0 contains the response (yes/no).
```

C example:

```
int board_char_present (void)
{
        asm (" move.l#0x0014,d0");          /* select the function */
        asm (" trap#15");                   /* make the call */
}
```

2.5.4 EXIT_TO_dBUG

This function (function code 0x0000) transfers the control back to the dBUG by terminating the user code. The register context is preserved.

Assembly example:

```
        move.l   #$0000,d0         Select the function
        trap     #15               Make the call,  exit to dBUG.
```

C example:

```
void board_exit_to_dbug (void)
{
        asm (" move.l#0x0000,d0");          /* select the function */
        asm (" trap#15");                   /* exit and transfer to dBUG */
}
```

Chapter 3
Hardware Description and Reconfiguration

This chapter provides a functional description of the M5251C3 board hardware. With the description given here and the schematic diagrams in Appendix A, the user can gain a good understanding of the board's design. In this manual, an active low signal is indicated by a "-" preceding the signal name in the text and a bar over the signal name in the schematics.

3.1 Processor and Support Logic

This part of the chapter discusses the CPU and general support logic on the M5251C3 board.

3.1.1 Processor

The microprocessor used on the M5251C3 is the highly integrated ColdFire® MCF5251, 32-bit processor. The MCF5251 implements a ColdFire Version 2 core with 8-Kbyte instruction cache, three UART channels, two timers, 128-Kbytes of SRAM, a QSPI (Queued Serial Peripheral Interface) module, two I^{2}C modules, 3x I^{2}S modules, an IDE module, a dedicate ATA interface with DMA support, two FlexCAN modules, an OTG USB 2.0 controller with integrated physical interface, Real Time Clock (RTC), a Flash memory stick interface, 60 parallel I/O ports (which are multiplexed with other signals) and the system integration module (SIM). All of the core processor registers are 32 bits wide except for the Status Register (SR) which is 16 bits wide. This processor communicates with external devices over a 16-bit wide data bus, D[31:16]. The chip can address 64 Mbytes of memory space using a 25-bit wide address bus and internal chip-select logic.

The MCF5251 processor has the capability to support both an IEEE JTAG-compatible port and a BDM debug port. These ports are multiplexed and can be used with third party tools to allow the user to download code to the board. The board is configured to boot up in the normal/BDM mode of operation. The BDM signals are available at port (J12). The processor also has the logic to generate up to four (4) chip selects, and support for a bank of SDRAM (included on the evaluation board as 8 Mbytes in total configured as 4 M x 16). The -SDRAM_CS1 signal is used to provide selection and control of this bank of SDRAM.

3.1.2 Reset Logic

The reset logic provides system initialization. Reset occurs during power-on or via assertion of the signal -RESET which causes the MCF5251 to reset. Reset is also triggered by the reset switch (S1) which resets the entire processor/system.

A hard reset and voltage sense controller (U18) is used to produce an active low power-on RESET signal. The reset switch S1 is fed into U18 which generates the signal which is fed to the MCF5251 reset, -RESET. The -RESET signal is an open collector signal and so can be wire OR’ed with other reset signals from additional peripherals.

dBUG configures the MCF5251 microprocessor internal resources during initialization. The instruction cache is invalidated and disabled. The Vector Base Register, VBR, contains an address which initially points to the Flash memory. The contents of the exception table are written to address $00000000 in the SDRAM. The Software Watchdog Timer is disabled, the Bus Monitor is enabled, and the internal timers are placed in a stop condition. The interrupt controller registers are initialized with unique interrupt level/priority pairs. A memory map for the entire board can be seen in Table 3-1.

3.1.3 HIZ Signal

The assertion of the -HIZ signal forces all output drivers to a high-impedance state. On the M5251C3 board the high impedance signal is pulled to +3.3V via a 4.7K pull-up resistor, ensuring that the output drivers will not be in a high-impedance state during reset.

3.1.4 Clock Circuitry

The M5251C3 board uses a 11.2896MHz crystal (X2 on the schematics) to provide the clock to the processor (IC1). In addition to the 11.2896MHz crystal, there is also a 24MHz crystal (X4) which feeds the USB section of IC1, a 32.768KHz crystal (X3) which feeds the RTC section of IC1 and a second 11.2896 MHz crystal (X1) that can be used to provide an external audio clock source

3.1.5 Watchdog Timer

The duration of the Watchdog is selected by the SWT[1:0] bits in the System Protection and Control Register (SYPCR), SWT[1:0] = 11 gives a maximum timeout period of 2^{28}/System frequency. The dBUG monitor initializes these bits with the value 0x11, which provides the maximum time-out period, but dBUG does **NOT** enable the watchdog timer via the SYPCR register SWE bit.

3.1.6 Interrupt Sources

The ColdFire® family of processors can receive seven levels of interrupt priorities. When the processor receives an interrupt which has a higher priority than the current interrupt mask (in the status register), it will perform an interrupt acknowledge cycle at the end of the current instruction cycle. This interrupt acknowledge cycle indicates to the source of the interrupt that the request is being acknowledged and the device should provide the proper vector number to indicate where the service routine for this interrupt level is located. If the source of interrupt is not capable of providing a vector, its interrupt should be set up as an autovector interrupt which directs the processor to a predefined entry in the exception table. (See the *MCF5251 Reference Manual*).

The processor goes to an exception routine via the exception table. This table is stored in the Flash EEPROM. The address of the table location is stored in the VBR. The dBUG ROM monitor writes a copy of the exception table into the RAM starting at $00000000. To set an exception vector, the user places the address of the exception handler in the appropriate vector in the vector table located at $00000000 and then points the VBR to $00000000.

The MCF5251 microprocessor has seven external interrupt request lines -INT[6:0], all of which are multiplexed with other functions. The interrupt controller is capable of providing up to 75 interrupt sources. These sources include:-

- External interrupt signals -INT[6:0]

- Software watchdog timer module
- Two general purpose timer modules
- UART module
- I^{2}C module
- Audio interface modules
- DMA module
- QSPI module
- CAN module
- USB module
- ATA module
- Flash media module

All external interrupt inputs are edge sensitive. The active level is programmable. An interrupt request must be held valid until an IACK cycle starts to guarantee correct processing. Each interrupt input can have it's priority programmed by setting the xIPL[2:0] bits in the Interrupt Control Registers.

NOTE

No interrupt sources should have the same level and priority as another. Programming two interrupt sources with the same level and priority can result in undefined operation.

The M5251C3 hardware uses -INT1 to support the ABORT function using the ABORT switch (S2). This switch is used to force an interrupt (level 7, priority 3) if the user's program execution should be aborted without issuing a RESET command. See Section 2.2.2.2, "ABORT Button" on page 2-4 for more information on ABORT.) Since the ABORT switch is not capable of generating a vector in response to a level seven interrupt acknowledge from the processor, the dBUG programs this interrupt request for autovector mode.

See the *MCF5251 Reference Manual* for more information about the interrupt controller.

3.1.7 Internal SRAM

The MCF5251 processor has 128 Kbtyes of internal memory which may be programmed as data or instruction memory. This memory is mapped to 0x20000000 and configured as data space but is not used by the dBUG monitor except during system initialization. After system initialization is complete, the internal memory is available to the user. The memory can be relocated to any 32-Kbyte boundary.

3.1.8 MCF5251 Registers and Memory Map

The memory and I/O resources of the M5251C3 hardware are divided into two groups, MCF5251 internal and external resources. All the I/O registers are memory mapped.

The MCF5251 processor has built in logic and up to four chip-select pins (-CS[3:0]) which are used to enable external memory and I/O devices. In addition there are -SDRAS and -SDCAS lines available for controlling SDRAMs. There are registers to specify the address range, type of access and the method of

-TA generation for each chip-select. These registers are programmed by the dBUG monitor to map the external memory and I/O devices.

The M5251C3 uses the following signals to select external peripherals:

-CS0 to enable the Flash ROM (See Section 3.1.13, "Flash ROM.")

-SDRAS, -SDCAS, and -SDRAM_CS1 to enable the SDRAM (See Section 3.1.12, "SDRAM.")

The chip select mechanism of the MCF5251 processor allows the memory mapping to be defined for the required memory space (User/Supervisor, Program/Data spaces).

All of the MCF5251 internal registers, configuration registers, parallel I/O port registers, UART registers and system control registers are mapped by the MBAR registers at any 1- KByte boundary. The MBAR1 register is mapped to 0x10000000 and MBAR2 mapped to 0x80000000 by the dBUG monitor. For a complete map of these registers, see the *MCF5251 Reference Manual*.

The M5251C3 board has 8 Mbytes of SDRAM installed. See Section 3.1.12, "SDRAM" for a discussion of the SDRAM on the board. The dBUG ROM monitor is programmed in one AMD Am29LV160DB-90 Flash ROM, which occupies 2 Mbytes of the address space. The first 256 Kbytes (that is., the first sector) are used by the ROM Monitor and the remainder is left for the user. (See Section 3.1.13, "Flash ROM.")

Figure 3-1 shows the M5251C3 memory map.

Table 3-1 The M5272C3 Memory Map

Address Range	Signal and Device	Memory Access Time
$00000000-$00020000	SDRAM space for dBug ROM monitor use	Refer to manufacturer specification
$00020000-$003FFFFF	SDRAM space	Refer to manufacturer specification
$10000000-$100003FF	System Integration Module (SIM) registers	Internal access
$10000000-$10000054	MBAR - Module Base Address Reg.	Refer to MCF5251UM SIM section
$80000000-$80000198	MBAR2 - Module Base Address Reg. 2	Refer to MCF5251UM SIM section
$20000000-$20000FFF	SRAM	Internal access (1 cycle)
$FFE00000-$FFFFFFFF	-CS0, 2M Flash ROM	8-7-7-7

All of the unused area of the memory map is available to the user.

3.1.9 Reset Vector Mapping

After reset, the processor attempts to read the initial stack pointer and program counter values from locations $00000000 & $00000004 (the first eight bytes of memory space). This requires the board to have a non-volatile memory device in this range with the correct information stored in it. In some systems, however, it is preferred to have RAM starting at address $00000000. The MCF5251 processor chip-select zero (-CS0) responds to any accesses after reset until the CSMR0 is written. Since -CS0 (the global chip select) is connected to the Flash ROM (U11), the Flash ROM initially appears at address $00000000 which provides the initial stack pointer and program counter (the first eight bytes of the Flash ROM). The initialization routine then programs the chip-select logic, locates the Flash ROM to start at $FFE00000, and configures the rest of the internal and external peripherals.

3.1.10 TA Generation

The processor starts a bus cycle by asserting -CSx with the other control signals. The processor then waits for a transfer acknowledgment (-TA) either from within (Auto acknowledge - AA mode) or from the externally addressed device before it can complete the bus cycle. -TA is used to indicate the completion of the bus cycle. It also allows devices with different access times to communicate with the processor properly (that is, asynchronously). The MCF5251 processor, as part of the chip-select logic, has a built-in mechanism to generate -TA for all external devices which do not have the capability to generate this signal. For example the Flash ROM cannot generate a -TA.signal. The chip-select logic is programmed by the dBUG ROM Monitor to generate -TA internally after a pre-programmed number of wait states.

3.1.11 Wait State Generator

The Flash ROM and SDRAM on the board may require some adjustments to the cycle time of the processor to make them compatible with the processor’s external bus speed. To extend the CPU bus cycles for the slower devices, the chip-select logic of the MCF5251 processor can be programmed to generate an internal -TA after a given number of wait states. See Table 3-1 for information about the address space of the memory and refer to the manufacturer’s specification for wait state requirements of the SDRAM and Flash ROM.

3.1.12 SDRAM

The M5251C3 has one 64-Mbit device on the board, in a 16-bit wide data bus configuration. The MCF5251 processor supports one bank of SDRAM, which on this board is represented by SDRAM device, (U12). These are connected to the MCF5251 to provide 4Mx16 of memory.

3.1.13 Flash ROM

There is one 2-Mbyte Flash ROM on the M5251C3, (U11).

The board is shipped with one AMD Am29LV160DB, 2-Mbyte Flash ROM. The first 256 Kbytes of the Flash contains the ROM Monitor firmware dBUG. The remaining Flash memory is available to the user.

The MCF5251 chip-select logic can be programmed to generate the -TA for -CS0 signal after a certain number of wait states (that is, auto-acknowledge mode). The dBUG monitor programs this parameter to be six wait-states.

3.2 Serial Communication Channels

The M5251C3 offers five types of serial communications channels. They are discussed in this section.

3.2.1 MCF5251 UARTs

The MCF5251 device has three built-in UARTs, each with its own software programmable baud rate generator. UART0 is the ROM Monitor to Terminal output. The ROM Monitor programs the interrupt level for UART0 to Level 3, priority 2 and autovector mode of operation. See the *MCF5251 Reference Manual* for programming the UARTs and their register maps.

3.2.2 QSPI Module

The QSPI (Queued Serial Peripheral Interface) module provides a serial peripheral interface with queued transfer capability. It will support up to 16 stacked transfers at one time, minimizing CPU intervention between transfers. Transfer RAMs in the QSPI are indirectly accessible using address and data registers.

Functionality is very similar, but not identical, to the QSPI portion of the QSM (Queued Serial Module) implemented in the MC68332 processor.

- Programmable queue to support up to 16 transfers without user intervention
- Supports transfer sizes of 8 to 16 bits in 1-bit increments
- Four peripheral chip-select lines for control of up to 15 devices
- Baud rates from 274.5 Kbps to 17.5 Mbps at 140 MHz.
- Programmable delays before and after transfers
- Programmable clock phase and polarity
- Supports wrap-around mode for continuous transfers

See the *MCF5251 Reference Manual* for more detail. The QSPI signals from the MCF5251 device are brought out to connector (J5). Some of these signals are multiplexed with other functions.

3.2.3 I^{2}C Modules

The MCF5251 processor's two I^{2}C modules include the following features:

- Compatibility with the I^{2}C bus standard
- Multimaster operation
- Software programmable for one of 64 different clock frequencies
- Software selectable acknowledge bit
- Interrupt driven byte by byte data transfer
- Arbitration-lost interrupt with auto mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation and detection
- Repeated start signal generation
- Acknowledge bit generation and detection
- Bus busy detection

For further details, see the *MCF5251 Reference Manual*.

3.2.4 FlexCAN Modules

The MCF5251 processor's two FlexCAN modules includes the following features:

- Full implementation of the CAN protocol specification version 2.0B
  - — Standard data and remote frames (up to 109 bits long)
  - — Extended data and remote frames (up to 127 bits long)

- — 0–8 bytes data length
- — Programmable bit rate up to 1 Mbps
- — Content-related addressing
- Up to 32 flexible message buffers of zero to eight bytes data length, each configurable as Rx or Tx, all supporting standard and extended messages
- Listen-only mode capability
- Three programmable mask registers: global (for MBs 0-13 and 16-31), special for MB14, and special for MB15
- Programmable transmission priority scheme: lowest ID or lowest buffer number
- Time stamp based on 16-bit, free-running timer
- Global network time, synchronized by a specific message
- Programmable I/O modes
- Maskable interrupts
- Independent of the transmission medium (an external transceiver assumed)
- Open-network architecture
- Multimaster bus
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages

For further details, see the *MCF5251 Reference Manual*.

3.2.5 USB 2.0 OTG Module

The MCF5251 processor's USB 2.0 OTG module includes the following features:

- Compliance with USB specification revision 2.0
- Support for operation as a standalone USB host controller, including support of enhanced host controller interface (EHCI)
- USB device mode
- USB On-The-Go mode, including host capability
- Support for high-speed (480 Mbps), full-speed (12 Mbps), and low-speed host (1.5 Mbit/s) operations
- Support for internal PHY (with UTMI+ interface) and external PHYs with ULPI interface
- Support for operation as a standalone USB device
- Support for one upstream facing port
  - — Support for four programmable, bidirectional USB endpoints
  - — Host mode support for direct connect of FS/LS devices.

For further details, see the *MCF5251 Reference Manual*.

3.3 General Purpose I/O Pins

The MCF5251 offers 60-bits of general-purpose I/O of which 6 are dedicated general purpose inputs and 3 are dedicated general purpose outputs. Seven of the GPIO lines are also available as edge sensitive interrupt inputs plus one dedicated pin is used to generate a WAKEUP interrupt from low power mode. The functions of all I/O pins are individually programmable, since they are multiplexed with other pin functions. All general-purpose I/O pins (unless dedicated as either only input or output) can be individually selected as input or output pins. After reset, all software configurable multi-function GPIO pins default to general purpose input pins. At the same time, all multifunction pins that are not shared with a GPIO pin default to high impedance. Internal pullup resistors avoid unknown read values in order to reduce power consumption. They remain active until the corresponding port direction registers are programmed.

Control registers are provided for each pin to select the function (GPIO or peripheral pin) assigned to each pin individually. Pins can have from 1 to 4 functions including GPIO.

For further details, see the *MCF5251 Reference Manual*.

3.4 Audio Module

The MCF5251 processor's audio module includes the following features:

- Support for reception and transmission of digital audio over serial interfaces IIS/EIAJ and digital interface IEC958
- 3x IIS/EIAJ interfaces
- 2x IEC958 receivers (4x multiplexed inputs)
- 1x IEC958 transmitter - two outputs - one with professional subcoding, one with consumer subcoding
- Allows direct transmission of received audio to an audio transmitter without CPU intervention.
- IEC958 receivers and transmitter support main audio, plus handling of IEC958 C, U and V sub-channels
- Frequency measurement block - precise measurement of the incoming sample frequency

For further details, see the *MCF5251 Reference Manual*.

3.5 Analog to Digital Converter (ADC) Module

The MCF5251 processor's ADC module includes the following features:

- Sigma-Delta based ADC with 12-bit resolution
- Six dedicated inputs - ADIN0/GPI52, ADIN1/GPI53, ADIN2/GPI54, ADIN3/GPI55, ADIN4/GPI56 and ADIN5/GPI57
- Single output - ADOUT/SCLK4/GPIO58, provides the reference voltage which requires an external integrator (resistor/capacitor circuit)
- Software interrupt provided when the ADC measurement is complete

For further details, see the *MCF5251 Reference Manual*.

3.6 Flash Memory Card/IDE Interface Module

The SCF5251 memory bus allows connection of an IDE hard disk drive or SmartMedia flash card with a minimum of external hardware. It can interface with both an IDE device and a SmartMedia device connected, although both cannot be supported simultaneously, as the $\overline{\text{IDE-DIOR}}$ and $\overline{\text{IDE-DIOW}}$ signals can only be used to interface to one or the other.

For further details, see the *MCF5251 Reference Manual*.

3.7 ATA Interface Module

The MCF5251 processor's ATA Interface module includes the following features:

- Programmable timing on the ATA bus. The interface works with wide range of bus frequencies.
- Compliance with ATA-6 specification
  - Support for PIO modes 0, 1, 2, 3, and 4
  - Support for multiword DMA modes 0, 1, and 2
  - Support for ultra DMA modes 0, 1, 2, 3, and 4 with bus clock of at least 50 MHz
  - Support for ultra DMA mode 5 with bus clock of at least 80 MHz
- 128-byte FIFO part of interface
- FIFO receive alarm and FIFO transmit alarm to DMA unit
- Zero-wait cycles transfer between DMA bus and FIFO, which allows fast FIFO reading and writing

For further details, see the *MCF5251 Reference Manual*.

3.8 Real-Time Clock (RTC) Module

The MCF5251 processor's RTC module is a mixed-signal circuit that provides an indicator of time (in seconds) for various purposes in the system. The MCF5251 processor's RTC module includes the following features:

- 32.768 kHz crystal
- Independent power supply pins to allow battery backup operation
- Circuitry to detect power supply tampering

For further details, see the *MCF5251 Reference Manual*.

3.9 Debug Connector J12

The MCF5251 processor has a Background Debug Mode (BDM) port, which supports Real-Time Trace Support and Real-Time Debug. The signals which are necessary for debug are available at connector (J12). Figure 3-1 shows the (J12) connector pin assignment.

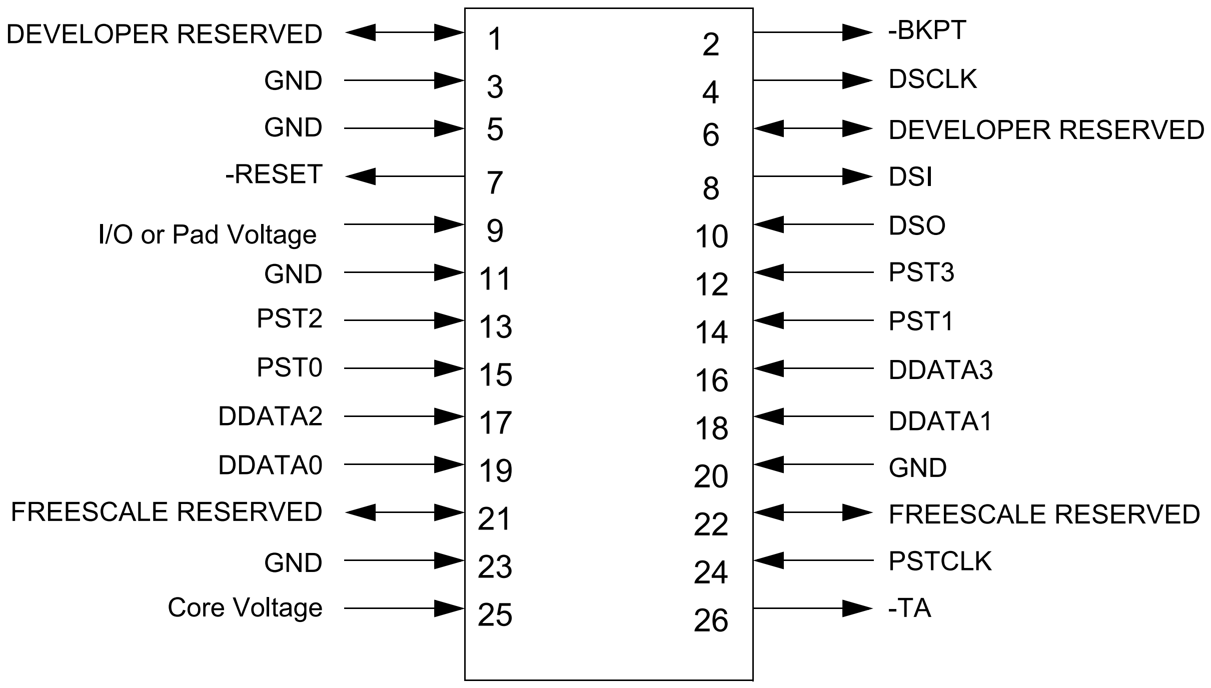


Figure 3-1 J12 Connector Pin Assignment

Appendix A Schematics

This section is composed of the schematics for the M5251C3 Evaluation Board, including the schematic for the MCF5251 Evaluation Board.

Figure 3-2 shows the Hierarchical Block Diagram.

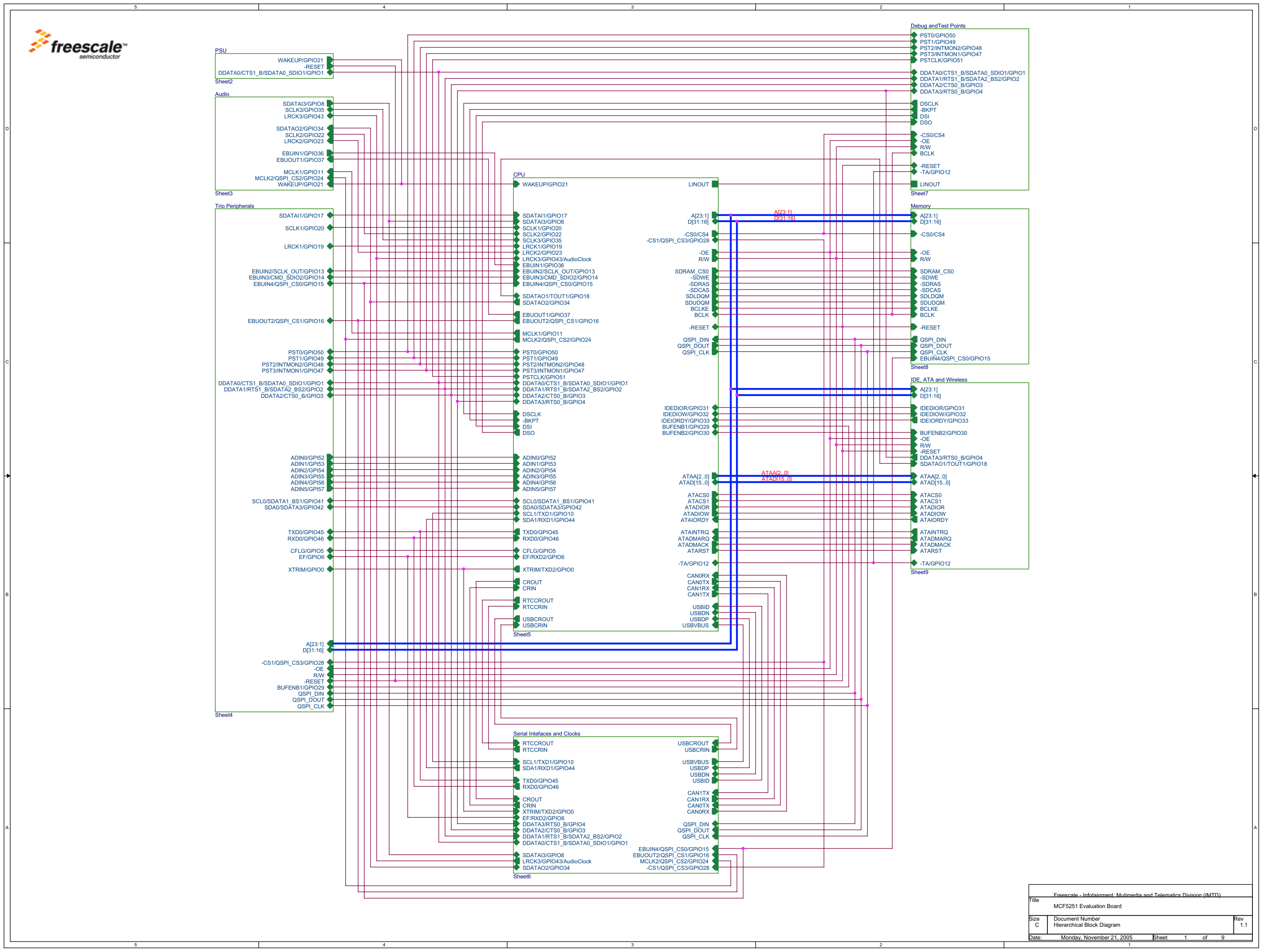


Figure 3-2 Hierarchical Block Diagram

Figure 3-3 shows the Audio Interface.

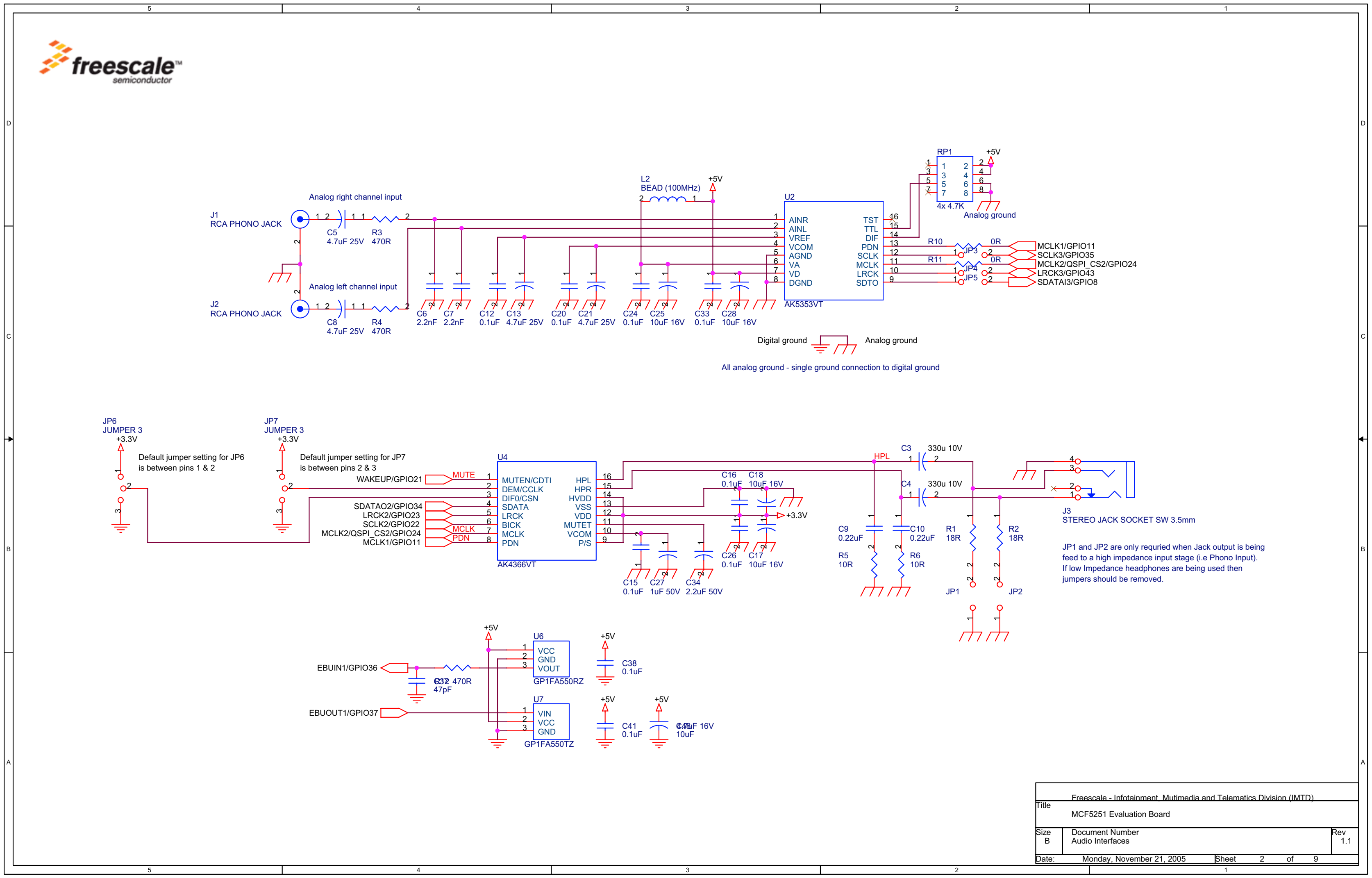

freescale semiconductor
Analog right channel input
J1 RCA PHONO JACK
C5 4.7uF 25V
R3 470R
Analog left channel input
J2 RCA PHONO JACK
C8 4.7uF 25V
R4 470R
L2 BEAD (100MHz)
+5V
U2 AK5353VT
AINR AINL VREF VCOM AGND VA VD DGND
TST TTL DIF PDN SCLK MCLK LRCK SDTO
RP1 4x 4.7K
Analog ground
R10 0R
R11 0R
JP3 JP4 JP5
MCLK1/GPIO11
SCLK3/GPIO35
MCLK2/QSPI_CS2/GPIO24
LRCK3/GPIO43
SDATAI3/GPIO8
C6 2.2nF C7 2.2nF C12 0.1uF C13 4.7uF 25V C20 0.1uF C21 4.7uF 25V C24 0.1uF C25 10uF 16V C33 0.1uF C28 10uF 16V
Digital ground
Analog ground
All analog ground - single ground connection to digital ground
JP6 JUMPER 3 +3.3V
Default jumper setting for JP6 is between pins 1 & 2
JP7 JUMPER 3 +3.3V
Default jumper setting for JP7 is between pins 2 & 3
WAKEUP/GPIO21
MUTE
SDATAO2/GPIO34
LRCK2/GPIO23
SCLK2/GPIO22
MCLK2/QSPI_CS2/GPIO24
MCLK1/GPIO11
MCLK
PDN
U4 AK4366VT
MUTEN/CDTI DEM/CCLK DIF0/CSN SDATA LRCK BICK MCLK PDN
HPL HPR HVDD VSS VDD MUTET VCOM P/S
C16 0.1uF C18 10uF 16V
+3.3V
C26 0.1uF C17 10uF 16V
C15 0.1uF C27 1uF 50V C34 2.2uF 50V
HPL
C3 330u 10V
C4 330u 10V
C9 0.22uF C10 0.22uF
R5 10R R6 10R
R1 18R R2 18R
JP1 JP2
J3 STEREO JACK SOCKET SW 3.5mm
JP1 and JP2 are only requried when Jack output is being feed to a high impedance input stage (i.e Phono Input). If low Impedance headphones are being used then jumpers should be removed.
+5V
U6 GP1FA550RZ
VCC GND VOUT
C38 0.1uF
EBUIN1/GPIO36
R12 470R
C32 47pF
U7 GP1FA550TZ
VIN VCC GND
EBUOUT1/GPIO37
C41 0.1uF
C40 10uF 16V
Freescale - Infotainment, Mutimedia and Telematics Division (IMTD)
Title MCF5251 Evaluation Board
Size B
Document Number Audio Interfaces
Rev 1.1
Date: Monday, November 21, 2005
Sheet 2 of 9

Figure 3-3 Audio Interfaces

Figure 3-4 shows the separate schematic for the MCF5251 Evaluation Board.

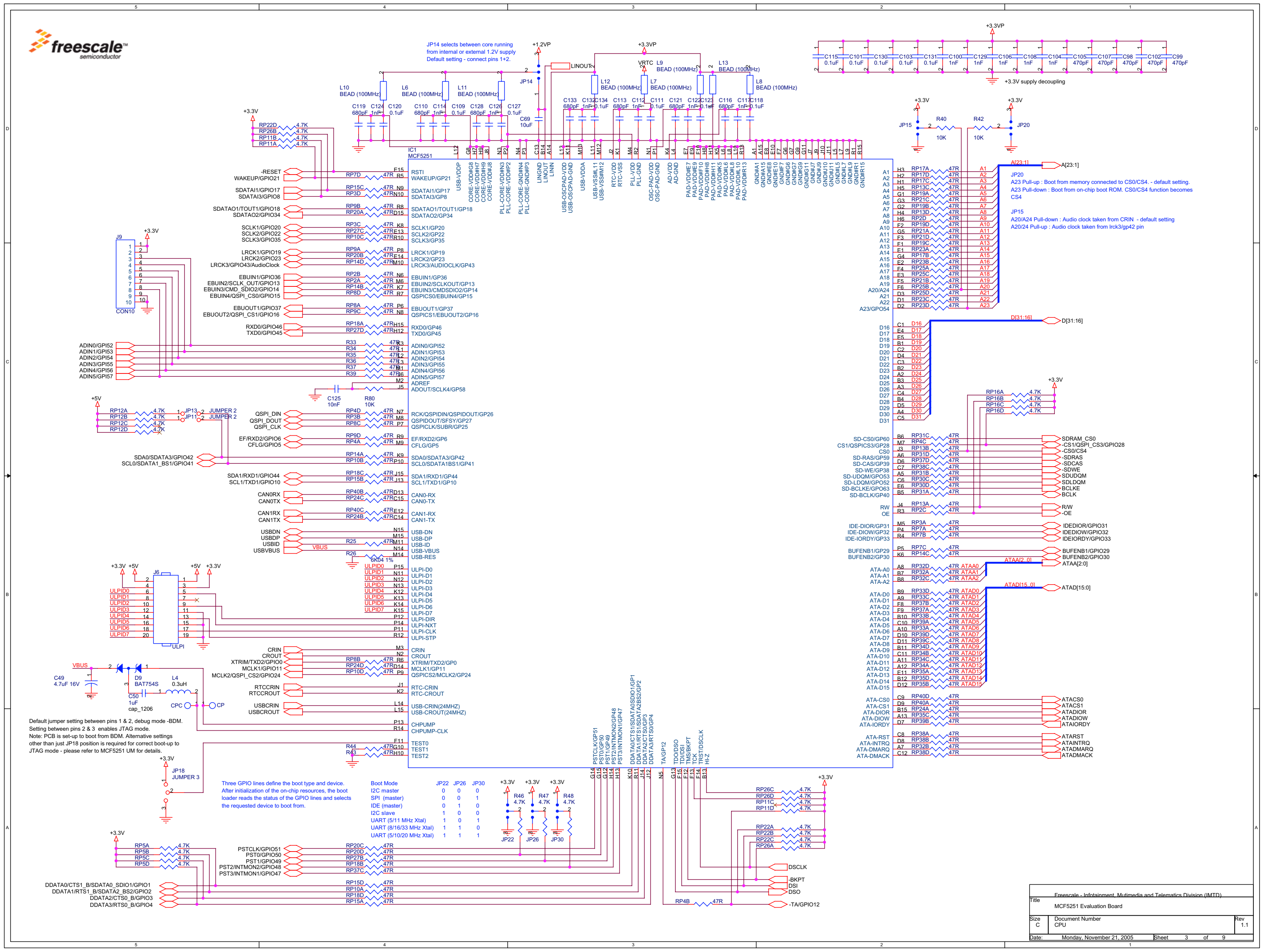

freescale semiconductor
JP14 selects between core running from internal or external 1.2V supply Default setting - connect pins 1+2.
+1.2VP
+3.3VP
+3.3V supply decoupling
IC1 MCF5251
JP20 A23 Pull-up : Boot from memory connected to CS0/CS4. - default setting. A23 Pull-down : Boot from on-chip boot ROM. CS0/CS4 function becomes CS4
JP15 A20/A24 Pull-down : Audio clock taken from CRIN - default setting A20/24 Pull-up : Audio clock taken from lrck3/gp42 pin
A[23:1]
D[31:16]
ATAA[2..0]
ATAD[15..0]
Default jumper setting between pins 1 & 2, debug mode -BDM. Setting between pins 2 & 3 enables JTAG mode. Note: PCB is set-up to boot from BDM. Alternative settings other than just JP18 position is required for correct boot-up to JTAG mode - please refer to MCF5251 UM for details.
Three GPIO lines define the boot type and device. After initialization of the on-chip resources, the boot loader reads the status of the GPIO lines and selects the requested device to boot from.
Boot Mode JP22 JP26 JP30
I2C master 0 0 0
SPI (master) 0 0 1
IDE (master) 0 1 0
I2C slave 1 0 0
UART (5/11 MHz Xtal) 1 0 1
UART (8/16/33 MHz Xtal) 1 1 0
UART (5/10/20 MHz Xtal) 1 1 1
Freescale - Infotainment, Mutimedia and Telematics Division (IMTD)
Title MCF5251 Evaluation Board
Size C
Document Number CPU
Rev 1.1
Date: Monday, November 21, 2005
Sheet 3 of 9

Figure 3-4 MCF5251 Evaluation Board

Figure 3-5 shows the Debug and Test Points.

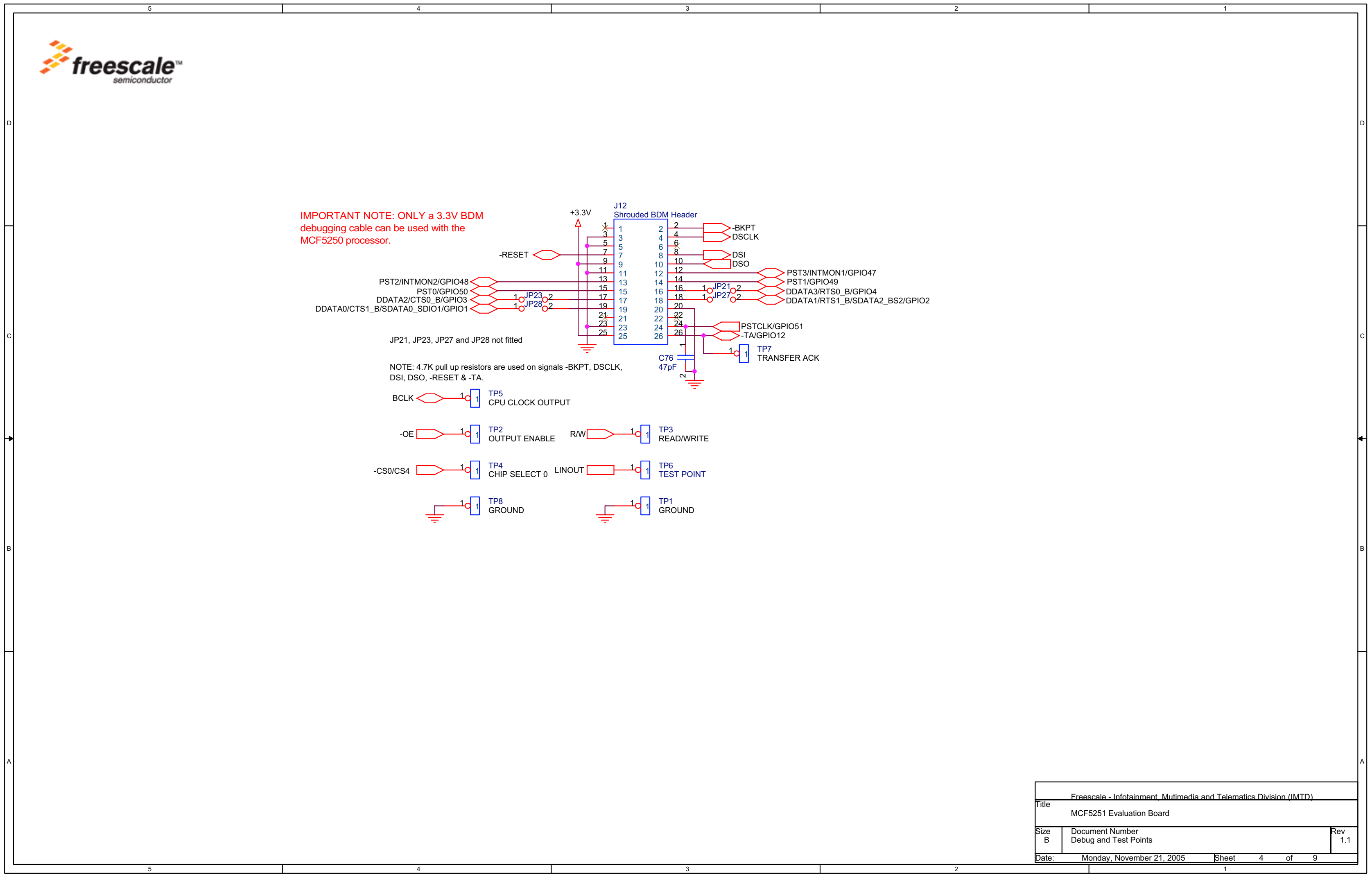


Figure 3-5 Debug and Test Points

Figure 3-6 shows the IDE, ATA, and Wireless.

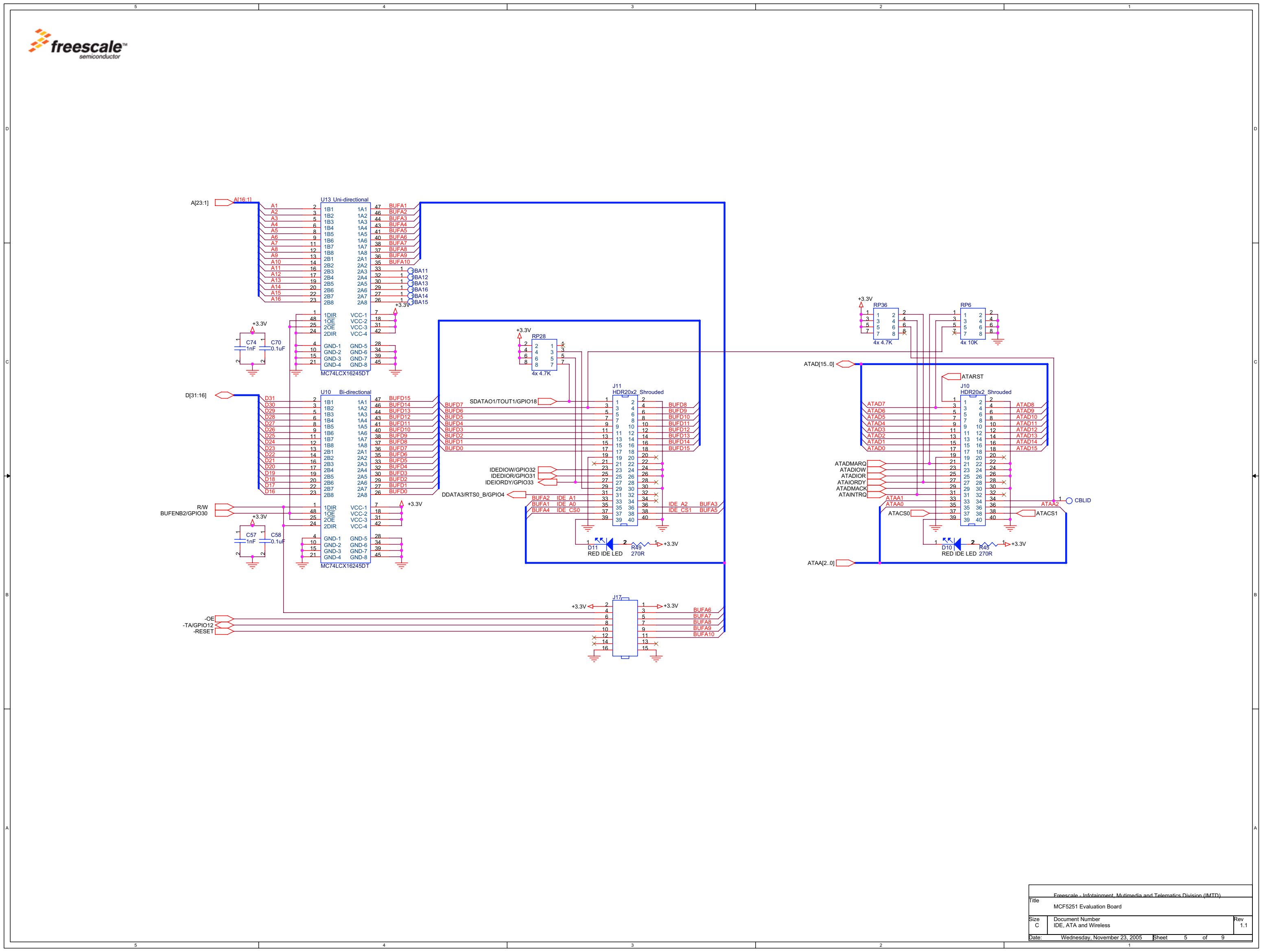


Figure 3-6 IDE, ATA, and Wireless

Figure 3-7 shows Memory.

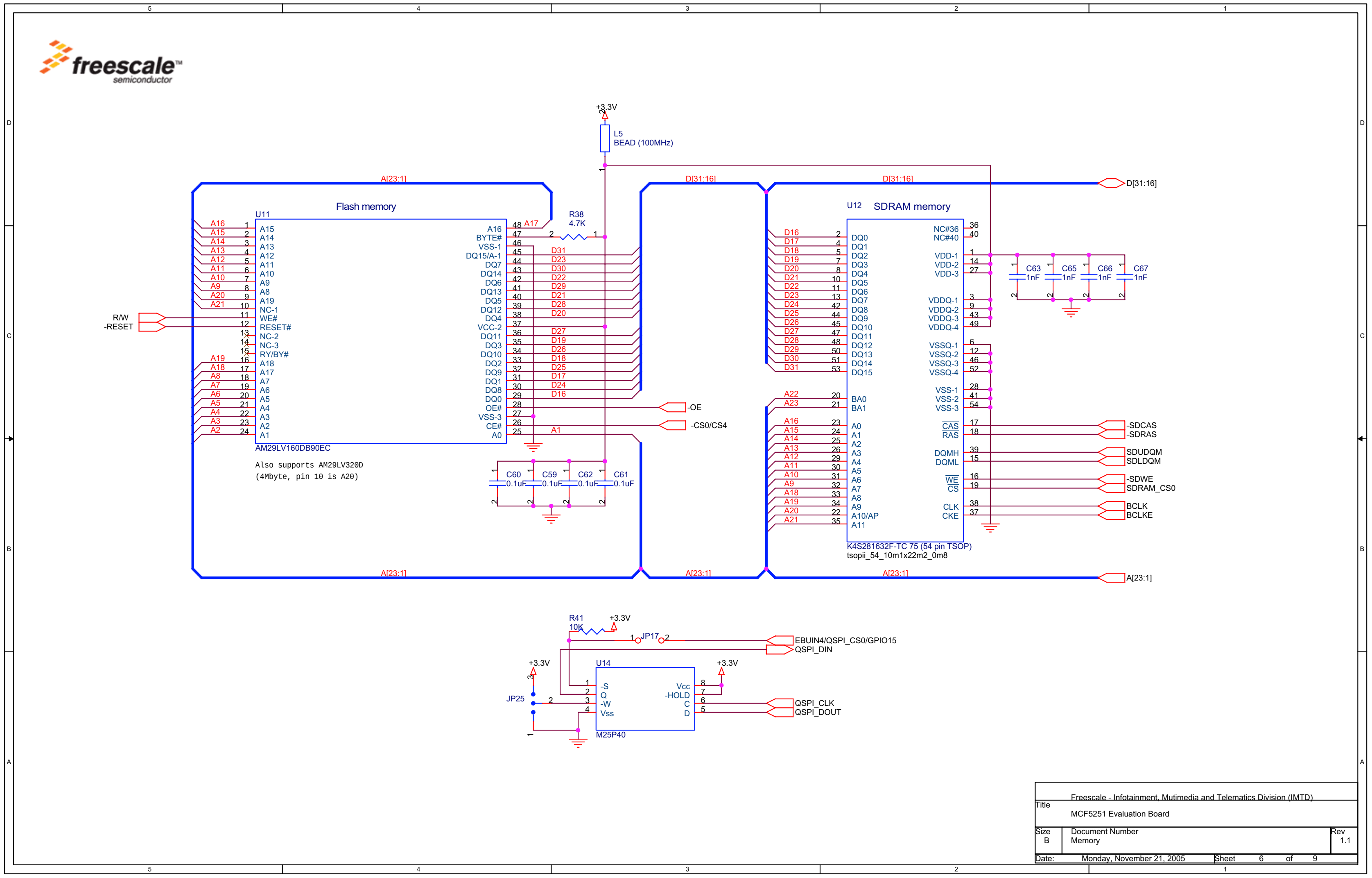


Figure 3-7 Memory

Figure 3-8 shows Power Supply.

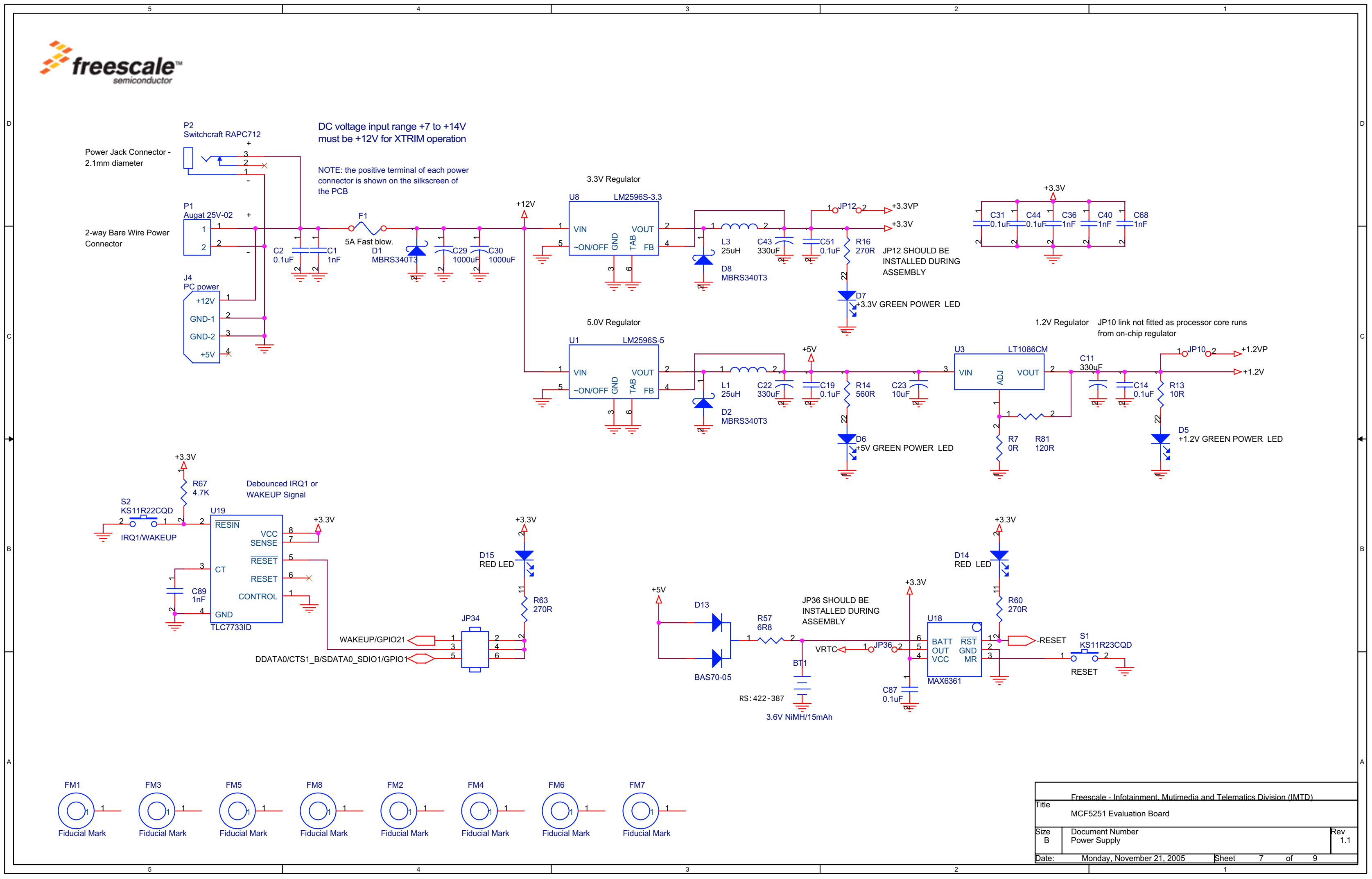


Figure 3-8 Power Supply

Figure 3-9 shows Serial Interfaces and Clocks.

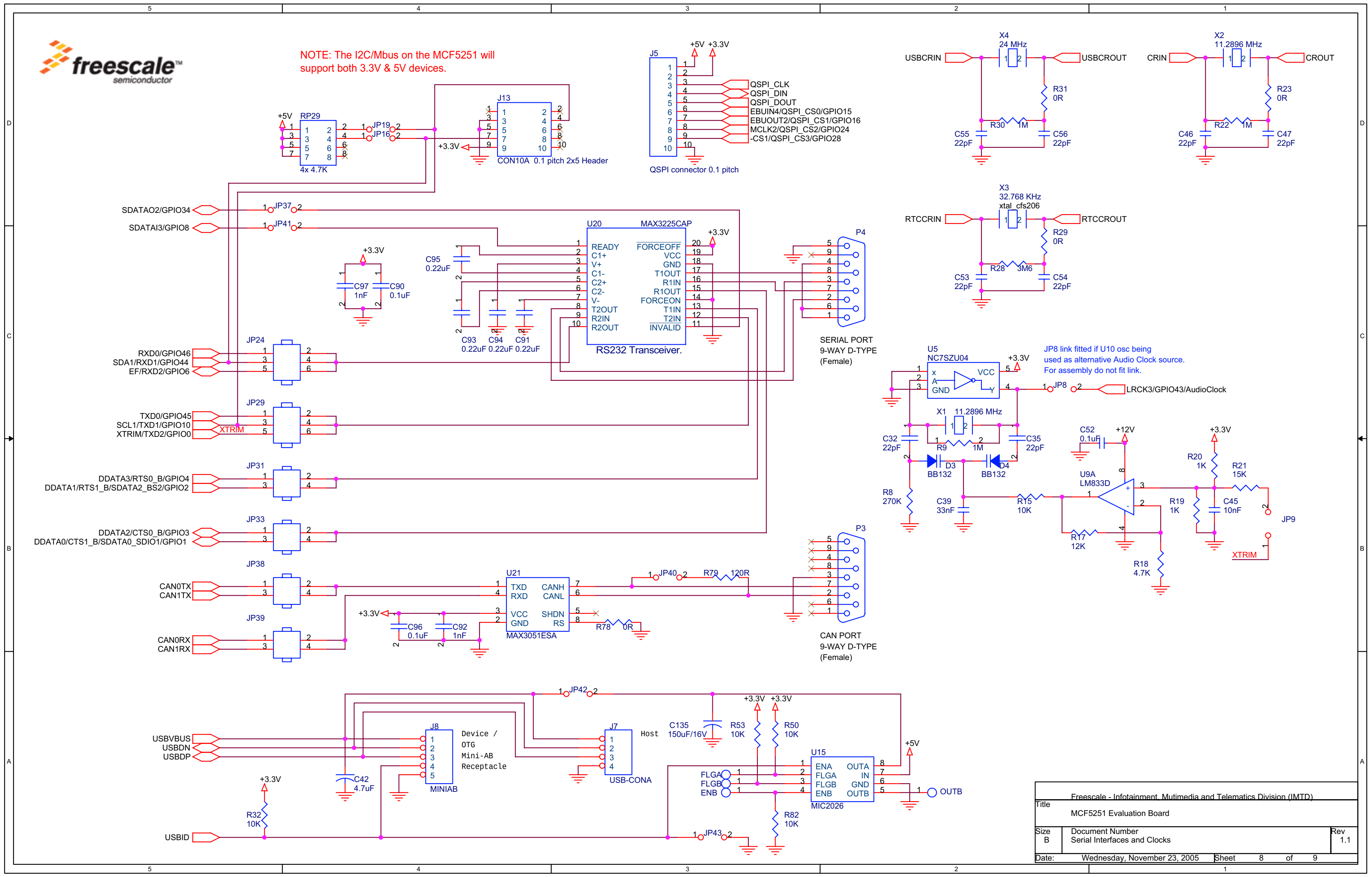


Figure 3-9 Serial Interfaces and Clocks

Figure 3-10 shows Trio Peripherals.

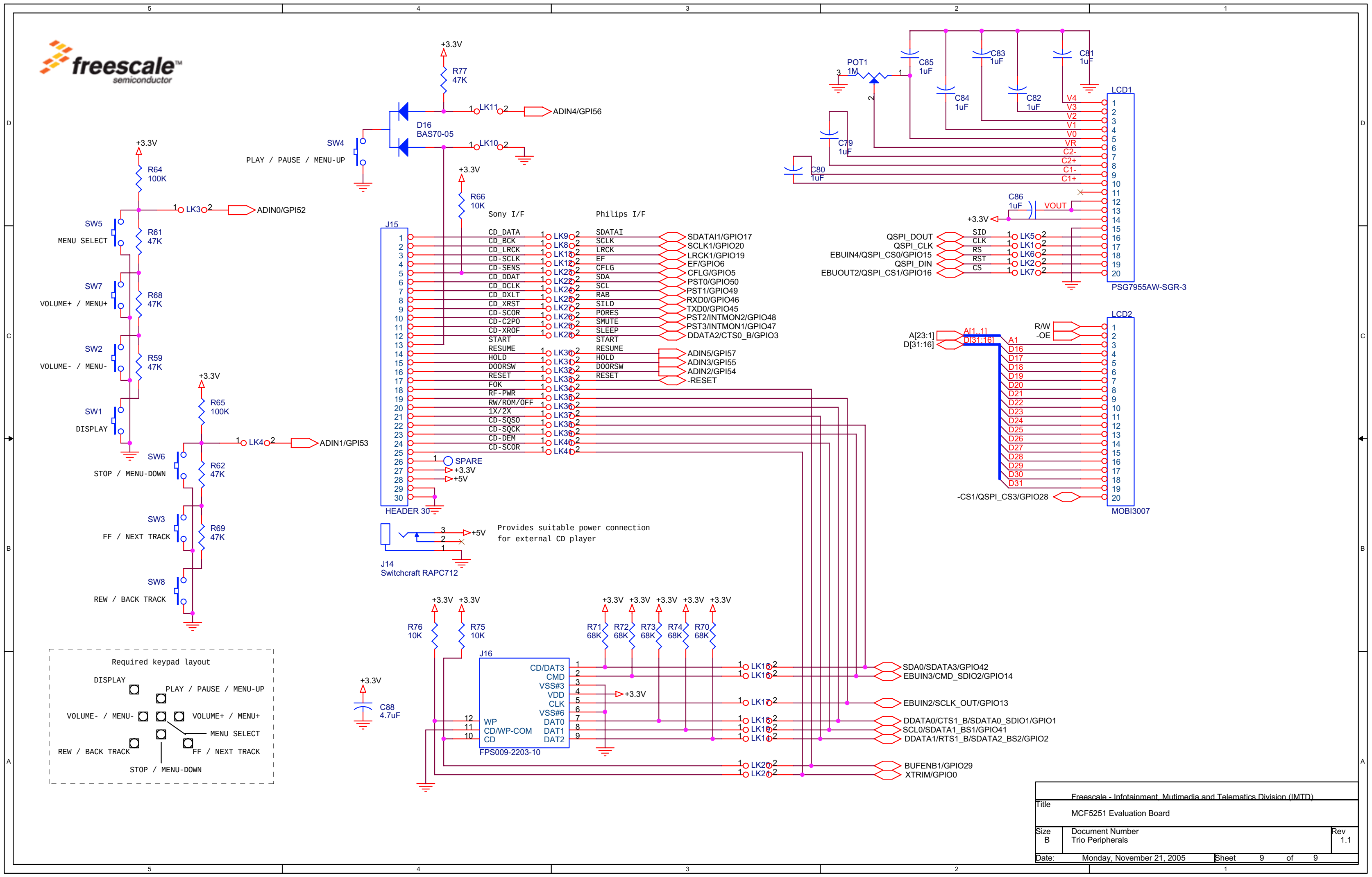


Figure 3-10 Trio Peripherals

Appendix B
Evaluation Board BOM

Table B-1 lists the bill of materials.

Table B-1 M5251C3 BOM

Item	Qty	Reference	Part	Function
1	1	ANT1	392-8093	2.45GHz_WLAN_Antenna
2	0	BA11 BA12 BA13 BA14 BA15 BA16 BT ENA ENB FLGA FLGB OUTB SPARE WLAN	PTH	
3	1	BT1	524-864	3.6V NiMH/15mAh
4	27	C1 C19 C20 C22 C30 C73 C74 C75 C76 C83 C84 C85 C95 C101 C111 C113 C115 C135 C142 C144 C146 C149 C152 C155 C160 C161 C171	1nF	Cap, 1nF,NPO or COG, 25v, 0603,5%
5	40	C2 C3 C5 C9 C15 C16 C18 C69 C70 C71 C72 C81 C82 C88 C91 C94 C98 C102 C108 C112 C114 C124 C125 C126 C127 C130 C132 C136 C138 C140 C141 C147 C150 C153 C156 C159 C162 C164 C165 C170	0.1uF	Cap, 0.1uf, 50v,X7R, 0603,10%
6	9	C4 C10 C86 C121 C122 C133 C139 C172 C173	10uF	Cap, 10uF, 16v, SMT, Elect, 4mm, 4.3mm base
7	12	C6 C8 C157 C158 C174 C175 C176 C177 C178 C179 C180 C181	1uF	Cap. 1 uF35v, SMT, Electrolytics
8	5	C11 C12 C87 C90 C97	AVX TPSE337K10CLR	Cap, Tan, 330UF, D, 10V, Low ESR
9	6	C13 C14 C99 C100 C103 C104	0.22uF	Cap, 0.22uF, 0603, 16V
10	5	C23 C24 C25 C26 C143	470pF	Cap, 470pF,NPO or COG 0603, 10%
11	7	C31 C129 C131 C145 C148 C151 C154	680pF	Cap, 680pF,NPO or COG 0603, 10%
12	2	C39 C137	47pF	Cap, 47pf, 0805, 50V, NPO or COG
13	8	C92 C93 C105 C106 C166 C167 C168 C169	22pF	Cap, 22PF, NPO, 0603
14	2	C96	ECA-1VM102 or UVZ1H102MHH	Cap, 1000uf, rad. T.H.
15	2	C107 C134	10nF	Cap, 10nF,25v,X7R,0603,10%
16	1	C109	06035C333KAT2A	CAP CER .033UF 10% 50V X7R 0603
17	6	C116 C118 C119 C120 C163 C182	C1608X5R0J475M	CAP CER 4.7UF 6.3V X5R 20% 0603

Table B-1 M5251C3 BOM (Continued)

Item	Qty	Reference	Part	Function
18	2	C117 C123	C1608C0G1H222J	Cap, 2.2nF,NPO or COG, 25v, 0603,5%
19	5	D5 D9 D10 D27 D28	AA3528EC	LED, 1206, SMT. Red
20	3	D6 D7 D8	MBRS340T3	
21	3	D11 D12 D22	AA3528SGC	LED, 1206, SMT. Green
22	2	D23 D25	BB132	
23	1	D26	BAT754S	
24	2	D29 D30	BAS70-05	
25	1	F1	4527K + 0216005.H	Fuse Holder + 5a, 250v, 2 x 20 mm fuse
26	1	IC4	MCF5251	
27	1	J1	MJ158L	STEREO JACK SOCKET SW 3.5mm
28	1	J2	636-2627OR 2526-6002UB	Shrouded BDM Header
29	0	J6		QSPI connector 0.1 pitch
30	2	J8 J16	636-4027 or 2540-6002UB	HDR20x2_Shrouded
31	2	J9 J10	MR-551L	RCA PHONO JACK
32	1	J11	350211-1	PC power
33	1	J13	S1011-10	CON10
34	1	J14	S2011-05	CON10A 0.1 pitch 2x5 Header
35	1	J15	S1011-15	CON15
36	1	J17	KMAB-SMT-5S-S-30TR	MINIAB
37	1	J18	87520-0010B	USB-CONA
38	1	J19	76342-315	HEADER 30
39	2	J20 P1	RAPC712	
40	1	J21	FPS009-2203-10	
41	12	JP1 JP4 JP7 JP50 JP79 JP80 JP84 JP91 JP92 JP93 JP98 JP100	S2105-03 or MTMM-103-04-GS-148	JUMPER 3
42	22	JP5 JP6 JP8 JP9 JP13 JP14 JP41 JP42 JP43 JP53 JP68 JP69 JP70 JP71 JP72 JP73 JP94 JP95 JP96 JP97 JP101 JP107	S2105-02 or MTMM-102-04-GS-148	JUMPER 2
43	2	JP102 JP103	PRPN032PAEN	JUMPER 3X2
44	4	JP104 JP105 JP106 JP108	PRPN022PAEN	JUMPER 2X2

Table B-1 M5251C3 BOM (Continued)

Item	Qty	Reference	Part	Function
45	2	L1 L2	B82111-B-C24	
46	9	L3 L4 L7 L8 L9 L10 L11 L12 L13	BLM21A601S	
47	1	L14	1210-331K	0.3uH or 0.33uh, 1210
48	1	LCD1(socket)	M22-6112022 or MMS-120-01-L-SV	Socket for LCD1, LDC1 is PSG7955AW-SGR-3
49	1	LCD2(socket	9-188275-0	Socket for LCD2, LCD2 is MOBI3007
50	0	LK1 LK2 LK3 LK4 LK5 LK6 LK7 LK8 LK9 LK10 LK11 LK12 LK13 LK14 LK15 LK16 LK17 LK18 LK19 LK20 LK21 LK22 LK23 LK24 LK25 LK26 LK27 LK28 LK29 LK30 LK31 LK32 LK33 LK34 LK35 LK36 LK37 LK38 LK39 LK40 LK41	LINK	
51	1	M1	Wi-Fi Module APM6125	
52	1	P2	2SV-02 or 571-14376711	Conn., 2 pin, thru hole
53	2	P3 P4	748875-1 or747844-3	DB9 RS232 PORT THRU HOLE DB9
54	1	POT1	3214W-1-105E	TRIMMER, SMD 5 TURN 1M; RoHS Compliant:TBA; Resistance:100kR; Power rating:0.25W;
55	3	R101 R128 R151	1M	
56	3	R1 R2 R118	10R	
57	11	R31 R43 R44 R100 R131 R158 R163 R167 R168 R176 R177	10K	
58	29	RP14 RP15 RP16 RP17 RP18 RP19 RP20 RP22 RP23 RP25 RP26 RP27 RP28 RP29 RP30 RP32 RP33 RP34 RP38 RP39 RP40 RP45 RP46 RP47 RP48 RP49 RP50 RP51 RP52	4x 47R	
59	9	R37 R38 R39 R40 R41 R42 R141 R174 R175	47R	
60	9	R45 R46 R47 R95 R122 R135 R143 R145 R146	4.7K	
61	2	R58 R162	100K	
62	6	R64 R68 R161 R164 R165 R166	47K	

Table B-1 M5251C3 BOM (Continued)

Item	Qty	Reference	Part	Function
63	5	R97 R102 R104 R105 R144	270R	
64	5	R98 R123 R126 R152 R155	0R	
65	1	R106	560R	
66	3	R107 R108 R140	470R	
67	2	R119 R120	18R	
68	1	R129	15K	
69	3	R130 R139 R147	1K	
70	1	R132	12K	
71	1	R138	270K	
72	2	R142 R150	?	
73	0	R148 R149	DNF	
74	1	R153	3M6	
75	1	R154	120R	
76	2	R156 R157	22R	
77	5	R169 R170 R171 R172 R173	68K	
78	10	RP2 RP5 RP41 RP43 RP44 RP6 RP7 RP8 RP9 RP53	4x 4.7K	
79	1	RP54	4x 10K	
80	1	S1	KS11R23CQD	
81	1	S2	KS11R22CQD	
82	1	SMA1	1053378-1 or 901-143-6RFX	SMA_connector
83	8	SW1 SW2 SW3 SW4 SW5 SW6 SW7 SW8	SDTX-620N	
84	0	TP1 TP2 TP3 TP4 TP5 TP6 TP7 TP8	TEST POINT	
85	1	U1	AK4366VT	
86	1	U15	LM833D	
87	1	U6	AM29LV160DB90EC	
88	1	U7	K4S641633D-G (4Mx16) 52PBGA (4x13)	
89	1	U8	LT1086CM	
90	0	U9	MAX6361LUT29-T	
91	1	U10	NC7SZU04	
92	1	U11	LM2596S-3.3	IC, TO263, 3A, 150mhz
93	1	U12	LM2596S-5	IC, TO263

Table B-1 M5251C3 BOM (Continued)

Item	Qty	Reference	Part	Function
94	1	U13	MAX3225CAP	IC, 16 PIN TSSOP
95	1	U14	M25P40-VMN6T	IC SRL FLASH 4MBIT 3.6V 8-SOIC
96	2	U16 U17	MC74LCX16245DT	IC, 48 PIN, TSSOP,3.3V, 16 BIT TRANSCEIVER
97	1	U18	TLC7733ID	IC, 8 PIN, SOIC
98	1	U19	AK5353VT	IC, 16 PIN,TSSOP
99	0	U20	GP1FA550RZ REPLACEMENT PART GP1FA553RZ	GP1FA550RZ REPLACEMENT PART GP1FA553RZ
100	0	U21	GP1FA550TZ Replacement GP1FA551TZ, 425-1101-5-ND	GP1FA550TZ Replacement GP1FA551TZ, 425-1101-5-ND
101	1	U22	SN74ALVCH374PWR	IC, 20 Pin TSSOP
102	1	U23	CY2300SC	IC, 8 PIN, SOIC
103	0	U24	MAX3051ESA	
104	1	U25	MIC2026	
105	2	X1 X2	ABL-11.2896MHz	Xtal, 11.2896 MHz
106	1	X3	HC49US24.000MABJ	Xtal, 24 MHz, HC49U
107	1	X4	C-2 32.0000K-P	Xtal, 32 KHz, 2 dip T.H. cyl.