NetBurner Mod5213 Development Kit

User's Manual

Revision 1.2
July 19, 2007
Released

Table of Contents

1. Introduction

Thank you for purchasing the NetBurner Mod5213 Development Kit. Welcome to the NetBurner family. NetBurner is your single source for hardware, software, development kits, tools, technical support, and custom design services. NetBurner solutions allow you to reduce risk and improve functionality with a complete proven design.

Whether you want to design your own hardware, or are looking for a standard off-the-shelf solution, NetBurner provides the software, hardware, and tools to get your product to market in the shortest possible time. NetBurner offers a full line of services from board level designs and hourly consulting to complete turnkey systems. NetBurner also offers a Royalty-Free License option. Please contact our **Sales** Department for more information on any of these options.

The software included in your Mod5213 Development Kit is licensed to run only on NetBurner provided hardware. If your application involves manufacturing your own hardware, please contact our **Sales** Department for details on a royalty free software license.

Please ensure that your NetBurner Mod5213 Development Kit is registered by going to NetBurner **Support** now to set up your account. Registration is quick and easy. The registration data stored on NetBurner's server will **not** be sold, exchanged, or knowingly released to third parties **without** prior written permission from the individuals affected. You also get **90 days** of **free** E-Mail Support (and Software upgrades) with the purchase of your Mod5213 Development Kit.

The NetBurner Mod5213 Development Kit includes all the tools necessary to complete your embedded design; including the NBEclipse IDE with an integrated debugger, a fully ANSI compliant C/C++ Compiler and Linker, and a Real Time Operating System (RTOS).

1.1. Additional Documentation

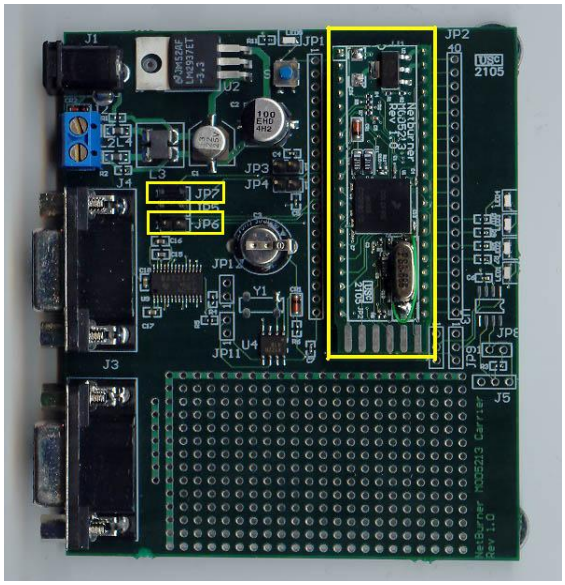
- A **Documentation Overview** can be found in **C:\Nburn\docs**
- For additional **programming** information, please refer to your **Mod5213 Programmers Guide** located (by default) in **C:\Nburn\docs**
- The **Netburner uCOS Library** is located (by default) in **C:\Nburn\docs**
- The **NetBurner RTC**, **CAN**, and **I²C** Libraries are all documented in the **NetBurner Runtime Libraries Manual**, located (by default) in **C:\Nburn\docs**.
- An **NBEclipse Getting Started Guide** is located (by default) in **C:\Nburn\docs**. For NBEclipse specific help, please refer to your **NBEclipse User's Manual**. From the **Help** pull down menu (**in NBEclipse**) select **Help Contents**.
- All **Freescale Manuals** are located (by default) in **C:\Nburn\docs**
- All **GNU Manuals** are located (by default) in **C:\Nburn\docs**
- All **License** information is located (by default) in **C:\Nburn\docs**

Warning: The only NetBurner example applications that will work with your Mod5213 Development Kit are located (by default) in C:\Nburn\examples\MOD5213.

2. Hardware Features

- ColdFire® V2 Core (63 MIPS @ 66 MHz)
- Physical Dimensions: 1.9" x 0.6"
- PCB Dimensions: 2.3" x 0.7"
- Industry Standard 40 Pin DIP
- Temperature range: -40º C to +85º C
- MAC Module and HW Divide
- Low-power optimization
- 32 KB SRAM and 256 KB FLASH
- CAN 2.0B controller with 16 message buffers
- Three UARTs with DMA capability
- Queued serial peripheral interface (QSPI)
- Inter-integrated circuit (I²C) bus controller
- Four 32-bit timer channels with DMA capability
- Four 16-bit timer channels with capture/compare/PWM
- 4-channel 16-bit/8-channel 8-bit PWM generator
- Two periodic interrupt timers (PITs)
- 4-channel DMA controller
- 8-channel 12-bit ADC
- Up to 33 general-purpose I/O
- System integration (PLL, SW watchdog)
- Power Requirements - Configurable Maximum of 120mA
- DC Input Voltage
  - **o** Unregulated: 4V - 7V input to an integrated 3.3V regulator on pin 40
  - **o** Regulated: 3.3V input on the VDD (pin 39)
- 3.3V I/O (**Warning: The Mod5213 is not 5V tolerant**)

The picture below shows the Mod5213 seated (between JP1 and JP2) correctly in the NetBurner Mod5213 Mod-Dev 40 Carrier board.



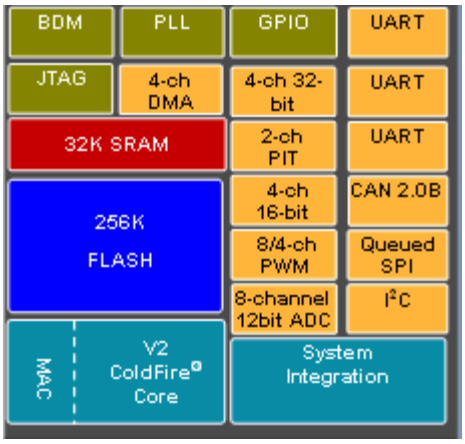
Warning: Your Mod5213 will not work (i.e. you will have no serial communication) unless JP6 and JP7 (shown above) are jumpered on your Carrier board.

3. Memory Map

Address	Type	Size*	Description
0x20000000 to 0x200003FF	SRAM	1K	Vector Base Register
0x20000400 to 0x20007FFF	SRAM	31K	Application Space
0x20008000 to 0x3FFFFFFF	-------	-------	Unused
0x40000000 to 0x401CFFFF	Processor Registers	-------	Processor SIM Registers
0x401D0000 to 0xFFBFFFFF	-------	-------	Unused
0xFFC00000 to 0xFFC03FFF	FLASH	16K	Monitor
0xFFC04000 to 0xFFC3EFFF	FLASH	244K	Application Code
0xFFC3F000 to 0xFFC3FFFF	FLASH	4K	User Parameter Storage

*** The size is in 8-bit bytes.**

4. MCF5213 Block Diagram



5. Directory Structure

The NetBurner Mod5213 installation creates a number of directories located under the root **C:\Nburn** directory. These directories are listed in the table below along with a brief description.

Directory	Description
\bin	The directory that contains all compiled application files.
\docs	All NetBurner documentation is located in this directory.
\examples\MOD5213	The source code for the NetBurner Mod5213 example applications. **Warning: All other example applications (in the C:\Nburn\example directory) will not work with the Mod5213 platform**
\gcc-m68k	The egcs/gnu compiler executables and Libraries.
\lib	The NetBurner system Libraries and linker scripts.
\make	The make file fragments that implement the NetBurner make environment.
\pcbin	The executable files for the NetBurner generated (non-GNU) tools.
\pctools	Source code for the NetBurner generated (non-GNU) tools.
\include_nn	Source code for the NetBurner API Libraries.
\system_nn	Source code for the NetBurner System Libraries.
\Mod5213	Source code for the Mod5213 System and API Libraries.

Note: To find out what **version** of the NNDK tool set you are using, navigate to your **C:\Nburn** directory, and open up (e.g. in notepad) the **release_tag** file.

6. The Development Process

The NetBurner environment is setup to run normal C and C++ programs. It includes all of the standard C Library functions. As the programmer, you can choose to do as much or as little system initialization as desired. If your code includes a function named 'main', you have complete control. However, if you want the NetBurner library to initialize the timer and serial port, then use as your starting function:

```
void UserMain( void * pd )
```

With this method, the main function is provided by the main.c file in the \Nburn\system directory. All of the examples in the \Nburn\examples directory begin with the UserMain( ) function. This is the method most commonly used by developers.

7. SerialLoad

Synopsis:

`serialload.exe`

Description:

SerialLoad will download a new code image (i.e. a _APP.s19 file) to your Mod5213 module. You do **not** have to use MTTTY. Therefore, if you are out in the field, you can use the SerialLoad application (in **C:\Nburn\pcbin**) to download a working application to your Mod5213 module. **Important:** The **current** application in your Mod5213 **must** have SerialLoad enabled in order to use this feature.

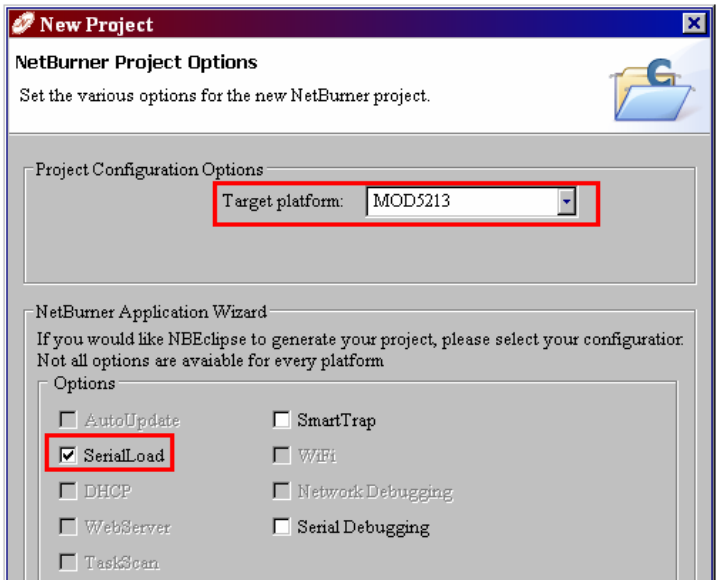
Warning: Your Mod5213 will not work (i.e. you will have no serial communication) unless JP6 and JP7 on your Carrier Board are jumpered.

See Also:

- EnableSerialUpdate --- Enables serial updates (**Section 11.1**)
- MTTTY --- In your NetBurner Tools Manual (in **C:\Nburn\docs**)

7.1. SerialLoad and NBEclipse

If you are using NBEclipse, your application (once successfully compiled) will be **automatically** downloaded to your Mod5213 (**if the current application in your Mod5213 supports SerialLoad downloads**). No further action is required. If you are using **NBEclipse**, you can include SerialLoad **automatically** when you create your project (as shown below). **Note:** The Mod5213 Factory Application **supports** SerialLoad downloads.



8. General Functions

8.1. ForceReboot

Required Header File

```
#include <bsp.h>          // Found in C:\Nburn\include_nn
```

Synopsis:

```
void ForceReboot( );
```

Description:

This function will reboot your Mod5213.

Parameters:

None

Return Value:

Nothing --- This is a void function

8.2. GetUserParameters

Required Header File:

```
#include <system.h>       // Found in C:\Nburn\include_nn
```

Synopsis:

```
void *GetUserParameters( );
```

Description:

This function returns a pointer to the user parameter area. This area is intended for storage of **non-volatile** configuration parameters. **Note:** The type and format of the stored data is **entirely** up to the individual developer. The system stores this as a **blob**, and provides **no** protection from uninitialized data. The developer **needs** to add some **uninitialized** data protection to his stored structure.

Parameters:

None

Returns:

A read only pointer to the user parameter area

8.3. SaveUserParameters

Required Header File:

```
#include <system.h>        // Found in C:\Nburn\include_nn
```

Synopsis:

```
int SaveUserParameters( void * pCopyFrom, int len );
```

Description:

This function stores up to **4K** of **arbitrary data** in the user configuration space. **Note:** The type and format of the stored data is **entirely** up to the individual developer. The system stores this as a **blob**, and provides **no** protection from uninitialized data. The developer **needs** to add some **uninitialized** data protection to his stored structure.

Parameters:

Type	Name	Description
void	***pCopyFrom	A pointer to the data to store.
int	len	The length of the data to store.

Returns:

A non zero number --- If the user parameter region was programmed successfully
0 (zero) --- For failure

9. Analog-to-Digital Converter (ADC)

The Mod5213 has two separate 12-bit A/D converters, each with their own sample and hold circuit. Each A/D converter has 4 multiplexed analog inputs providing 8 channels of analog input.

The Mod5213 API supports automatic continuous sampling of all 8 input channels, and a function to read the last sampled value for a particular analog input channel.

The **Mod5213 A/D example** application is located in **C:\Nburn\examples\MOD5213\a2d**.

Note: If you need precise interrupt driven sampling and control; you **will** need to create an A/D driver specific to your application.

The MCF5213 Manual (in **C:\Nburn\docs\FreescaleManuals**) is a good reference on how to configure the Mod5213 A/D system to meet your application requirements.

Required Header File:

```
#include <a2d.h>          // Found in C:\Nburn\MOD5213\include
```

9.1. EnableAD

Synopsis:

```
void EnableAD( BYTE clock_div );

void EnableAD( BYTE clock_div=7 );   // Default divider value is 7
```

Description:

This is a very simple A/D driver. It simply sets up the selected A/D inputs, and samples all of the A/D inputs over and over. The A/D sample rate is 33177600 / (clock_div*6).

If all 8 A/D's (i.e. 0 to 7) are running, then it is divided further by 8. Therefore, for the default value of 7, 33177600 / (7*6*8) = 98,742 samples per second.

Note: In the default mode, **no** interrupts are generated. If you want precise interrupt generated A/D samples, it is best that you refer to the A/D Converter section in the MCF5213 Manual (in **C:\Nburn\docs\FreescaleManuals**).

Returns:

Nothing ---This is a void function

9.2. ReadA2DResult

Synopsis:

```
WORD ReadA2DResult( int ch );
```

Description:

This function reads the A/D results. **Note:** The scale factor is 0 to 32767, with the bottom 3 bits **always** zero.

Parameter:

Type	Name	Description
int	ch	The A/D channel number to read (0 to 7).

Returns:

The number of A/D counts

10. Pins and GPIO

Required Header File:

```
#include <pinconstant.h>    // Found in C:\Nburn\MOD5213\include
```

The Pin Class definition file is located in **C:\Nburn\MOD5213\include\pinconstant.h**. The GPIO example application (SimpleGPIO) is located in C:\Nburn\examples\MOD5213.

Important: Before you use a pin, you **must** set it up.

Warning: Pins 1, 9, 10, 19, 20, 39, and 40 cannot be used for GPIO.

Note: There is **no** error checking, because it would **slow** performance. For example, if you wrote code and tried to use these pins (i.e. 1, 9, 10, 19, 20, 39, and/or 40) for GPIO, **no** error will be generated, and the pin will **not** be changed. This lack of error checking makes the (correctly) requested pins operate faster.

The descriptions (functions) in section 9.2 represent the actual definitions that can be used to configure each pin if you choose to use the NetBurner Pin Class For additional information, please read your MCF5213 User's Manual (in **C:\Nburn\docs\FreescaleManuals**).

10.1. Function

Required Header File:

```
#include <pins.h>          // Found in C:\Nburn\include_nn
```

Synopsis:

```
void Function( int ft );
```

Description:

This function sets the pin up for use.

Parameter:

Type	Name	Description
int	ft	Function type - determines what function the selected pin will have.

Returns:

Nothing ---This is a void function

Examples:

1. To set pin 4 to UART TX:

```
Pins[4].function( pin4_UART2_TX );
```

2. To set pin 5 to GPIO:

```
Pins[5].Function( pin5_GPIO );
```

3. To set pin 5 high:

```
Pins[5]=1;
```

4. To set pin 5 low:

```
Pins[5]=0;
```

5. To read the value of pin 5 into a variable:

```
BOOL b; // Declare the variable
b=Pins[5];
```

6. To set pin 5 to high impedance:

```
Pins[5].hiz();
```

7. To make pin 5 go from high impedance back to being driven, while keeping the last set value:

```
Pins[5].drive();
```

10.2. Signal Description and Pinout

Connector Pin No.	Primary Function	Secondary Function	Tertiary Function	GPIO Port	Processor Pin No.
1	RSTI			0	A3
2	URXD0			1	D1
3	UTXD0			1	D2
4	SDA	CANRX	URXD2	1	E2
5	SCL	CANTX	UTXD2	1	E1
6	IRQ1	SYNCA	PWM1	1	C6
7	IRQ4			1	C5
8	IRQ7			1	C4
9	VDDA			0	H8
10	VRH			0	J8
11	AN2			1	G6
12	AN1			1	H6
13	AN0			1	J6
14	AN3			1	G7
15	AN7			1	H9
16	AN6			1	G9
17	AN5			1	G8
18	AN4			1	F9
19	VSSA/VRL			0	H7
20	VSS			0	J1
21	DTIN3	DTOUT3	PWM6	1	H3
22	DTIN2	DTOUT2	PWM4	1	J3
23	DTIN1	DTOUT1	PWM2	1	G4
24	DTIN0	DTOUT0	PWM0	1	H4
25	GPT3		PWM7	1	D8
26	GPT2		PWM5	1	D9
27	GPT1		PWM3	1	E9
28	GPT0		PWM1	1	F7
29	URXD1			1	B2
30	UTXD1			1	A2
31	UCTS1	SYNCA	URXD2	1	C3
32	URTS1	SYNCB	UTXD2	1	B1
33	QSPI_CS2			1	F2
34	QSPI_CS1			1	H2
35	QSPI_CS0	SDA	CTS1	1	H1
36	QSPI_DOUT	CANTX	TXD1	1	G1
37	QSPI_DIN	CANRX	RXD1	1	F3
38	QSPI_CLK	SCL	RTS1	1	G2
39	VDD			0	E3
40	Unregulated Power			0	N/A
Connector Pin No.	**Primary Function**	**Secondary Function**	**Tertiary Function**	**GPIO Port**	**Processor Pin No.**

11. Serial Library

Required Header Files

```
#include <SerialUpdate.h>   // Found in C:\Nburn\include_nn
#include <Serial.h>         // Found in C:\Nburn\include_nn
```

Functions

- **EnableSerialUpdate** --- Enables Serial updates
- **InitUart** --- Opens up a specific Serial port
- **SimpleUart (MACRO)** --- Opens up a specific Serial port
- **close** --- Closes (the opened) Serial port
- **assign_stdio** --- Assigns the Serial port as stdio
- **assign_sterr** --- Assigns the Serial port as sterr
- **create_file** --- Creates a read/write file
- **charavail** --- Makes a char available for reading
- **sgetchar** --- Gets a char from the Serial port
- **writechar** --- Writes a char to the Serial port
- **writestring** --- Writes a string to the Serial port

11.1. EnableSerialUpdate

Synopsis:

```
void EnableSerialUpdate();
```

Description:

This function enables serial updates.

Parameters:

None

Returns:

Nothing ---This is a void function

Example Code:

```
#include <stdio.h>
#include <ctype.h>
#include <basictypes.h>
#include <serialirq.h>
#include <system.h>
#include <constants.h>
#include <ucos.h>
#include <serialupdate.h>       // Enables Serial Updates

extern "C"
 {
   void UserMain( void * pd );
 }

const char * AppName="My Mod5213 Application";

 void UserMain( void * pd )
  {
    OSChangePrio( MAIN_PRIO );
    EnableSerialUpdate();       // Enables Serial Updates
    SimpleUart( 0, SystemBaud );
    assign_stdio( 0 );
    iprintf( "Application started\r\n" );
    while ( 1 )
      {
        OSTimeDly( TICKS_PER_SECOND );
      }
  }
```

11.2. InitUart

Synopsis:

```
int InitUart( int portnum, unsigned int baudrate, int stop_bits, int
data_bits, parity_mode parity );
```

Description:

This function opens the Serial UART for use.

- If you have included **serialpoll.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **InitPolledUart**.
- If you have included **serialirq.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **InitIRQUart**.

Important: When selecting Port number two (**UART2**) for initialization, **pins 31 and 32** of the Mod5213 become used for **URXD2 and UTXD2**, respectively.

Parameters:

Type	Name	Description
int	portnum	Determines which Serial port will be opened. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
unsigned int	baudrate	Specifies the baudrate (1200 to 115200).
int	stop_bits	Specifies the stop bits. **Note:** This value must be either 1 or 2.
int	data_bits	Specifies the data bits. **Note:** This value must be either 7 or 8.
parity_mode	parity	Specifies one of the four parity modes for the Serial port**:** eParityNone, eParityOdd, eParityEven, or eParityMulti.

Returns:

0 --- If successful
-1 --- If the port does not exist (SERIAL_ERR_NOSUCH_PORT)
-2 --- If the port is not opened (SERIAL_ERR_PORT_NOTOPEN)
-3 --- If the port is already opened (SERIAL_ERR_PORT_ALREADYOPEN)
-4 --- If the parameters are not correct (SERIAL_ERR_PARAM_ERROR)

11.3. SimpleUart (MACRO)

Synopsis:

```
int SimpleUart( int portnum, unsigned int baudrate );
```

Description:

This MACRO opens the **Serial UART** for use. If you prefer to **specify** the **port** and **baud rate**, you can use this MACRO.

Warning: If you use this MACRO, the stop_bits will be set to 1, the data_bits set to 8, and the parity set to none (i.e. the factory default values).

- If you have included **serialpoll.h** (found in **C:\Nburn\include_nn**) in your application code, this MACRO actually calls **SimpleInitPolledUart**.
- If you have included **serialirq.h** (found in **C:\Nburn\include_nn**) in your application code, this MACRO actually calls **SimpleInitIRQUart**.

Parameters:

Type	Name	Description
int	portnum	Determines which Serial port will be opened. The port number **must** be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
unsigned int	baudrate	Specifies the baudrate.

Returns:

0 --- If successful
-1 --- If the port does not exist (SERIAL_ERR_NOSUCH_PORT)
-2 --- If the port is not opened (SERIAL_ERR_PORT_NOTOPEN)
-3 --- If the port is already opened (SERIAL_ERR_PORT_ALREADYOPEN)
-4 --- If the parameters are not correct (SERIAL_ERR_PARAM_ERROR)

11.4. Close

Synopsis:

```
void close( int portnum );
```

Description:

This function closes the specified (open) Serial port.

- If you have included **serialpoll.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **polled_close**.
- If you have included **serialirq.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **IRQ_close**.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) Serial port will be closed. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

Nothing ---This is a void function

11.5. assign_stdio

Required Header File:

```
#include <Serial.h>
```

Synopsis:

```
void assign_stdio( int portnum );
```

Description:

This function assigns a specified Serial port as stdio. This function allows you to use standard I/O with the selected Serial port (e.g. printf, scanf etc.).

- If you have included **serialpoll.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **polled_assign_stdio**.
- If you have included **serialirq.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **IRQ_assign_stdio**.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) Serial port will be assigned as stdio. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

Nothing ---This is a void function

Example:

```
assign_stdio( 0 );    // Assigns Serial port 0 as stdio
```

11.6. assign_sterr

Required Header File:

```
#include <Serial.h>
```

Synopsis:

```
void assign_sterr( int portnum );
```

Description:

This function assigns a specified Serial port as sterr. This function allows you to use standard I/O with the selected Serial port (e.g. printf, scanf etc.).

- If you have included **serialpoll.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **polled_assign_sterr**.
- If you have included **serialirq.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **IRQ_assign_sterr**.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) Serial port will be assigned as sterr. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

Nothing ---This is a void function

Example:

```
assign_sterr( 0 );    // Assigns Serial port 0 as sterr
```

11.7. create_file

Required Header File:

```
#include <Serial.h>
```

Synopsis:

```
FILE * create_file( int portnum );
```

Description:

This function creates a pointer of type FILE that you can use to read/write (with fprintf fscanf etc.) to the selected serial port with functions that take file pointers as parameters (e.g. fprintf(), fscanf(), etc.).

- If you have included **serialpoll.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **polled_create_file**.
- If you have included **serialirq.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **IRQ_create_file**.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

A pointer of type FILE

11.8. charavail

Required Header File:

```
#include <Serial.h>
```

Synopsis:

```
BOOL charavail( int portnum );
```

Description:

Use this function to make a char available for reading on the specified Serial port.

- If you have included **serialpoll.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **polled_charavail**.
- If you have included **serialirq.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **IRQ_charavail**.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

True --- If a char is available for reading

11.9. sgetchar

Required Header File:

```
#include <Serial.h>
```

Synopsis:

```
char sgetchar( int portnum );
```

Description:

This function gets a char from the specified Serial port. **Note**: This function **will** also block until a char is available to be read.

- If you have included **serialpoll.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **polled_getchar**.
    - The **polled** version of this function does **not** yield to the OS, so **no** lower priority task can run.
- If you have included **serialirq.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **IRQ_getchar**.
    - The **IRQ** version **will** yield to the OS until a char is available.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

The char

11.10. writechar

Required Header File:

```
#include <Serial.h>
```

Synopsis:

```
void writechar( int portnum, char c);
```

Description:

This function writes a character to the specified Serial port.

- If you have included **serialpoll.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **polled_writechar**.
- If you have included **serialirq.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **IRQ_writechar**.

Note: Both of these functions **will** block until at least **one** character can be written.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
char	c	The character that will be written to the Serial port.

Returns:

Nothing ---This is a void function

11.11. writestring

Required Header File:

```
#include <Serial.h>
```

Synopsis:

```
void writestring( int portnum, const char * s );
```

Description:

This function writes a string to the specified Serial port.

- If you have included **serialpoll.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **polled_writestring**.
- If you have included **serialirq.h** (found in **C:\Nburn\include_nn**) in your application code, this function actually calls **IRQ_writestring**.

Parameters:

Type	Name	Description
int	portnum	Determines which (open) Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
const char	*s	The string that will be written to the Serial port

Returns:

Nothing ---This is a void function

12. Polled Serial Driver Library

The following functions operate the Serial port in polled mode. This means "reads" do **not** block, and "writes" will **not** return until **all** the chars are in the hardware output buffers. This is accomplished by polling a status bit in the UART registers. This also means that you can **loose** incoming chars, if you do **not** read often enough. **Note:** Flow control with the polled UART is entirely up to the Programmer.

The advantage of **polling** is that it takes up **less SRAM** resources than an IRQ driven scheme because the Serial I/O is **not** buffered.

Warning: You cannot include both #include <Serialirq.h> and #include <Serialpoll.h> in your application. If you do, you will get errors when you try to compile your application.

Required Header File

```
#include <Serialpoll.h>    // Found in C:\Nburn\include_nn
```

Functions

- **InitPolledUart** --- Opens the polled Serial port
- **SimpleInitPolledUart (MACRO)** --- Opens the polled Serial port
- **polled_close** --- Closes (the opened) polled Serial port
- **polled_assign_stdio** --- Assigns the polled Serial port as stdio
- **polled_assign_sterr** --- Assigns the polled Serial port as sterr
- **polled_create_file** --- Creates a read/write file
- **polled_charavail** --- Makes a char available for reading
- **polled_getchar** --- Gets a char from the polled Serial port
- **polled_writechar** --- Writes a char to the polled Serial port
- **polled_writestring** --- Writes a string to the polled Serial port

12.1. InitPolledUart

Synopsis:

```
int InitPolledUart( int portnum, unsigned int baudrate, int stop_bits,
int data_bits, parity_mode parity );
```

Description:

This function opens the specified Serial UART for use in polled mode.

Parameters:

Type	Name	Description
int	portnum	Determines which polled Serial port will be opened. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
unsigned int	baudrate	Specifies the baudrate.
int	stop_bits	Specifies the stop bits. **Note:** This value must be either 1 or 2.
int	data_bits	Specifies the data bits. **Note:** This value must be either 5, 6, 7, or 8.
parity_mode	parity	Specifies one of the four parity modes for the Serial port**:** eParityNone, eParityOdd, eParityEven, or eParityMulti.

Returns:

 0 --- If successful
-1 --- If the port does not exist (SERIAL_ERR_NOSUCH_PORT)
-2 --- If the port is not opened (SERIAL_ERR_PORT_NOTOPEN)
-3 --- If the port is already opened (SERIAL_ERR_PORT_ALREADYOPEN)
-4 --- If the parameters are not correct (SERIAL_ERR_PARAM_ERROR)

12.2. SimpleInitPolledUart (MACRO)

Synopsis:

```
SimpleInitPolledUart( port, baud )
```

Description:

If you prefer to **specify** the **port** and **baud rate**, you can use this MACRO to open the specified **Serial UART** for use in **polled mode**.

Warning: If you use this MACRO, the stop_bits will be set to 1, the data_bits set to 8, and the parity set to none (i.e. the factory default values).

Parameters:

Name	Description
port	Determines which polled Serial port will be opened. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
baud	Specifies the baudrate.

12.3. polled_close

Synopsis:

```
void polled_close( int portnum ) ;
```

Description:

This function closes the specified (open) polled Serial port.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) polled Serial port will be closed. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

Nothing ---This is a void function

12.4. polled_assign_stdio

Synopsis:

```
void polled_assign_stdio( int portnum );
```

Description:

This function assigns a specified polled Serial port as stdio. This function allows you to use standard I/O with the selected open Serial port (e.g. printf, scanf etc.).

Parameter:

Type	Name	Description
int	portnum	Determines which (open) polled Serial port will be assigned as stdio. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

Nothing ---This is a void function

Example:

```
polled_assign_stdio( 0 );   // Assigns polled Serial port 0 as stdio
```

12.5. polled_assign_sterr

Synopsis:

```
void polled_assign_sterr( int portnum );
```

Description:

This function assigns a specified polled Serial port as stderr. This function allows you to use standard error I/O with the selected open Serial port (e.g. printf(stderr,..., fscanf(stderr,... etc.).

Parameter:

Type	Name	Description
int	portnum	Determines which (open) polled Serial port will be assigned as sterr. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

Nothing --- This is a void function

Example:

```
Polled_assign_sterr( 0 );    // Assigns polled Serial port 0 as sterr
```

12.6. polled_create_file

Synopsis:

```
FILE * polled_create_file( int portnum );
```

Description:

This function create a FILE that you can use to read/write (with fprintf, fscanf etc.).

Parameter:

Type	Name	Description
int	portnum	Determines which (open) polled Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

A pointer of type FILE

12.7. polled_charavail

Synopsis:

```
BOOL polled_charavail( int portnum );
```

Description:

This function makes a char available for reading on the specified polled Serial port.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) polled Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

True --- If a char is available for reading

12.8. polled_getchar

Synopsis:

```
char polled_getchar( int portnum );
```

Description:

This function gets a char from the specified polled Serial port. **Important:** This function **will** also block until a char is available. **Note:** This function does **not** yield to the OS, so no lower priority task can run.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) polled Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

The char

12.9. polled_writechar

Synopsis:

```
void polled_writechar( int portnum, char c );
```

Description:

This function writes a char to the specified polled Serial port.

Parameters:

Type	Name	Description
int	portnum	Determines which (open) polled Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
char	c	The char that will be written to the opened polled Serial port.

Returns:

Nothing ---This is a void function

12.10. polled_writestring

Synopsis:

```
void polled_writestring( int portnum, const char * s );
```

Description:

This function writes a string to the specified polled Serial port.

Parameters:

Type	Name	Description
int	portnum	Determines which (open) polled Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
const char	*s	The string that will be written to the polled Serial port.

Returns:

Nothing --- This is a void function.

13. Interrupt Driven Serial Driver Library

Interrupt driven means that the Serial I/O is buffered, therefore, your application does **not** have to wait for the actual I/O to occur. This also means that your application will **not** miss any incoming characters because it is busy elsewhere. The Serial I/O buffer sizes for the Mod5213 are located in **C:\Nburn\include_nn\ constants.h**. Unless you are constrained on space, interrupt driven **Serial I/O is recommended**.

Warning: You cannot include both #include <Serialpoll.h> and #include <Serialirq.h> in your application. If you do, you will get errors when you try to compile your application.

Required Header File

```
#include <Serialirq.h>   // Found in C:\Nburn\include_nn
```

Functions

- **InitIRQUart** --- Opens the Serial port in interrupt driven mode
- **SimpleInitIRQUart (MACRO)** --- Opens the Serial port in interrupt driven mode
- **IRQ_Close** --- Closes (the opened) interrupt driven Serial port
- **IRQ_TX_XON_Flow** --- Enables or disables the TX flow using received xon/xoff
- **IRQ_RX_XON_Flow** --- Enables or disables via xon/xoff to protect the RX buffer
- **IRQ_assign_stdio** --- Assigns the interrupt driven Serial port as stdio
- **IRQ_assign_sterr** --- Assigns the interrupt driven Serial port as sterr
- **IRQ_create_file** --- Creates a file
- **IRQ_charavail** --- Makes a char available for reading
- **IRQ_getchar** --- Gets a char from the interrupt driven Serial port
- **IRQ_writechar** --- Writes a char to the interrupt driven Serial port
- **IRQ_writestring** --- Writes a string to the interrupt driven Serial port

13.1. InitIRQUart

Synopsis:

```
int InitIRQUart( int portnum, unsigned int baudrate, int stop_bits, int
data_bits, parity_mode parity );
```

Description:

This function opens the IRQ driven Serial UART for use. This function operates the serial port in interrupt driven mode. This means "reads" block, allowing other RTOS threads to run, and "writes" will return immediately.

You will **not** loose chars sent to the serial port **unless** the input buffers overflow. **Important:** If you have **enabled** flow control, then flow control **will** keep from overflowing buffers. The size of the receive buffer is set in **C:\Nburn\include_nn\constants.h**.

Parameters:

Type	Name	Description
int	portnum	Determines which IRQ driven Serial port will be opened. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
unsigned int	baudrate	Specifies the baudrate.
int	stop_bits	Specifies the stop bits. **Note:** This value must be either 1 or 2.
int	data_bits	Specifies the data bits. **Note:** This value must be either 5, 6, 7, or 8.
parity_mode	parity	Specifies one of the four parity modes for the Serial port**:** eParityNone, eParityOdd, eParityEven, or eParityMulti.

Returns:

 0 --- If successful
-1 --- If the port does not exist (SERIAL_ERR_NOSUCH_PORT)
-2 --- If the port is not opened (SERIAL_ERR_PORT_NOTOPEN)
-3 --- If the port is already opened (SERIAL_ERR_PORT_ALREADYOPEN)
-4 --- If the parameters are not correct (SERIAL_ERR_PARAM_ERROR)

13.2. SimpleInitIRQUart (MACRO)

Synopsis:

```
SimpleInitIRQUart( port, baud )
```

Description:

If you prefer to **specify** the **port** and **baud rate**, you can use this MACRO to open the IRQ driven **Serial UART** (**IRQ driven**) for use.

Warning: If you use this MACRO, the stop_bits will be set to 1, the data_bits set to 8, and the parity set to none (i.e. the factory defaults).

Parameters:

Name	Description
port	Determines which IRQ driven Serial port will be opened. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
baud	Specifies the baudrate.

13.3. IRQ_Close

Synopsis:

```
void IRQ_close( int portnum );
```

Description:

This function closes the specified IRQ driven open Serial port.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) IRQ driven Serial port will be closed. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

Nothing ---This is a void function

13.4. IRQ_TX_XON_Flow

Synopsis:

```
void IRQ_TX_XON_Flow( int portnum, BOOL enable );
```

Description:

This function enables or disables the TX flow using received xon/xoff.

Parameters:

Type	Name	Description
int	portnum	Determines which (open) IRQ driven Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
BOOL	enable	Do you want TX flow control enabled?

Returns:

Nothing --- This is a void function

13.5. IRQ_RX_XON_Flow

Synopsis:

```
void IRQ_RX_XON_Flow( int portnum, BOOL enable );
```

Description:

This function enables or disables via xon/xoff to protect the RX buffer.

Parameters:

Type	Name	Description
int	portnum	Determines which (open) IRQ driven Serial port will be used. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
BOOL	enable	Do you want xon/xoff enabled?

Returns:

Nothing ---This is a void function

13.6. IRQ_assign_stdio

Synopsis:

```
void IRQ_assign_stdio( int portnum );
```

Description:

This function assigns the specified IRQ driven specified Serial port as stdio. This function allows you to use standard I/O with the specified IRQ driven Serial port (e.g. printf, scanf etc.).

Parameter:

Type	Name	Description
int	portnum	Determines which (open) IRQ driven Serial port will be assigned as stdio. The port number must be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

Nothing ---This is a void function

Example:

```
IRQ_assign_stdio( 0 );   // Assigns IRQ driven Serial port 0 as stdio
```

13.7. IRQ_assign_sterr

Synopsis:

```
void IRQ_assign_sterr( int portnum );
```

Description:

This function assigns the specified IRQ driven specified Serial port as stderr. This function allows you to use standard error I/O with the specified IRQ driven Serial port (e.g. printf(stderr,..., fscanf(stderr,... etc.).

Parameter:

Type	Name	Description
int	portnum	Determines which (open) IRQ driven Serial port will be assigned as stderr. The port number **must** be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

Nothing ---This is a void function

Example:

```
IRQ_assign_sterr( 0 );     // Assigns IRQ driven Serial port 0 as sterr
```

13.8. IRQ_create_file

Synopsis:

```
FILE * IRQ_create_file( int portnum );
```

Description:

This function create a FILE that you can use to read/write (with fprintf, fscanf etc.).

Parameter:

Type	Name	Description
int	portnum	Determines which (open) IRQ driven Serial port will be used. The port number **must** be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

A pointer of type FILE

13.9. IRQ_charavail

Synopsis:

```
BOOL IRQ_charavail( int portnum );
```

Description:

This function makes a char available for reading on the specified IRQ driven Serial port.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) IRQ driven Serial port will be used. The port number **must** be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

True ---If a char is available for reading

13.10. IRQ_getchar

Synopsis:

```
char IRQ_getchar( int portnum );
```

Description:

This function gets a char from the specified IRQ driven Serial port. **Note:** This function **will** also block **until** a char is available. This function **will** yield to the OS **until** a char is available.

Parameter:

Type	Name	Description
int	portnum	Determines which (open) IRQ driven Serial port will be used. The port number **must** be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.

Returns:

The char

13.11. IRQ_writechar

Synopsis:

```
void IRQ_writechar( int portnum, char c );
```

Description:

This function writes a char to the specified IRQ driven Serial port.

Parameters:

Type	Name	Description
int	portnum	Determines which (open) IRQ driven Serial port will be used. The port number **must** be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
char	c	The char that will be written to the opened IRQ driven Serial port.

Returns:

Nothing ---This is a void function

13.12. IRQ_writestring

Synopsis:

```
void IRQ_writestring( int portnum, const char * s );
```

Description:

This function writes a string to the specified IRQ driven Serial port.

Parameters:

Type	Name	Description
int	portnum	Determines which (open) IRQ driven Serial port will be used. The port number **must** be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
const char	*s	The string that will be written to the opened IRQ driven Serial port.

Returns:

Nothing --- This is a void function.

14. Serial Debugging

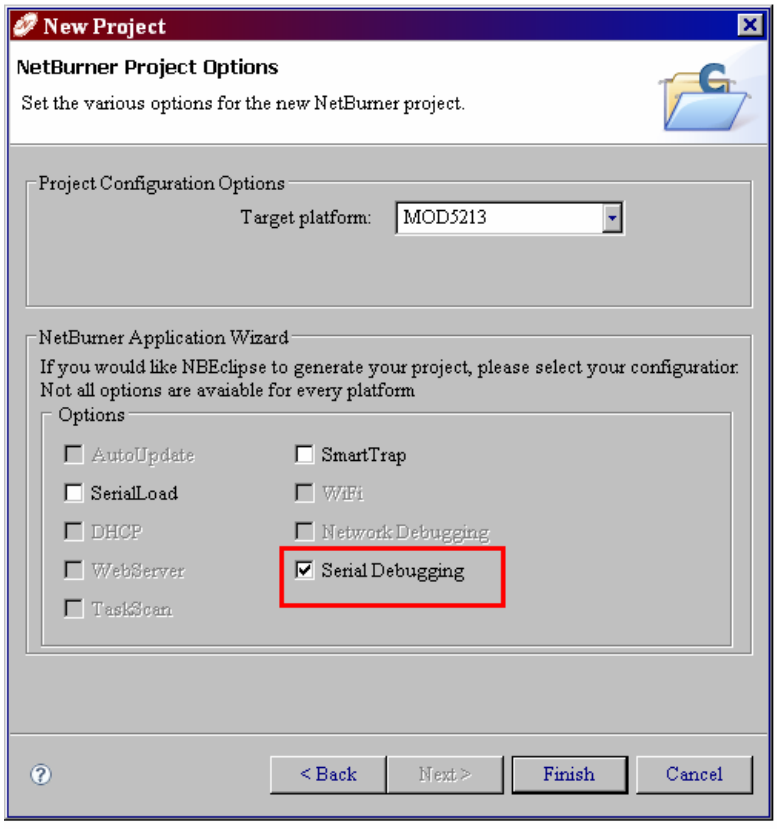
14.1. Introduction

The serial debugging stub is configured to use the second serial port with a level 7 nonmaskable interrupt. This level of control allows the debugger to debug all application code, even the operating system interrupt routines. **Warning: The only code you cannot debug is code within a level 7 interrupt.**

The serial stub uses the GDB remote serial protocol to communicate with the host computer over one of its serial ports. This allows the host computer to stop and start execution, set **breakpoints** (**you are limited to three**), and examine memory. For the curious, the details of this protocol are all covered in the GDB remote serial protocol documentation on Red Hat's web site.

14.2. Setting up your Debug Application

Before attempting these steps, you should become familiar with making and downloading a file to your Mod5213. **Important:** You **must** build your program **without** optimization, so that the debugger can figure out what is going on. In addition, you **must** change your source code to include and start the debugger. **Note**: If you are using **NBEclipse**, you can **include Serial Debugging** automatically when you create your project (as shown below). Please read your **NBEclipse Getting Started Guide** (in **C:\Nburn\docs**) for step-by-step instructions.



14.3. InitGDBStub

Required Header File:

```
#include <gdbstub.h>       // Found in C:\Nburn\include_nn
```

Synopsis:

```
void InitGDBStub( int port, int baudrate );
```

Description:

This function will **initialize** the debugger. This function **will** cause the program to **stop** execution at the point when the stub is called **and wait** for the debugger to connect and provide instructions.

Note: The **recommended** debugger serial port **baud rate** is **57600**.

Parameters:

Type	Name	Description
int	port	Determines which port will be used. The port number **must** be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
int	baudrate	Specifies the baudrate of the opened port. **Note:** This value **must** be: 300, 600, 1200, 2400, 4800, 9600, 119200, 38400, 57600, or 115200

Returns:

Nothing --- This is a void function.

Example Application:

DebugDemo --- C:\Nburn\examples\MOD5213

Example Code:

```
#include <gdbstub.h>

*
  {
    /* somewhere in your start up code */
    /* this starts GDB on port 1 at 57600 baud */
    InitGDBStub( 1, 57600 );
  }
```

14.4. InitGDBStubNoBreak

Required Header File:

```
#include <gdbstub.h>       // Found in C:\Nburn\include_nn
```

Synopsis:

```
void InitGDBStubNoBreak( int port, int baudrate );
```

Description:

This function will initialize the debugger, but it **will** allow program execution to continue normally.

For debugging during your development process, we recommend using this function.

Using this function is a good way to attach the debugger to a program that is already running, to hunt down intermittent bugs.

Note: The **recommended** debugger serial port **baud rate** is **57600**.

Parameters:

Type	Name	Description
int	port	Determines which port will be used. The port number **must** be 0, 1, or 2. **Note:** Port 0 is the primary Serial port.
int	baudrate	Specifies the baudrate of the opened port. **Note:** This value **must** be: 300, 600, 1200, 2400, 4800, 9600, 119200, 38400, 57600, or 115200

Returns:

Nothing --- This is a void function

Example Code:

```
#include <gdbstub.h>

*
  {
    /* somewhere in your start up code */
    /* this starts GDB on port 1 at 57600 baud */
    InitGDBStubNoBreak( 1, 57600 );
  }
```