NetBurner's Network Development Kit

MODL2E Hardware Manual

Revision 1.0
July 18, 2007
Released

Table of Contents

1. Introduction

NetBurner is your single source for hardware, software, development kits, tools, technical support, and custom design services. These elements are combined in a unique package that lets you concentrate on developing your product instead of reinventing network protocols and designing hardware.

NetBurner solutions also allow you to reduce risk and improve functionality with a complete proven design, including hardware, TCP/IP Stack, RTOS, and all necessary tools. NetBurner is indeed the fastest way to network enable your product.

Whether you want to design your own hardware, or are looking for a standard off-the-shelf network 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.

Please ensure that your NetBurner MODL2E Development Kit is registered by going to our **Support** site 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.

Your MODL2E Development Kit includes all the tools necessary to complete your embedded design - Real Time Operating System (RTOS), Fully ANSI Compliant C/C++ Compiler, and Linker, NBEclipse IDE with a fully integrated graphical debugger, end user device configuration, deployment tools, and so much more.

The MODL2E network enables serial devices right out of the box. No programming or development is required. The MODL2E is pre-programmed to convert TTL data to Ethernet, enabling communication with the serial device over a network or the Internet. The onboard web server provides easy device configuration using a standard web browser.

The resident boot monitor enables code development from the moment the NetBurner tools are installed on your host computer.

Traditionally, companies using 8 and 16-bit platforms find it nearly impossible to run resource-intensive applications. The NetBurner MODL2E Module features a full 32-bit architecture providing 63 MIPS @ 60 MHz. The MODL2E processor board is based on the Version 2 Coldfire core. Equipped with a full featured real time operating system, the MODL2E is ideal for any deterministic application.

The MODL2E also has a rich set of peripherals making it ideal for both communications and control applications. Real time capability, extended temperature operation, and low power features allow the MODL2E to operate even in the harshest of environments.

This easy-to-read User's Manual along with the enclosed Quick Start Guide, NBEclipse User's Manual, Programmer's Guide, and Hardware Documentation contain all the information you need to be Networking in 1 day. After reading these documents, you will be able to:

- Set up your NetBurner MODL2E hardware properly
- Create, Debug, and Download your own embedded applications in C/C++ using NetBurner's NBEclipse

2. Additional Documentation

- MODL2E (Hard Copy) Quick Start Guide
- All User's Manuals are located (by default) in C:\Nburn\docs
- A NetBurner's Programmer's Guide is located (by default) in C:\Nburn\docs
- An NBEclipse Getting Started Guide is located (by default) in C:\Nburn\docs
- All **Freescale** Manuals are located (by default) in C:\Nburn\docs
- All License Information is located (by default) in C:\Nburn\docs
- ALL GNU Information is located (by default) in C:\Nburn\docs

3. Hardware Features

- ColdFire 52234 32-bit processor
- 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
- One Serial Port
- 10/100 BaseT with RJ-45 connector
- CAN 2.0B controller with 16 message buffers
- One TTL
- 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: Unregulated: 4V - 7V input to an integrated 3.3V regulator on pin 40
- DC Input Voltage: Regulated: 3.3V input on the VDD (pin 39)
- 3.3V I/O (**Warning: The MODL2E is not 5V tolerant**)

4. Software Features

- Configuration through web browser, telnet, NetBurner's IPSetup utility, or SNMP (supports MIB-II).
- Password protection
- Firmware updates through network or serial connections
- Network Protocols Supported: ARP, DHCP, BOOTP, TCP, UDP, ICMP, Telnet, HTTP, and SNMP (**optional**)
- SSL Support (**optional**)

Please contact our **Sales** Department for more information.

5. Life Support Disclaimer

NetBurner's MODL2E is not authorized for use as a critical component in life support devices or systems, without the express written approval of NetBurner, Inc. prior to use. As used herein:

1. Life support devices or systems are devices or systems that (**a**) are intended for surgical implant into the body or (**b**) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

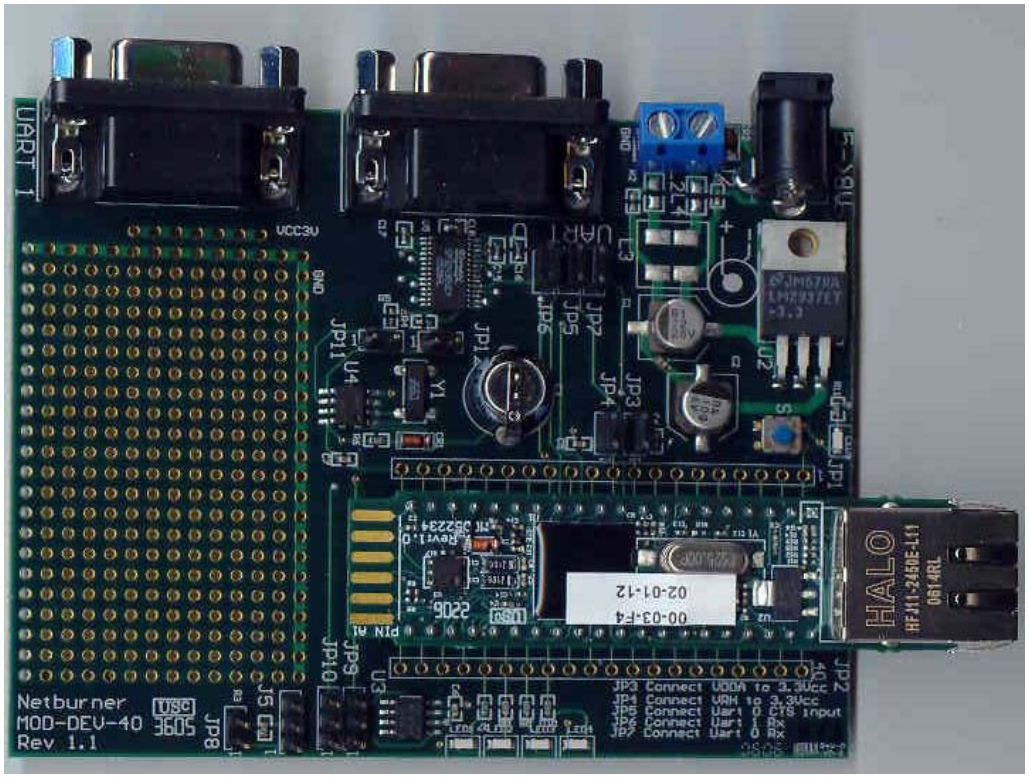
NetBurner, Inc. makes no representations or warranties with respect to the contents of this manual and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, NetBurner, Inc. reserves the right to revise this manual, make changes, and/or discontinue it without notice. Every effort has been made to ensure all information is correct, but NetBurner, Inc. is not responsible for inadvertent errors.

This manual contains links to third-party web sites that are not under the control of NetBurner, and NetBurner is not responsible for the content on any linked site. If you access any third party sites listed in this manual, then you do so at your own risk. NetBurner provides these links only as a convenience, and the inclusion of the link does not imply that NetBurner endorses or accepts any responsibility for the content on those third-party sites.

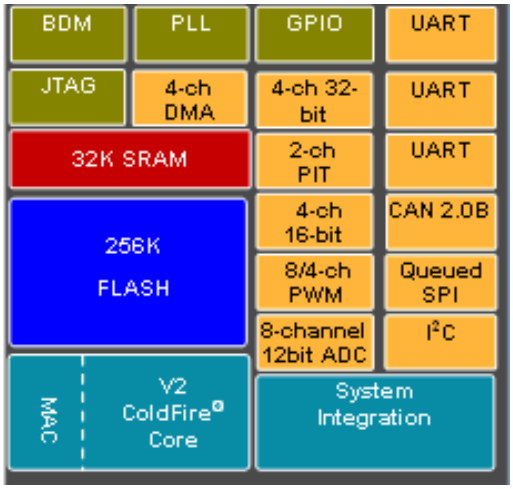
NetBurner, Inc.
5405 Morehouse Drive, Suite 200
San Diego, CA 92121 USA
NetBurner Sales
NetBurner Support
858-558-0293 (Telephone)
858-558-8549 (Fax)

6. Hardware Specifications

The ModL2E seated correctly in its Carrier board.



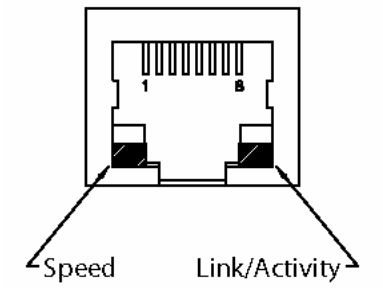
6.1. MCF52234 Block Diagram



6.2. RJ-45 Connector

6.2.1. LEDs

- LED 1: Speed: 10 MB (off) or 100 MB (on)
- LED 2: Link/Activity



6.2.2. Pinout Information

Pin	Signal	Pin	Signal
1	TX+	5	----
2	TX-	6	RX-
3	RX+	7	----
4	----	8	----

6.3. Signal Description and Pinout Information

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.**

6.4. Memory Map

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

Note: The size is in 8-bit bytes.

7. Directory Structure

The NetBurner MODL2E 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\ MODL2E	The source code for the NetBurner MODL2E example applications
\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
\MODL2E	Source code for the MODL2E System and API Libraries

8. 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_nn directory. All of the examples in the \Nburn\examples directory begin with the UserMain( ) function. This is the method most commonly used by developers.

9. Analog-to-Digital Converter

Introduction

The MODL2E 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 MODL2E 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.

If you need precise interrupt driven sampling and control, you will need to create an A/D driver specific to your application. The MCF52234 User's Manual is a good reference on how to configure the A/D system to meet your application requirements. **Note:** By default, all **Freescale** Manuals are located in **C:\Nburn\docs**.

Required Header File

```
#include <a2d.h>    // Found in Nburn\MODL2E\include
```

ADC Functions

- **EnableAD** --- Sets up the selected A/D inputs and samples them repeatedly
- **ReadA2DResult** --- Reads the A/D results

9.1. EnableAD

Synopsis:

```
void EnableAD( BYTE clock_div );

// default divider value is 7
void EnableAD( BYTE clock_div=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 repeatedly. 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.

Parameters

Type	Name	Description
Byte	clock_div	The default value of clock_div = 7. **Note:** In the default mode, **no** interrupts are generated.

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 (0 to 7) .

Returns:

The number of A/D counts

10. Pins and GPIO

Introduction

The Pin Class definition file (i.e. **pinconstant.h**) is located (by default) in **Nburn\MODL2E.**

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

Important: 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.

Required Header Files

```
#include <pinconstant.h>   // Found in Nburn\MODL2E\include
#include <pins.h>          // Found in Nburn\include_nn
```

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

Pins Function

- **Function** --- Sets the pin up for use

10.1. Function

Synopsis:

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

Description:

This function sets the pin up for use.

Parameter:

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

Returns:

Nothing --- This is a void function

Examples:

- To set pin 4 to UART TX:

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

- To set pin 5 to GPIO:

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

- To set pin 5 high (and low):

```
Pins[5]=1; ( Pins[5]=0; )
```

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

```
BOOL  b;  // Declare the variable.

b=Pins[5];
```

- To set pin 5 to high impedance:

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

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

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

11. Serial Library

Introduction

The NetBurner Serial Library consists of General, Polled, and Interrupt Driven Serial Functions.

Warning: You cannot include both polled and Interrupt driven functions in your application. If you do, you will get errors when you try to compile your application.

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 the 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 ModL2E 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.

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
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.

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 Serial port will be opened. 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

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

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

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

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

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

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

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

Introduction

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 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. 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

Introduction

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 are 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 Nburn\include_nn
```

Functions

- **InitIRQUart** --- Opens the Serial port in interrupt driven mode
- **SimpleInitIRQUart (MACRO)** --- Opens the Serial port
- **IRQ_Close** --- Closes (the opened) interrupt driven Serial port
- **IRQ_TX_XON_Flow** --- Enables/disables the TX flow using received xon/xoff
- **IRQ_RX_XON_Flow** --- Enables/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. **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:

Name	Type	Description
portnum	int	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.
c	char	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