Freescale Semiconductor
Application Note

Document Number: AN3906
Rev. 0, 06/2009

Serial-to-Ethernet Bridge Using MCF51CN Family and FreeRTOS

by: Paolo Alcantara
RTAC Americas
Mexico 2009

1 Overview

1.1 Purpose

This document describes a serial-to-Ethernet bridge using the MCF51CN128, the open-source RTOS FreeRTOS™ v5.3.0 and the TCP/IP stack LwIP v1.3.0. Serial interfaces used are UART and SPI. Ethernet connects using a well-known socket implementation.

1.2 Scope

This document was written to help you enable older serial interface-related designs for Ethernet connectivity. Microcontrollers usually connect through interfaces like SCI (or UART) and SPI. For this particular application, the microcontroller used is the MCF51CN128, which supports Ethernet-based connectivity.

To take advantage of this document, it is not mandatory to have in-depth Ethernet protocol knowledge. For custom modification, enough information is provided in this application note.

Contents

freescale™ semiconductor

1.3 Audience Description

This document can be used by software development engineers, test engineers, and anyone else who needs to use a serial-to-Ethernet bridge.

1.4 Problem Reporting Instructions

Issues and suggestions about this document (and associated software) should be reported through the support web page at www.freescale.com/support.

2 Introduction to Serial-to-Ethernet Bridge Hardware

You can easily add Ethernet (ETH) capabilities to your design using the serial-to-Ethernet bridge. This implementation includes a small memory footprint web server and TCP/IP socket-to-serial communication for a low-end Ethernet-based microcontroller solution. It does not require in-depth knowledge of Ethernet / TCP/IP.

The following block diagrams are examples of how to use it:

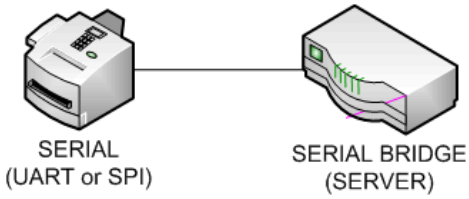

Figure 1. Basic Configuration: A Serial Device Connected to Ethernet Using Serial Bridge

Figure 2. A Serial Device Connected to Another Serial Device Using Client and Server Implementation Using Two Serial Bridges

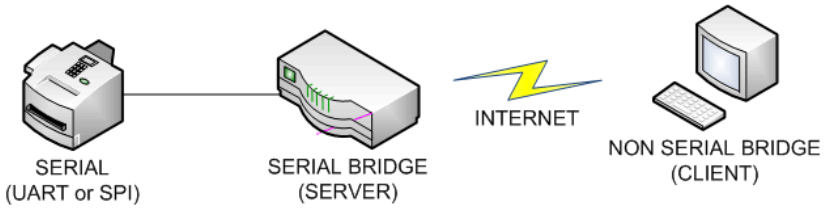

Figure 3. A Serial Device Connected to a Serial Bridge, which is Connected to the Internet, and a Non-Serial Bridge Receives and Transmits the Information

2.1 Hardware Implementation

The MCF51CN128 reference design board has the smallest size specially designed for customers looking for an end application solution. It's a low-cost board that comes with a schematic, layout files in CAD Allegro Editor, and Gerber files, provided for free. Figure 4 shows a block diagram of the hardware used for this serial bridge.

The MCF51CN128 reference design hardware is divided into three parts:

a) Minimal System — contains all the hardware needed for a minimal implementation with Ethernet functionality. Layout files can be copy-pasted to a new design for a customization in the use of it by just providing an unregulated power source from 3.7 V to 5.5 V to finish the design. This is the smallest system on the board and is delimited at 1.15" × 1.55".

It has the following main components:

— MCF51CN128 48-pin QFN MCU
— RJ45 connector
— Use of two undefined twisted pairs of cable from RJ45 to power the board
— Ethernet PHY
— Reset button
— Power LED
— Non-standard 1 × 4 BDM
— Low dropout (LDO): the DC linear voltage regulator support from 3.7 V to 5.5 V
— 25 MHz crystal

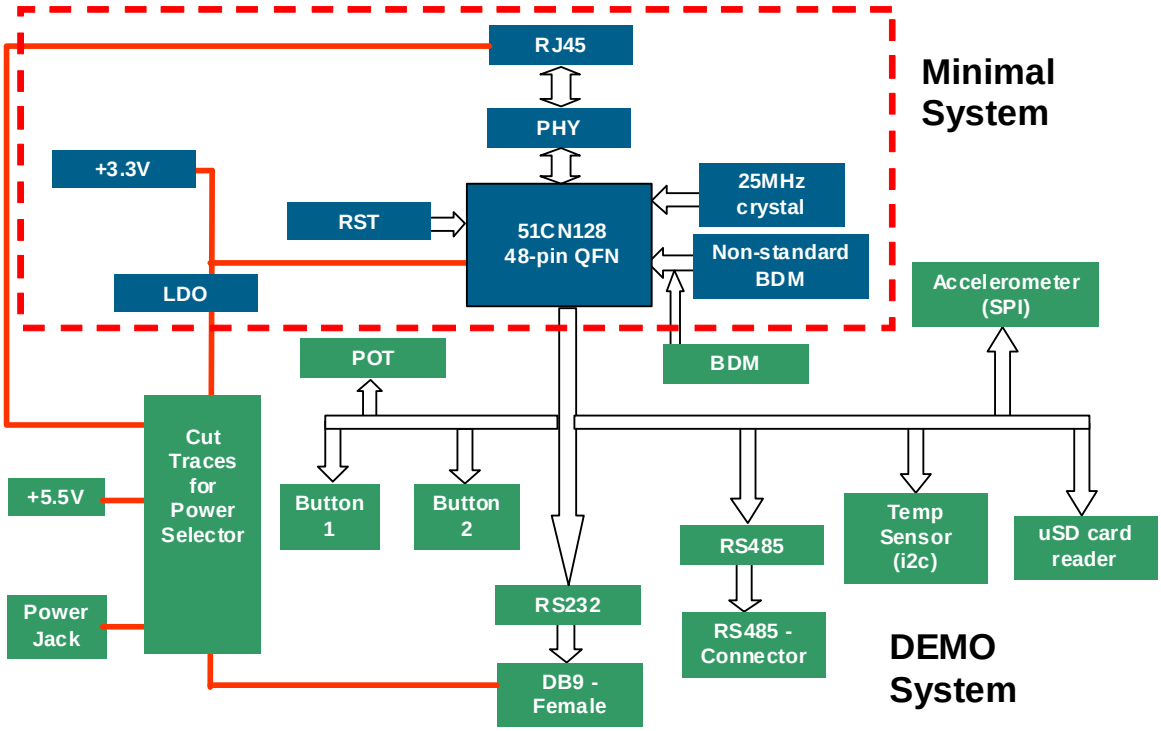

Figure 4. Hardware Block Diagram of MCF51CN128 Reference Design Board

b) Demo System — includes all hardware necessary to test the minimal system. For this bridge application, it has an RS232 and RS485 to test the UART interface. An accelerometer is used to test the SPI bridge. For debugging purposes, a 2 × 6 header is provided, so you can easily use serial signals outside the board.

It has the following main components:

— Power jack
— +5.5 V connector
— 2 × 6 serial header with the main serial signals
— Standard 2 × 3 BDM connector
— POT
— µSD card connector
— Two buttons
— RS232 and RS485 transceivers
— Accelerometer thru SPI interface
— Temperature sensor thru IIC interface
— General-purpose LED
— Cut traces to select among all the power options

Hardware limitations are as follows:

— Use of UART with hardware flow control or µSD card

— Use of potentiometer (POT) or accelerometer (SPI)

NOTE

When using MCF51CN128, the µSD card must not be connected to the µSD card connector, or the SPI-to-Ethernet bridge will not work.

c) Connection between minimal system and demo system — a set of zero-Ω resistors that isolate both parts are present and visible at the top and bottom layers. Disconnecting them isolates the minimal system from the demo system.

2.2 MCF51CN128 Reference Design Board Power Options

The following figure shows the power options for the MCF51CN128 reference design board.

The options are listed below:

- Ethernet: Brown and blue pairs take power. The brown pair is positive and the blue pair is negative. Power consumption drop due to Ethernet cable length must be considered when carrying power through the RJ45 cable. Note this is not power over Ethernet (PoE) but a way to power the MCF51CN128 reference design board.
- UART: pin six, +5.5 V unregulated power
- Power jack connector (default power source)
- Regulated 3.3 V jack connector

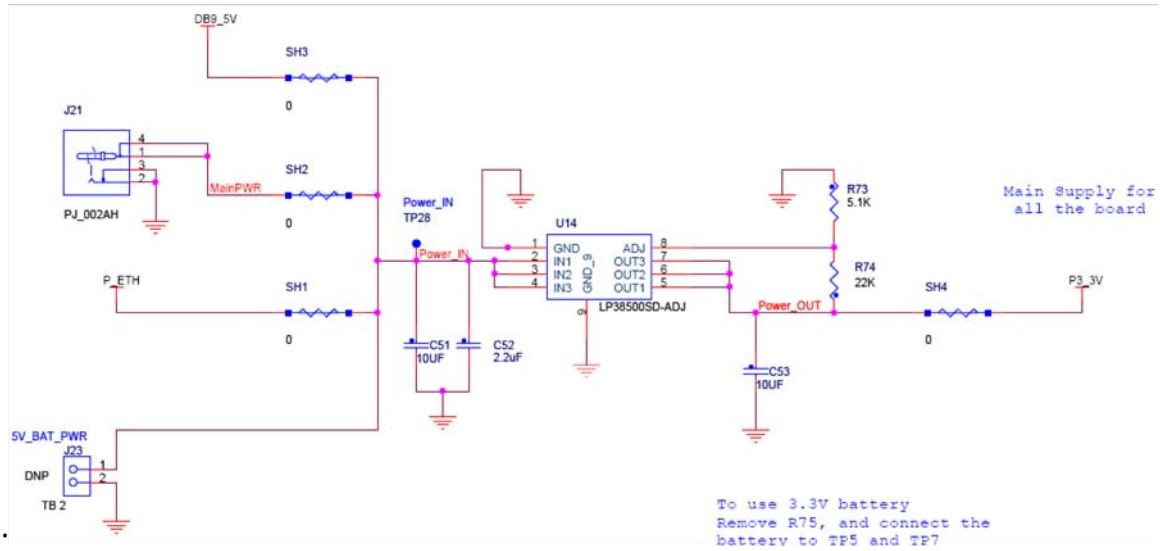

Figure 5. MCF51CN128 Reference Design Board Power Options Schematic

2.3 Hardware Pictures

The following picture shows the MCF51CN128 reference design hardware.

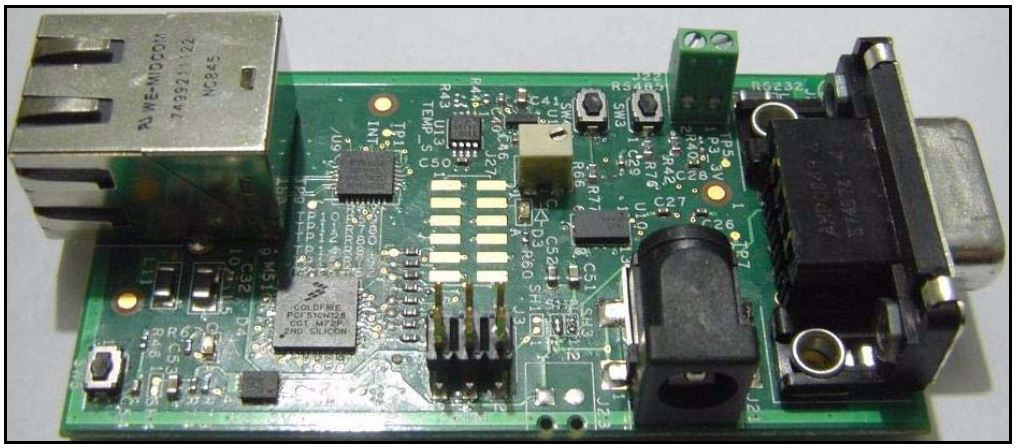

Figure 6. MCF51CN128 Reference Design Board Rev. A

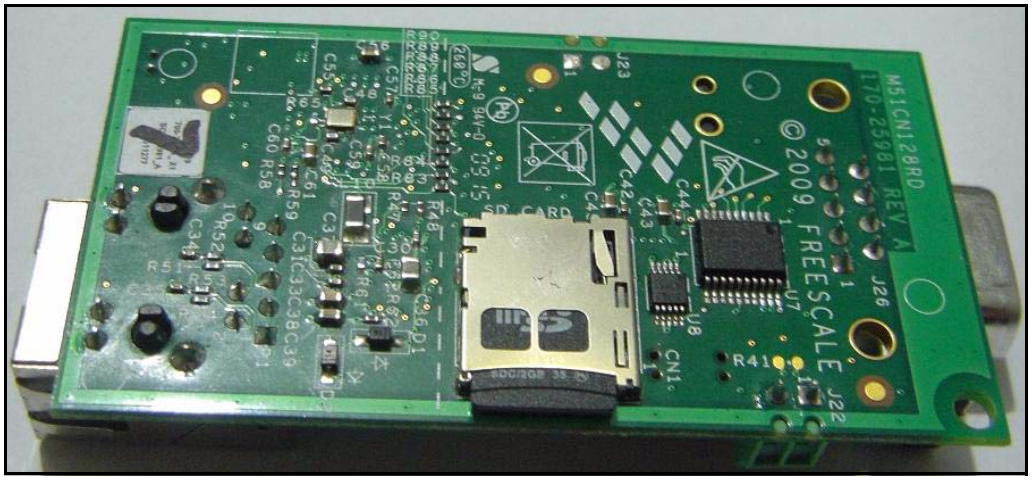

Figure 7. MCF51CN128 Reference Design Board Rev. A

The serial bridge is targeted at the MCF51CN128 reference design board but also works with the tower board, shown below. For more details about the tower visit www.freescale.com/tower.

Figure 8. Tower System with MCF51CN128 Card Rev. C

For the MCF51CN128 reference design hardware, there is no need to move jumpers because it doesn't have any. The board is ready to use as it is. Board schematics, layout, and gerber file are provided in case a customization in hardware is required for specific use of the serial-to-Ethernet bridge.

For the tower rev. C, use the TWR-MCFSICN user manual default jumper configuration.

2.4 Board Operation

The serial bridge can work in two modes:

- Configuration mode — a serial interface (UART or SPI) is used to configure serial bridge. This is the default mode.
- Bridge mode — characters received at serial interface, are sent to Ethernet and vice versa.

The web server interface works all the time but the serial configuration (UART or SPI) works only in configuration mode.

The serial bridge starts with the following configuration:

MAC Parameters	
MAC address	00:CF:52:35:00:07
IP address	192.168.1.3 for static implementation
Mask address	255.255.255.0
Gateway address	192.168.1.1
Server address to connect to an address	192.168.1.3
Static or dynamic address	Dynamic

TCP Parameters	
TCP port to connect to	1234
Client or server implementation	192.168.1.3
Configuration or bridge Implementation	Bridge

UART Parameters	
Port	First port
Baud rate	19200
Parity	None
Number of bits	8
Number of stop bits	1
Flow control	Software flow control

SPI Parameters	
Port	Second port
Baud rate	1 Mbps
Polarity	Low
Phase	Middle
Master or slave	Master
Polling or interrupt handling	Polling

The following is the list of commands that can be sent through UART or SPI to configure the bridge only in configuration mode. For example, if board_get_uart_port command is requested, the serial interface must send packets as shown in the following table and Figure 9.

Table 1. GET Commands to Get Bridge Parameters

GET Commands			
Command Name	**1st Character: Command ID**	**2nd Character: Number of Parameters**	**3rd Character: SubCommand ID**
board_get_eth_dhcp_auto	0x50	1	0
board_get_bridge_configuration	0x50	1	1
board_get_bridge_tcp_mode	0x50	1	2
board_get_bridge_tcp_server	0x50	1	3
board_get_uart_port	0x50	1	4
board_get_uart_parity	0x50	1	5
board_get_uart_number_of_bits	0x50	1	6
board_get_uart_stop_bits	0x50	1	7
board_get_uart_flow_control	0x50	1	8
board_get_spi_port	0x50	1	9
board_get_spi_polarity	0x50	1	10
board_get_spi_phase	0x50	1	11
board_get_spi_master	0x50	1	12
board_get_spi_interrupt	0x50	1	13
board_get_email_authentication_required	0x50	1	14
board_get_bridge_tcp_port	0x50	2	0
board_get_spi_baud	0x50	2	1
board_get_eth_ip_add	0x50	4	0
board_get_eth_netmask	0x50	4	1
board_get_eth_gateway	0x50	4	2
board_get_eth_server_add	0x50	4	3
board_get_uart_baud	0x50	4	4
board_get_eth_ethaddr	0x50	6	0
board_get_email_username	0x50	6	1
board_get_email_password	0x50	6	2
board_get_email_smtp_server	0x50	6	3

NOTE

GET functions will return the same number of characters as “Number of Parameters.”

Table 2. SET Commands to Change Bridge Parameters

SET Commands			
Command Name	**1st Character: Command ID**	**2nd Character: Number of Parameters**	**3rd Character: SubCommand ID**
board_set_eth_dhcp_auto	0xA0	1	0
board_set_bridge_configuration	0xA0	1	1
board_set_bridge_tcp_mode	0xA0	1	2
board_set_bridge_tcp_server	0xA0	1	3
board_set_uart_port	0xA0	1	4
board_set_uart_parity	0xA0	1	5
board_set_uart_number_of_bits	0xA0	1	6
board_set_uart_stop_bits	0xA0	1	7
board_set_uart_flow_control	0xA0	1	8
board_set_spi_port	0xA0	1	9
board_set_spi_polarity	0xA0	1	10
board_set_spi_phase	0xA0	1	11
board_set_spi_master	0xA0	1	12
board_set_spi_interrupt	0xA0	1	13
board_set_email_authentication_required	0xA0	1	14
board_set_bridge_tcp_port	0xA0	2	0
board_set_spi_baud	0xA0	2	1
board_set_eth_ip_add	0xA0	4	0
board_set_eth_netmask	0xA0	4	1
board_set_eth_gateway	0xA0	4	2
board_set_eth_server_add	0xA0	4	3
board_set_uart_baud	0xA0	4	4
board_set_eth_ethaddr	0xA0	6	0
board_set_email_username	0xA0	6	1
board_set_email_password	0xA0	6	2
board_set_email_smtp_server	0xA0	6	3

NOTE

SET functions will return zero if the command was correctly executed. Another number means the command failed.

RST Command	
Command name	1st Character
Reset the board	0x88

NOTE

The RST function returns zero if it was correctly received.

For example, to ask for the serial port configuration, send the following sequence:

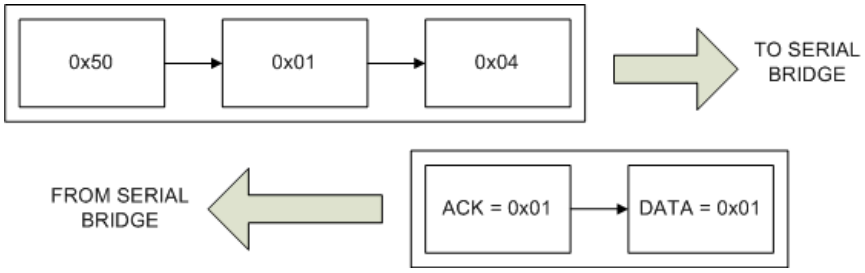

Figure 9. Packet Visual Explanation

The answer from the bridge means it correctly received the command and UART port configuration is located at port number one.

3 Introduction to the Serial-to-Ethernet Bridge Software

A serial-to-Ethernet bridge works between two interfaces. The purpose of serial-to-Ethernet bridge software is to get all the information at one interface and send it to the other as quickly as possible. Because of this, several features must be considered during development.

3.1 Flow Control

When a high-speed interface like Ethernet tries to connect to low-speed interfaces like UART or SPI, a flow-control protocol must be implemented as a speed adapter.

Ethernet does not feature a flow-control implementation. Transport control protocol (TCP) from the TCP/IP suite is used in this case. TCP provides a reliable connection by ensuring all the packets arrive at their destination using an acknowledge (ACK) scheme. TCP also negotiates packet lengths from one Ethernet device to the other, automatically.

For UART, two flow-control schemes and null-flow control were implemented. Using hardware flow control, two extra pins (clear-to-send (CTS) and request-to-send (RTS)) in addition to TxD and RxD are used to stop or start communication at both UART sides. Four communication pins (CTS, RTS, TXD, and RXD) are needed.

For software flow control, start (XON) and stop (XOFF) commands are sent as characters, as part of data communication. Only two pins are used (RxD and TxD). However, an extra software layer must be added to UART software drivers to have this feature. Additionally a non-flow control is presented.

However, at continuous transfers, either baud rate, transfers can quickly fill the UART software buffer if they are not taken by the serial bridge application at the same rate. Then the “extra” received at this point will be dropped. Baud rate and amount of data needed to calculate buffer’s high watermark are highly dependent on the tasks being run and tasks’ priorities when using an RTOS. This will be explained in the next section.

For SPI, no flow control is implemented to avoid altering an SPI message with an extra header. Flow control must be implemented at the application layer by using acknowledges or another customized flow-control protocol.

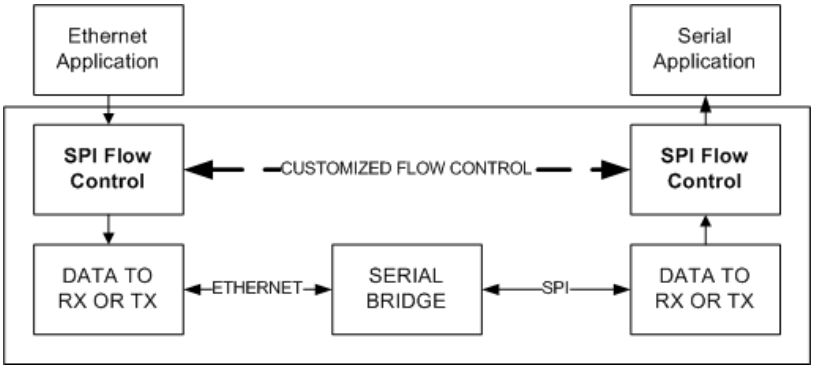

Figure 10. How Flow Control Must be Implemented for the SPI Interface at Upper Software Layers

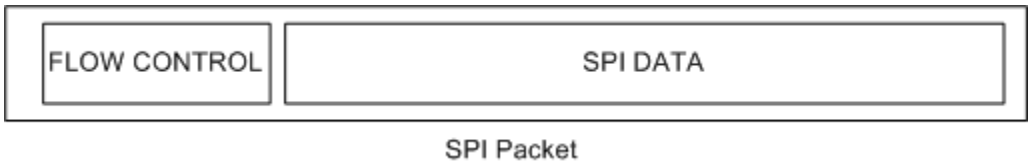

Figure 11. A Hypothetical SPI Packet Using Flow Control

3.2 Buffer Information Between Interfaces

Even if we can stop communication very easily with a flow-control protocol, if we do it very often, communication performance will go down, especially for serial protocols like UART and SPI.

This performance issue can be easily predicted by using UART or SPI hardware buffers, which are one-to-four bytes buffers long most of the time. In this hypothetical case, the Ethernet data length can go from 64 to 1518 bytes (this can be limited by TCP maximum length packet). Data will be sent as soon as possible to the UART controller, but its hardware buffer will be filled very quickly. Delays can be solved by using an interrupt-to-signal application bridge that a character can be sent. However, the delay until an interrupt requesting free space on UART hardware buffer happens must be considered. This delay can be decreased by using a software buffer between the bridge application and the UART controller. In this way, the application bridge can fill the software buffer and the UART ISR will take it character by character until it is emptied in this transmission case. In this way, the number of times communication is stopped by using flow control is reduced. See Figure 12 and Figure 13 for more details.

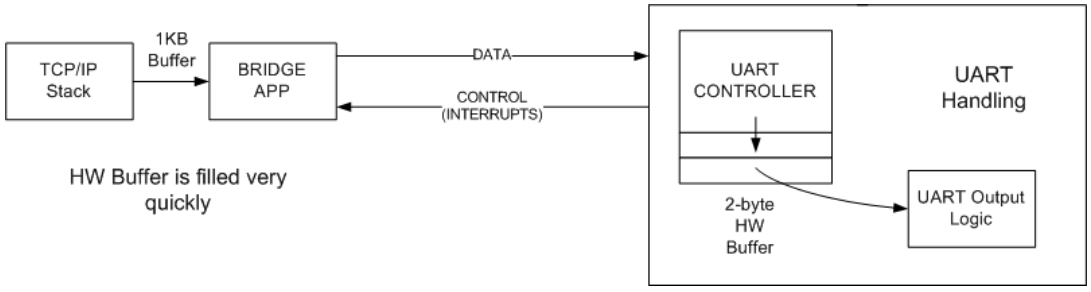

Figure 12. Bridge without Software Buffer Implementation

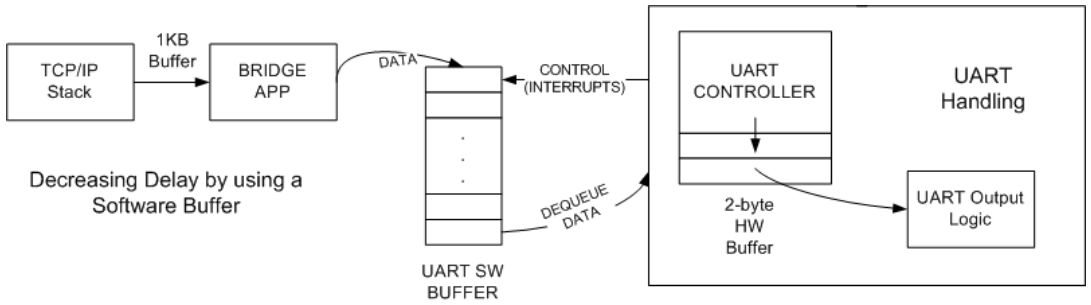

Figure 13. Bridge with Software Buffer Implementation

For the UART Rx case, software buffer is important to store all characters that can be received in one RTOS tick starting to count from the first character and send a more efficient Ethernet packet by using all the available max bytes for a packet instead of a few bytes. Ethernet is more efficient by using as much data as possible to avoid an Ethernet header bigger than the data (protocol overhead). This is useful when a lot of data is to be sent.

The serial bridge solves this problem by using software memory spaces known as buffers in a FIFO fashion-way to queue and de-queue data. We use independent buffers for Rx and Tx as a way to store all the information that can't be processed at the time by the bridge. For the UART software, Tx buffer length is 512 bytes, which matches with the half of TCP max packet and the half for Rx buffer length. 16 bytes is used for each SPI software buffer: Tx and Rx.

3.3 Communication Processing

Media access controllers (MACs) or Ethernet controllers have a default buffer implementation, which means the received data is stored as complete buffers that are managed by a higher Ethernet software implementation like a TCP/IP stack. In this way, hardware receives the packet as a complete array of bytes instead of byte per byte, getting a higher communication performance. These buffers' lengths are usually fixed to maximal Ethernet packet, which is 1536 bytes. In this implementation, interrupt processing for each packet seems better than a polling implementation that wastes time waiting for a reception flag to be set. However, protocols like UART and SPI receive character per character.

At low baud rates, like 300 kbps for UART and SPI, an interrupt approach seems prudent to avoid consuming extra CPU time waiting for a character transmission or reception in a while loop implementation until it's done. A problem arises when baud rate becomes higher (for example 12.5 Mbps for SPI), near CPU operational frequency like the MCF51CN128 (50 MHz CPU clock). In this case, a polling approach that will wait a few CPU cycles seems more adequate to avoid losing communication performance by each interrupt service routine (ISR) call that will occur very often as the baud rate is closer to the CPU frequency. For a UART implementation, which can get up to 115.2 kbps, an interrupt approach is correct for all the possible baud rates. For SPI, a 300 kbps is a valid baud rate to use with interrupts. Additionally, the MAC controller (Ethernet controller) doesn't show this problem because it receives as a complete packet and not byte per byte.

However, for SPI, which can get up to 12.5 Mbps, a polling approach is better. For UART implementation, interrupt approach is fixed as the default way to transmit and receive and cannot be changed at run time by the user. However, SPI has the option to choose between polling or interrupt character handling. At lower speeds like 300 kbps, interrupt handling makes sense, but at higher baud rates, polling must be chosen to avoid losing performance. For slave processing, interrupt approach must be used to receive/send characters. You must choose which is the better implementation for your SPI bridge depending on the baud rate and amount of data being sent. Table 3 shows a summary of this section.

Table 3. Baud Rates and Data Handling

Communication Process	
Baud rate	UART and SPI
Greater than 300 kbps	Pollings
Less than 300 kbps	Interrupts

3.4 Interoperability Among Several Serial Devices

The following serial bridge has specific default setting for Ethernet, SPI, and UART interfaces. This could be customized to fit specific needs, however, a default configuration is given for each interface. See Table 3 for details. These settings can be changed at run time by using a web page interface (Figure 26), which is embedded in HTML code with the firmware. Then, the serial bridge can be configured from any Ethernet node inside the network, like the Internet as shown in Table 3.

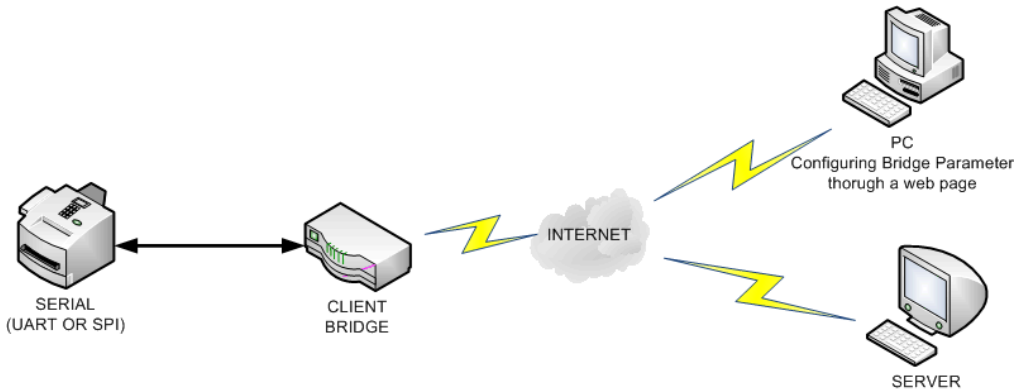

Figure 14. Serial Bridge in Bridge Mode

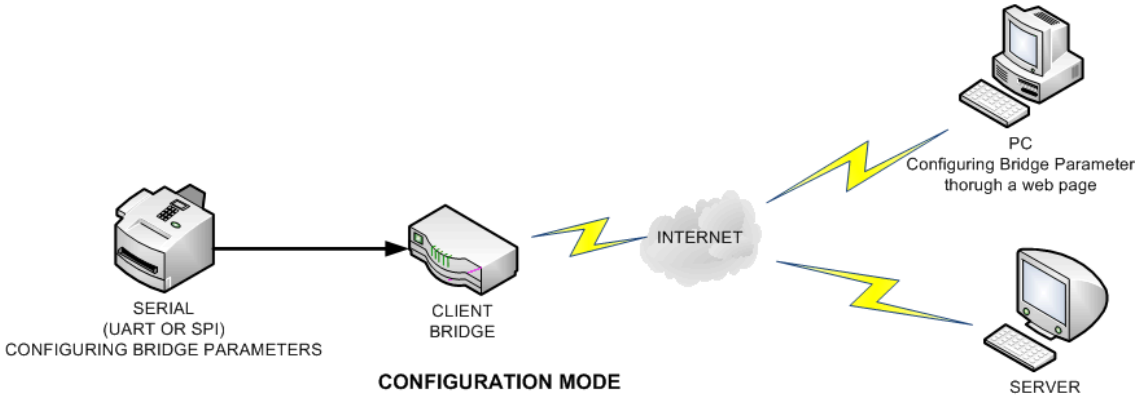

Figure 15. Serial Bridge in Configuration Mode

This webpage is present at any time and changes made through it take effect under the next reset, by either using the hardware (RST button) or by the web page as well. However, the serial bridge works in two modes: configuration mode or bridge mode. In configuration mode, serial interfaces like UART or SPI can be configured directly by a set of commands sent directly through the serial interface. As soon as these commands are according to customer user case, a switch to bridge command is requested and then a reset command. To change again to configuration mode, this must be requested by the web page. A switch from bridge mode to configuration mode from serial interface (UART or SPI) cannot be made to avoid altering bridge operability.

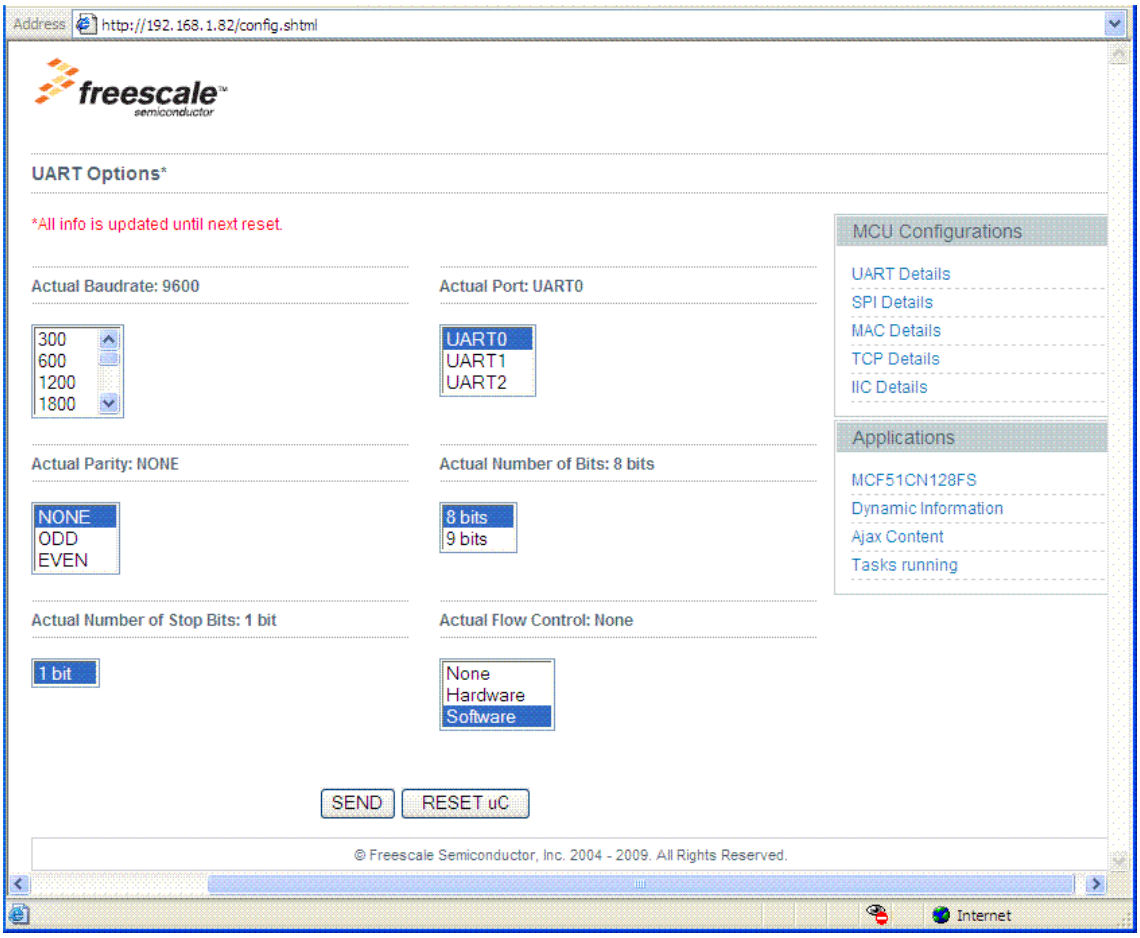

Figure 16. Webpage Configuring Serial Settings and Showing the Actual Configuration

3.5 Extra Features

The serial-to-Ethernet bridge works with TCP as its main feature. TCP is implemented in a server and client concept. A client connects only to a server. The server is always listening for new connections. The serial bridge can work in both directions by configuring a setting at run time through the serial interface at configuration mode or through the configuration web page at any time. As another feature, TCP/IP suite includes the Internet protocol (IP), which uses a unique address inside a network, like Internet, that allows us to communicate with any device inside this network of any architecture that runs TCP/IP suite. Both features together allow this serial-to-Ethernet bridge to connect a serial device (UART or SPI) virtually to any part of the world that is connected to the Internet. Figure 1, Figure 2, and Figure 3 show these cases.

3.6 Limitations

Ethernet is the key of the serial bridge. All the serial configurations must have Ethernet at one side and only one serial interface at a time (SPI or UART) at the other side. This means the bridge cannot be an SPI-to-UART bridge, because this goes out of the scope of this document. For more information about SPI and UART see the *MCF51CN128 Reference Manual*. For software go to Processor Expert™ for Codewarrior™ v6.2.1.

Security schemes like secure sockets layer (SSL) are not implemented in this phase of project. SSL works between the TCP/IP suite (TCP, UDP) and the applications layer (HTTP, FTP, SMTP). See Figure 17 for more details. Because of this, SSL software must point to receive and transmit functions from TCP/IP suite and applications must now point to this new SSL layer as the interface for Rx and Tx. However, security can be implemented externally by letting a firewall do this job.

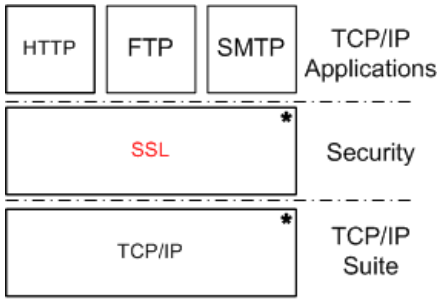

Figure 17. Figure H: SSL Layer in the TCP/IP Suite

3.7 Bridge Functionality

The following is the list of all features implemented by the serial bridge.

3.7.1 MAC Side

- DHCP service
- For MCF51CN128 reference design hardware, power can be applied to both unused pairs from RJ45 cable, so there's no need for a power-through jack. See Figure 5 for more details.
- For MCF51CN128 reference design hardware, we are using 48-pin QFN, using the smaller package for this software.
- For tower hardware, it uses 80-LQFP, allowing the Mini-FlexBus Interface to not be present in a smaller package.
- The following settings can be changed or shown at run time through the configuration web page or at compile time in the file constants.c:
  - MAC address
  - IP address
  - Mask address
  - Gateway address
  - Server address to connect as a client
  - Static or dynamic address

3.7.2 TCP Side

- Supports client or server implementation that allows connecting two serial bridges using the internet or a cross-over cable.
- A web page, always enabled, can configure all interfaces and bridge settings. A reset command can be sent through the web page to make settings take effect.
- The following settings can be changed or shown at run time through the configuration web page or at compile time in the file constants.c:
  - TCP port to connect or bind to
  - Client or server implementation
  - Configuration or bridge implementation.

3.7.3 UART Side

- Using hardware and software flow control for RS232 interface.
- Bridge can be configured through UART interface using a set of commands explained in Section 2.4, “Board Operation.”
- For MCF51CN128 reference design hardware, RS485 hardware is present as well.
- For MCF51CN128 reference design hardware, power can be applied to pin six of UART connector, so there’s no need for a power-through jack. See Figure 5 for more details.
- The following settings can be changed or shown at run time through the configuration web or at compile time in the file constants.c:
  - Port
  - Baud rate
  - Parity
  - Number of bits
  - Number of stop bits
  - Flow control

3.7.4 SPI Side

- Master or slave bridge support
- The bridge can be configured through SPI interface using a set of commands explained in Section 2.4, “Board Operation.”
- For MCF51CN128 reference design hardware, an accelerometer by SPI is on-board to test the SPI bridge.
- The following settings can be changed or shown at run time through the configuration web page or at compile time in the file constants.c:
  - Port
  - Baud rate
  - Polarity
  - Phase
  - Master or slave
  - Polling or interrupt handling.

NOTE

The software was developed for the MCF51CN128 reference design hardware to demonstrate low cost and small board size. It can also be used in the tower board.

Selection between either M51CN128RD or V1TOWER C-macros inside m51cn128evb.h file.

```
/*****************************************************************************/

/*Warning: only define one of them*/
#define M51CN128RD            /*pins moved to reference design hardware*/
//#define V1_TOWER              /*pins moved to reference design hardware*/
```

Figure 18. Code Snippet for Hardware Change

4 Serial-to-Ethernet Software

4.1 Software Architecture

The following figure shows how the serial bridge is divided and what software blocks are used for this implementation.

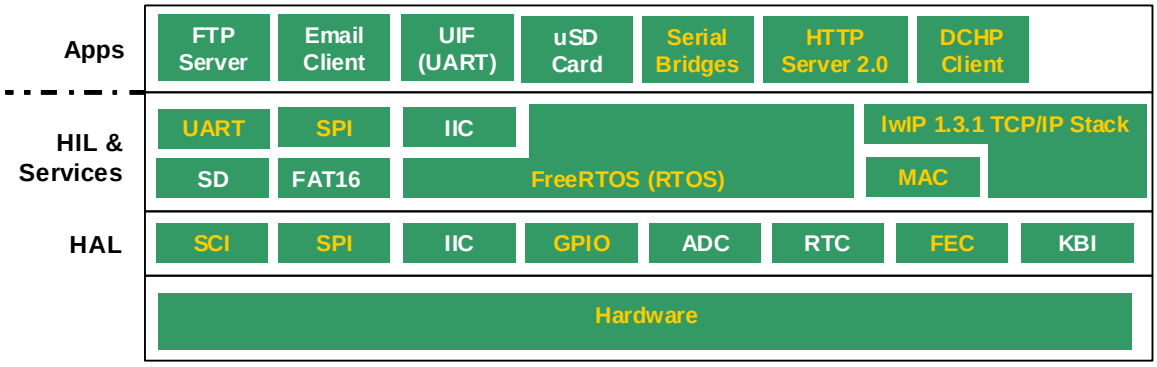

Figure 19. Software Segmentation

The main points to highlight are the use of open source software for the bridge.

4.1.1 FreeRTOS v5.0.3

FreeRTOS is the core of the serial bridge. It manages all the interrupts and tasks. The RTOS is highly coupled with the MCF51CN128 MCU by using RTC and CPU registers.

This RTOS was chosen because it has the following features:

- Open source
- Great features
- Can be used in commercial applications
- Forum support
- Smallest footprint (See Table 4)
- Available in 19 architectures
- Can be upgraded to SafeRTOS used for safety-critical applications

To change bridge source code, knowledge of RTOS is needed to change the following:

- Semaphores
- Mutexes
- Interrupt handling
- RAM memory handling
- Preemptive tasks
- Tasks' priorities and stack space

Table 4. FreeRTOS Memory Footprint for the Serial Bridge

FreeRTOS Files	ROM	RAM
list.c	212	0
queue.c	1728	0
tasks.c	3134	340
heap_3.c	68	0
port.c	300	4
portasm.s	248	0
Total	**5690 B**	**344 B**

The serial bridge uses a debugging service that uses two FreeRTOS services called configUSE_TRACE_FACILITY and INCLUDE_vTaskDelete. The first one is used to show tasks' usage at run time. The second is used to delete a task at run time. Both can be disabled if a final release doesn't need them, then decrease the memory footprint.

NOTE

Memory footprint doesn't consider RAM space taken by the kernel or the tasks at run time, only at compile time.

4.1.2 LwIP v1.3.0

This active open-source software handles all the Ethernet transactions and enables the TCP/IP suite. This TCP/IP stack is also highly coupled for this serial bridge and limited RAM memory on this MCU (24 KB). LwIP takes 12.15 KB from RAM memory at compile time to start all the TCP/IP services needed by the serial bridge. At run time, it takes necessary RAM memory available from the HEAP. It allocates and releases some buffers at run time. It uses a very limited set of network buffers to send and receive information from application higher layers.

Some of LwIP features are:

- IP
- ICMP
- UDP
- TCP
- Specialized raw API (used for this serial bridge)
- Optional socket API
- DHCP
- PPP
- ARP

The implementation of a layered software architecture allows to easily migrate this application software to other Freescale MCUs like MCF5225x, MCF5223x, or even higher processors if more resources are needed like FlexCAN, Encryption, or USB. The following sections give more detail.

The following table shows how modules are fit in the memory for the MCF51CN128 MCU with the debugging option enabled/disabled. The code snippet to enable/disable this feature is also shown.

```
/* ----------------------- DEBUG options

//#warning "still working with assetion, :
#define LWIP_NOASSERT
//#undef LWIP_NOASSERT
```

Figure 20. Code Snippet: Debugging Options OFF (cc.h)

Table 5. LwIP Memory Footprint for MCF51CN128 MCU (Debugging Options ON)

MODULE	ROM	RAM
TCP/IP API	7003	24
DHCP	6298	4
DNS	1887	569
ETHERNET	3201	47
FEC	1140	4678
ICMP	1188	0
IP	3556	12
LwIP+FREERTOS	1483	4
OTHERS	6087	8604
STDLIB	3056	20
TCP	15143	77
UDP	1594	4
TOTAL	**51636**	**14043**

Table 6. LwIP Memory Footprint for MCF51CN128 MCU (Debugging Options OFF)

MODULE	ROM	RAM
TCP/IP API	5887	24
DHCP	5213	4
DNS	1639	569
ETHERNET	2536	47
FEC	1140	4678
ICMP	772	0
IP	5072	12
LwIP+FREERTOS	824	4
OTHERS	4017	8604

Table 6. LwIP Memory Footprint for MCF51CN128 MCU (Debugging Options OFF) (continued)

STDLIB	3056	20
TCP	12896	77
UDP	1396	4
TOTAL	**44448**	**14043**

NOTE

Memory footprint doesn't consider RAM space taken by the TCP/IP suite at run time, only at compile time.

4.2 3.2 Software Hierarchy

The following figure shows files' hierarchy:

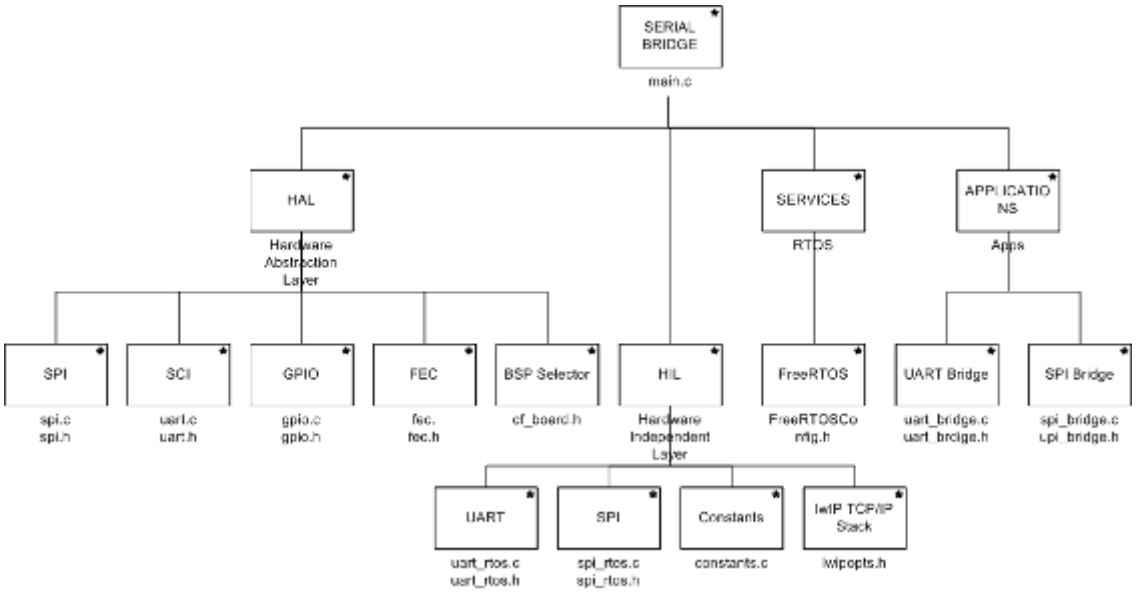

Figure 21. File Implementation

For this hierarchical files' distribution, an explanation of each one is provided as follows:

Table 7. File Purpose

Layer	File Name	Description
Main	main.c	Enable and disable the tasks running on MCF51CN128 MCU
HAL	spi.c	Low-level init for SPI bridge
	spi.h	Macros containing SPI low-level utilities
	uart.c	Low-level init for UART bridge
	uart.h	Macros containing SPI low-level utilities
	gpio.c	Routines that use pins for the selected MCU directly
	gpio.h	Points all the modules to a specific pin for the selected MCU
	fec.c	Low-level init for FEC driver
	fec.h	Number and length of Rx/Tx buffers
	cf_board.h	HAL layer to use with this serial bridge
HIL	uart_rtos.c	UART flow control and high level initialization
	uart_rtos.h	UART features like port, baud rates for high-level init
	spi_rtos.c	SPI-interrupted or polling processing
	spi_rtos.h	SPI features like port, baud rates for high-level init
	constants.c	Structure containing all the default parameters after reset
	lwipopts.h	LwIP options to enable/disable services
	FreeRTOSConfig.h	FreeRTOS options to enable/disable services
Applications	uart_bridge.c	Task running the UART-to-Ethernet bridge
	uart_bridge.h	Contains the UART software buffer length and task priority
	spi_bridge.c	Task running the SPI-to-Ethernet bridge
	spi_bridge.h	Contains the SPI software buffer length and task priority

5 Hardware Abstraction Layer (HAL) Implementation

The hardware abstraction layer (HAL) is defined as a collection of software components that make direct access to the hardware resources such as peripherals, configuration registers, optimized assembler routines (with their appropriate prototypes), pre-compiled object code libraries, or any other hardware-dependent resource, through the HAL-HW interface.

The following figures are representation of the software blocks more important for the HAL, the closest software layer to hardware, using modules and register present on the MCF51CN128 MCU.

5.1 Fast Ethernet Controller (FEC) Handling

The FEC is the Freescale implementation of the media access controller (MAC). The use of it for reception and transmission is explained in the following figures.

The FEC controller stores the Rx and Tx packets in a buffer descriptor scheme, which means eight bytes of memory are needed for each one of these. The buffer descriptor is mainly composed of the following elements:

- Buffer descriptor status and control flags
- Buffer length of received data or data to be transmitted
- Buffer pointer associated with reception or transmission

FEC controller has two registers (ERDSR and ETSDR) to point to start of buffer descriptor. Each one is for Rx and Tx. Then all the buffer descriptors (Rx or Tx) are consecutive. Then the FEC controller jumps from buffer descriptors waiting to the next available one until it finds the wrap flag in the buffer descriptor, which means it starts looking for an available buffer descriptor at the one pointed to by the FEC controller register (ERDSR and ETSDR). This means buffer descriptor acts in a circular buffer concept.

For this implementation, we have two Rx and Tx buffer descriptors. Each Rx buffer descriptor has an associated buffer of 1520 bytes. For the Tx buffer we have only one buffer of 1520 bytes.

The low-level reception of the FEC controller is divided into two parts:

- The FEC RX ISR acknowledges the FEC controller and signals the FEC reception task, highest priority task in this software project. The FEC controller automatically selects the next available Rx buffer and copies all the received data to it.
- The FEC reception task is waiting for the signaling from the FEC RX ISR. As soon as it gets it, it stores the buffer to the network buffers used by LwIP TCP/IP stack. Then it releases the buffer so the buffer can be available during the next reception by the FEC RX ISR. Finally, it waits for a new signal from the FEC RX ISR.

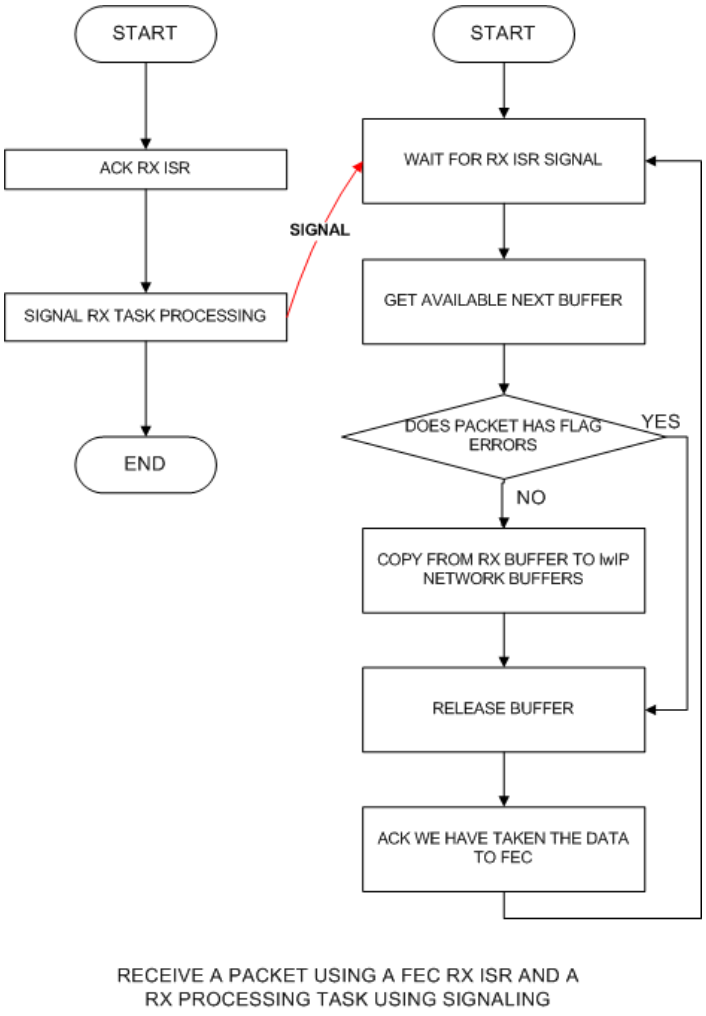

Figure 22. FEC Rx Processing

The low-level transmission of the FEC controller is as follows:

1. Wait for an available buffer descriptor
2. Fill from the network buffers (LwIP TCP/IP stack) to the Tx buffer
3. Start the Tx transfer

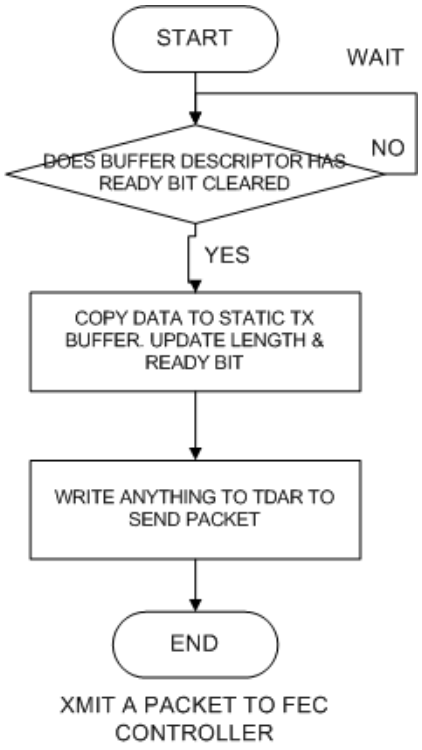

Figure 23. FEC Tx Processing

To avoid having the FEC transmit duplicate packets, follow these steps:

1. Create two Tx buffers' descriptors
2. Allocate one Tx buffer
3. Both Tx buffers' descriptors point to the same allocated Tx buffer
4. Send as usual; more details in Figure 24

```
/* Initialize transmit descriptor ring */
for( i = 0; i < NUM_TXBDS; i++ )
{
    tx_nbuf[i].status = TX_BD_L | TX_BD_TC;
    tx_nbuf[i].length = 0;
    tx_nbuf[i].data = &tx_buf[/*FSL: i * TX_BUFFER_SIZE*/0];/*FSL:workaround*/
}

/* Set the Wrap bit on the last one in the ring */
tx_nbuf[NUM_TXBDS - 1].status |= TX_BD_W;
```

Figure 24. Code Snippet Showing How Both Buffer Descriptors Point to the Same Buffer Area

5.2 Hardware-Independent Layer (HIL) Implementation

To maintain hardware independence, software components that belong to this layer can access controller's resources only by means of HIL components. Therefore, they refrain from directly accessing the resources of the controller on which they are running. This feature allows for components from this and the above layers to run on different controllers without further change.

The following figures represent the software blocks more important for the HIL, the closest software layer to applications, using HAL software layers.

5.2.1 UART Flow Control

The following flow diagrams are part of uart_rtos.c and uart_rtos.h implementation for the Rx and Tx flow control.

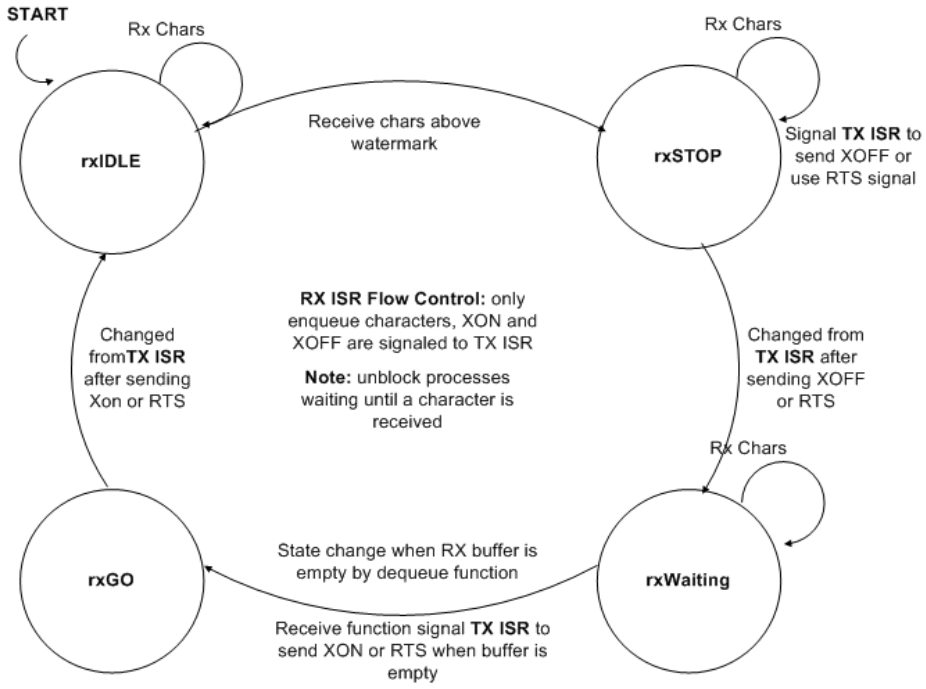

Figure 25. Rx ISR Flow Control State Machine

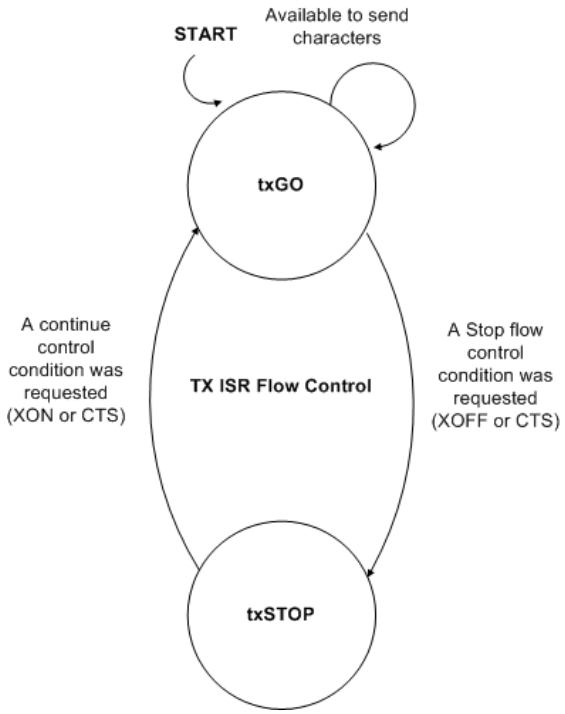

Figure 26. Tx ISR Flow Control State Machine

5.2.2 Overall Functionality

The following figures show the functionality of the application bridge as software and hardware blocks and the differences between them with a polling or interrupt implementation. Files uart_rtos.c and spi_rtos.c with their respective header files show this functionality.

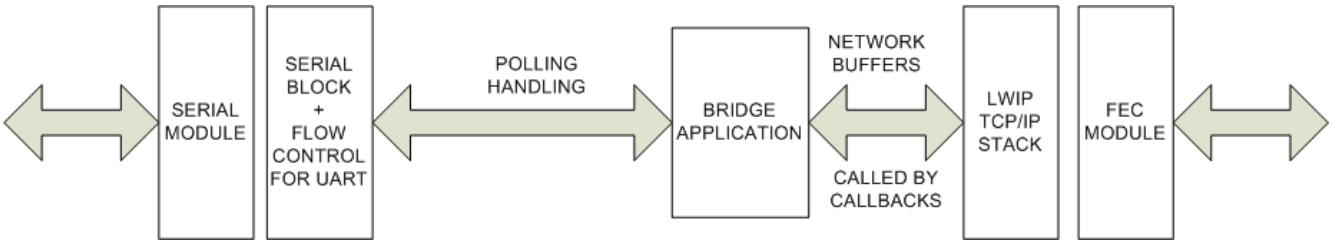

Figure 27. Polling Handling

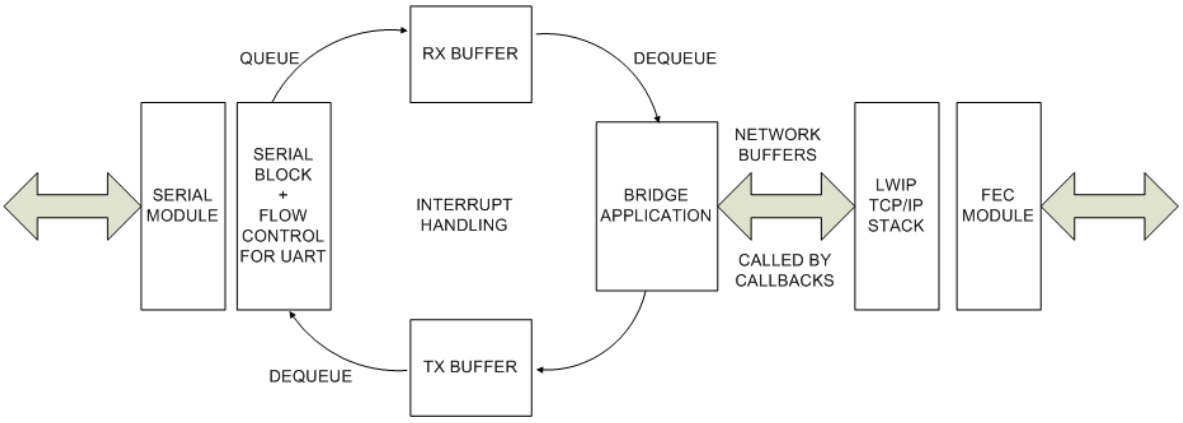

Figure 28. Interrupt Handling

6 Serial Bridge API

The following tables show the main functions used for the serial-to-Ethernet bridge

The bridge only has two main functions to allow the bridge to run:

Table 8. UART to Send and Receive Character

Sintaxis
void
BRIDGE_UART_Task(void *pvParameters)
Description
/** * Ethernet to SPI task * * @param none * @return none */
Description
UART to Ethernet Task

Table 9. Figure W: SPI-to-Send and Receive Characters

Sintaxis
void
BRIDGE_SPI_Task(void *pvParameters)
Description
/** * Ethernet to SPI task * * @param none * @return none */
Usage
SPI to Ethernet Task

7 Customization

For customization, the following files must be changed for a change in software or hardware:

File Name	Description
cf_board.h	Used to point to a new BSP, new HAL software drivers.
lwipopts.h	LwIP configuration file. Enables/disables TCP/IP options.
gpio.c/gpio.h	Changes GPIO used for all modules in the MCU.
FreeRTOSConfig.h	FreeRTOS user-configuration file. Enables/disables features.
constants.c	Default setting for this serial bridge.
uart_birdge.h	Select task priorities and buffer length for UART bridge.
spi_bridge.h	Select task priorities and buffer length for SPI bridge.

8 Conclusion

This document described considerations and use of a serial-to-Ethernet bridge. Parameters can be changed at both sides of the bridge. Flow control is a very important key to adapt higher-baud rate interfaces like Ethernet (10/100 Mbps) to lower-baud rate interfaces like UART (115.2 kbps). The use of sockets and software buffers are also main components that enabled interoperability and performance for this bridge. Bridge parameters can be configured using a web page or by commands using a serial interface. MCF51CN128 reference design hardware is a perfect choice to test and use as an end application board. Regarding the software, FreeRTOS and LwIP are greatly coupled for the MCF51CN128 and are open-source code, making a perfect choice as a firmware option. As a final note, the following table shows FreeRTOS and LwIP memory footprints:

Table 10. Table K: Final Results for FreeRTOS and LwIP

	ROM	RAM
FreeRTOS	5.5 KB	344 B
LwIP	43.4 KB	13.71 KB
Total	48.8 KB	12.8 KB

8.1 Considerations and References

Find the newest software updates and configuration files for the MCF51CN128 at the Freescale Semiconductor home page: www.freescale.com

- MCF51CN128 reference design and tower system were the hardware used to test the AN3906SW.
- For more information on the FEC, SPI, or UART interface, refer to MCF51CN128 Reference Manual at www.freescale.com.
- To learn more about the tower system please visit: www.freescale.com/tower.
- The BridgeSoftwareDemo software was developed and tested with CodeWarrior for Coldfire v6.2.1.
- Download the source files for AN3906SW.zip from www.freescale.com.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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

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

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3906
Rev. 0
06/2009

freescale™
semiconductor