

Application Note

SD Memory Card Interface Using SPI

The information in this document is current as of November 2007. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.

No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such NEC Electronics products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.

Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC Electronics no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.

NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".

The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

"Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.

"Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).

"Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

Notes:

1. "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
2. "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02.10

Revision History

Date	Revision	Section	Description
	—	—	First release

Contents

1.	Introduction	7
1.1	Overview of SD Memory-Card Interface	7
1.2	Overview of CSI Communications for SPI.....	8
1.2.1	CSI Features	8
1.3	Brief Overview of SPI Features	13
1.4	Comparison of NEC Electronics' CSI and SPI Transfer Operations	14
1.5	SD Memory Card System Features.....	15
1.5.1	SD Memory Card Hardware Interface.....	15
2.	Program Specification and Description.....	16
2.1	Initial Program Requirements.....	16
2.1.1	Program Description.....	17
2.2	Software Module Descriptions.....	17
2.2.1	Program Startup and Initialization.....	17
2.2.2	Main Program — NEC Electronics' CSI to SPI Serial-Communication Demo	18
2.2.3	SD Memory Card Functions.....	19
2.2.4	Serial-Interface Functions	21
2.2.5	SPI-Interface Initialization	22
2.2.6	Timer functions	22
2.2.7	Port Functions (Including Switch Input and LED Output)	23
2.2.8	Port Initialization	23
3.	Applilet Selections	25
3.1	Configuring Applilet for CSI (CSI5).....	28
3.2	Configuring Applilet for UART3	29
3.3	Generating Code with Applilet.....	35
4.	Demonstration Platform	40
5.	Program Demonstration	41
6.	Software Modules	43
7.	Appendix A — Flow Charts	44
7.1	System_initialize_flow	44
7.2	Main_flow	45
7.3	Sdmemory_flow	58
7.4	Serial_interface	76
7.5	Serial_interface_user	84
7.6	Port_interface.....	88
7.7	LED_interface	89
7.8	Switch_interface.....	103
7.9	Timer_interface.....	105
8.	Appendix B — Source Code Listings.....	108
8.1	crte.s.....	109
8.2	system.s	113
8.3	inttab.s	117
8.4	systeminit.c	121
8.5	main.c.....	123

8.6	sdmemory.c	129
8.7	serial.c	142
8.8	port.c	147
8.9	led_vjj2.c	149
8.10	sw_vjj2.c	157
8.11	timer.c	159
8.12	timer_user.c	162
8.13	system.inc	165
8.14	macrodriver.h	166
8.15	sdmemory.h	168
8.16	serial.h	170
8.17	port.h	172
8.18	led_vjj2.h	174
8.19	sw_vjj2.h	176
8.20	timer.h	177
9.	Appendix C — Development Tools	180
9.1	Software Tools	180
9.2	Hardware Tools	180
10.	Appendix D — Applicable Documents	181
11.	Appendix E — Modifications for MiniCube2	182
11.1	System Initialization Modifications	182
11.2	Link Directive Changes in 850.dir	182
12.	Appendix F— Port Association List	183

1. Introduction

This application note provides simple examples of the use of peripherals included in NEC Electronics microcontrollers (MCUs). The intent is to enhance your understanding of the setup and use of the peripheral functions. This application note focuses on using the MCU's clocked serial interface (CSI) in Synchronous Peripheral Interface (SPI) mode to control a Secure Digital (SD) or MultiMedia Card (MMC) memory card. The demonstration platform is the NEC Electronics AF-EV850 basic evaluation board, which features a V850ES/JJ2™ microcontroller (μPD70F3721).

Other features demonstrated in this application note are:

- ◆ Multiplexed LED drive
- ◆ Software-based switch debounce
- ◆ Interrupt-driven UART, using round-robin input buffering
- ◆ MINICUBE2 debugger
- ◆ Timer interrupts

This application note provides:

- ◆ Descriptions of peripheral features
- ◆ Program descriptions and specifications
- ◆ Software flow charts
- ◆ Applilet reference drivers
- ◆ A description of the demonstration platform
- ◆ Hardware block diagrams
- ◆ Software modules

The Applilet is a software tool that can generate simple driver code for processor peripherals. The Applilet provides a convenient means of generating the initial code for on-chip peripherals for quick evaluation. The generated code, however, usually requires modification to customize it to the specific requirements of an application.

Additional detail is available in the device user manuals and other related documents listed in Appendix A.

1.1 Overview of SD Memory-Card Interface

V850ES microcontrollers offer high-speed operation, large memory address space, and a variety of the most often used peripherals. These MCUs suit a variety of applications, including serving as controllers for stationary and portable mass-storage systems. The low-power operating characteristics of the V850ES MCUs make them ideal for battery-operated, portable, mass-storage systems.

This application note deals with the V850ES microcontroller interface connected to an SD memory card system using an SPI interface. The SD memory card is designed to provide high-capacity storage, high performance, and security in consumer electronic devices, such as audio and video electronics. The MMC is an earlier standard, which uses the same public protocol. Much of the SD memory card protocol is

proprietary. The SD memory-card system defines two alternative communication methods: SD and SPI communication interfaces.

In comparison, V850ES MCUs offer clocked-serial I/O (CSI), also known as 3-wire serial I/O, which uses three lines: serial clock (SCK), data input (SI) and data output (SO). In some cases, an additional handshake (HS) line between master and slave provides simultaneous transmission and reception. Data transmission and reception is synchronized with the clock, making communications straightforward. Most NEC Electronics microcontrollers implement one or more channels of the Serial Communication Interface (SCI) hardware.

The Serial Peripheral Interface (SPI) is an alternative to SCI that also uses the serial clock (SCK), data input (SI), and data output (SO) lines. Additionally SPI has a slave select (SS_B) signal used to select a communicating peripheral in a master/slave configuration. The clock also synchronizes SPI data transmission and reception.

While similar, the CSI and SPI interfaces have differences in their hardware implementations, clocking and control methods. This application note shows how to adapt NEC Electronics' CSI to interface with an SD memory-card system through an SPI interface, without additional hardware or modification.

1.2 Overview of CSI Communications for SPI

The NEC Electronics' CSI peripheral-communication method uses three lines: a serial clock (SCK) for synchronization, data-input (SI) and data-output (SO). In addition, the CSI interface has a chip-select line for each device on the bus. Data transmission and reception are synchronized with the SCK clock, making communications straightforward for most devices.

1.2.1 CSI Features

The CSI peripherals in V850ES Series MCUs typically offer multiple CSI channels. Most NEC Electronics 32-bit microcontrollers offer features similar those of the uPD70F3721, V850ES/JJ2—a 32-bit microcontroller. These features are:

- ◆ Transfer speeds as high as 5 Mbps
- ◆ Selectable master and slave mode
- ◆ 8- or 16-bit transmission data length
- ◆ MSB/LSB-first selection for data transfer
- ◆ Selection of multiple clock signals
- ◆ 3-wire interface
 - SO0n serial-transfer data output
 - SI0n serial receive data input
 - SCK0n_B serial clock
 - Where n = 0, 1 or 2. Thus, the uPD70F3318 (V850ES/KJ1Plus) MCU provides three CSI channels.

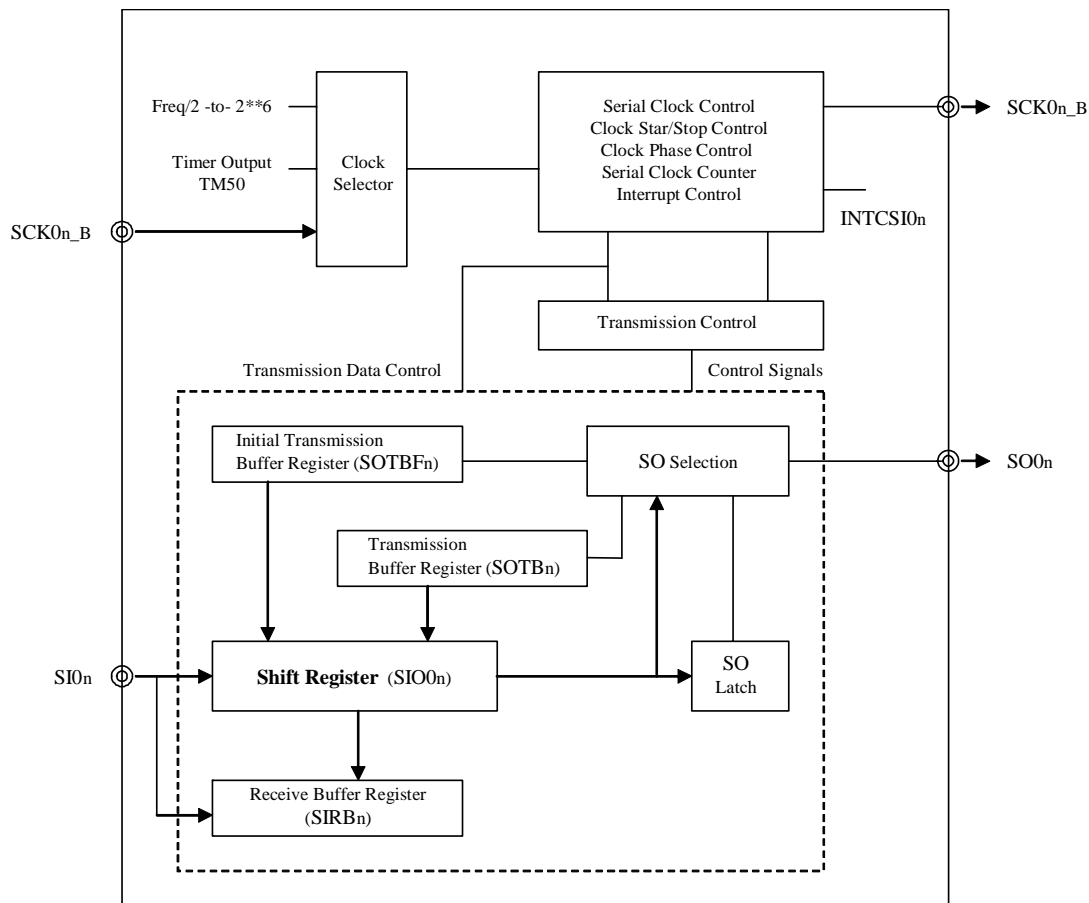
- ◆ Transmission/reception-completion interrupt
- ◆ Selectable transmission/reception mode or reception-only mode
- ◆ Two transmission-buffer registers and two reception-buffer registers
- ◆ Selectable single- or continuous-transfer mode

When the CSI peripheral is not used, SCK, SO and SI I/O pins can serve as port pins. The CSI units are configured using mode, control, and configuration registers, and dedicated hardware logic.

Table 1. CSI Control Registers

Register Type	Register Name	Symbol	Description
Control Registers	CSI Mode Register	CSIM0n	8-bit register specifies CSI operation modes
	Clock-Selection Register	CSICn	Controls CSI serial-transfer operation
Configuration Registers	Shift Register	SIO0n/SIO0nL	8/16-bit register converts parallel data to serial
	Receive-Buffer Register	SIRBn/SIRBnL	8/16-bit buffer register for receive data
	Transmit-Buffer Register	SOTBn/SOTBnL	8/16-bit buffer register for transmit data
	Initial Transmit Buffer	SOTBFn/SOTBFnL	Stores initial data in continuous-transfer mode
Configuration Hardware Logic	Clock-Select Logic		Selects serial clock to be used
	Serial-Clock Counter		Controls serial clock to shift register
	Interrupt Controller		Controls interrupt-request timing

Figure 1. CSI Peripheral Block Diagram



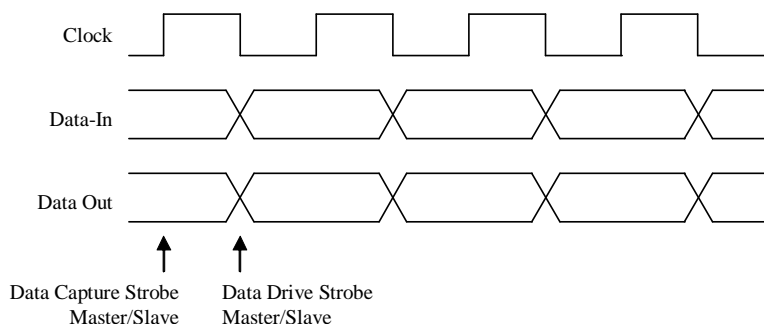
The CSI mode register configures the CSI unit for:

- ◆ Enable or disable
- ◆ Receive-only or transmit-and-receive mode
- ◆ 8- or 16-bit data length
- ◆ MSB or LSB first
- ◆ Single or continuous transfer

The clock selection and CSI transfer operation depends on:

- ◆ Whether the clock's positive or negative edge acts as the data-capture strobe (clock polarity)
- ◆ Whether the clock's first edge is used for data capture or as a data-drive strobe (clock phase)

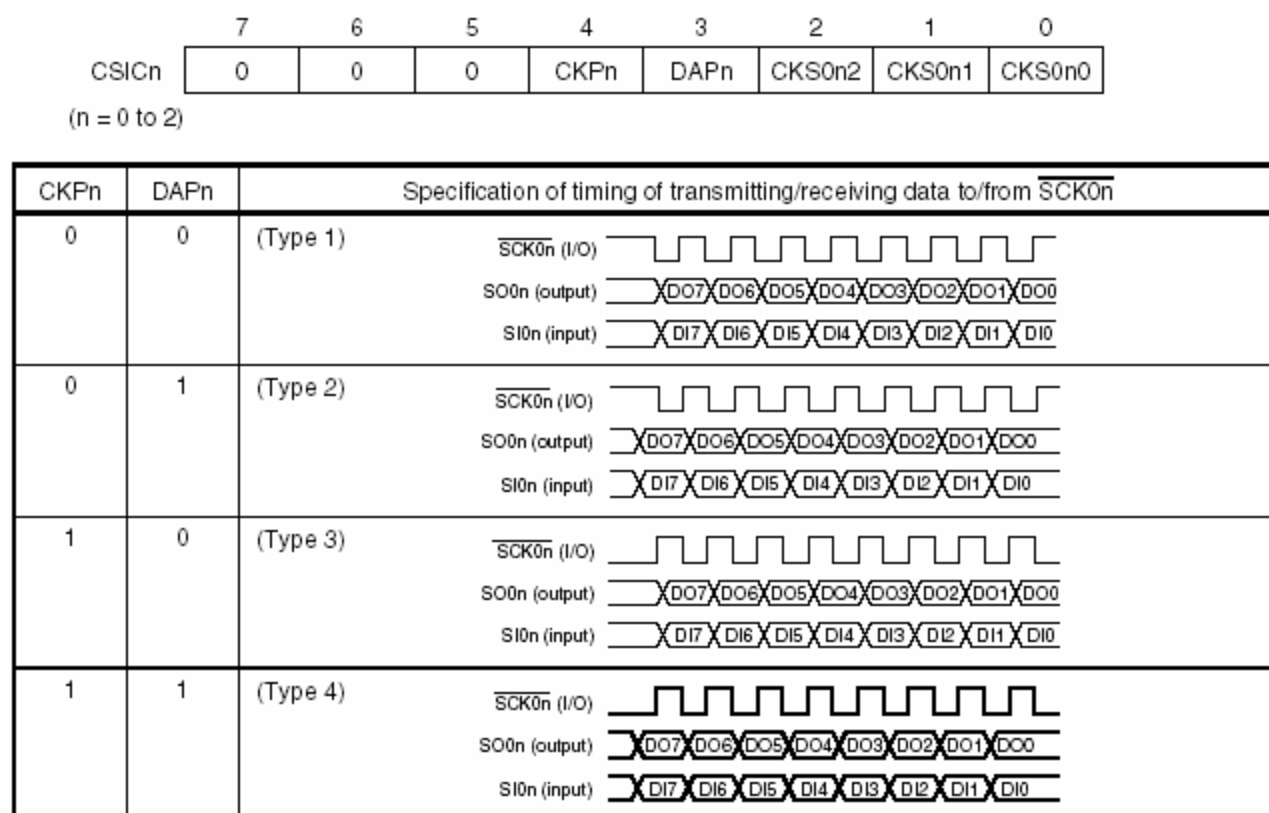
For example, the timing diagram below illustrates positive-edge data capture and using the first edge of the clock for data capture.

Figure 2. Timing When Using Positive Clock Edge for Data Capture

Note that the master unit controls the serial clock. If the first edge of the serial clock serves as the data-capture strobe, the slave-unit must be ready with data (driving data) before the first edge of the serial clock. Typically, in this case, chip select indicates the start of transmission from the master.

The second data-transfer method uses the first clock edge as the data drive strobe and the second edge as the data capture strobe. The first clock edge indicates the start of transmission from the master.

For NEC Electronics microcontrollers, the clock-selection register, CSICn, specifies the CSI transfer operation. CKPn selects the clock polarity, and DAPn selects whether the first edge of the serial clock is used for data capture or data drive.

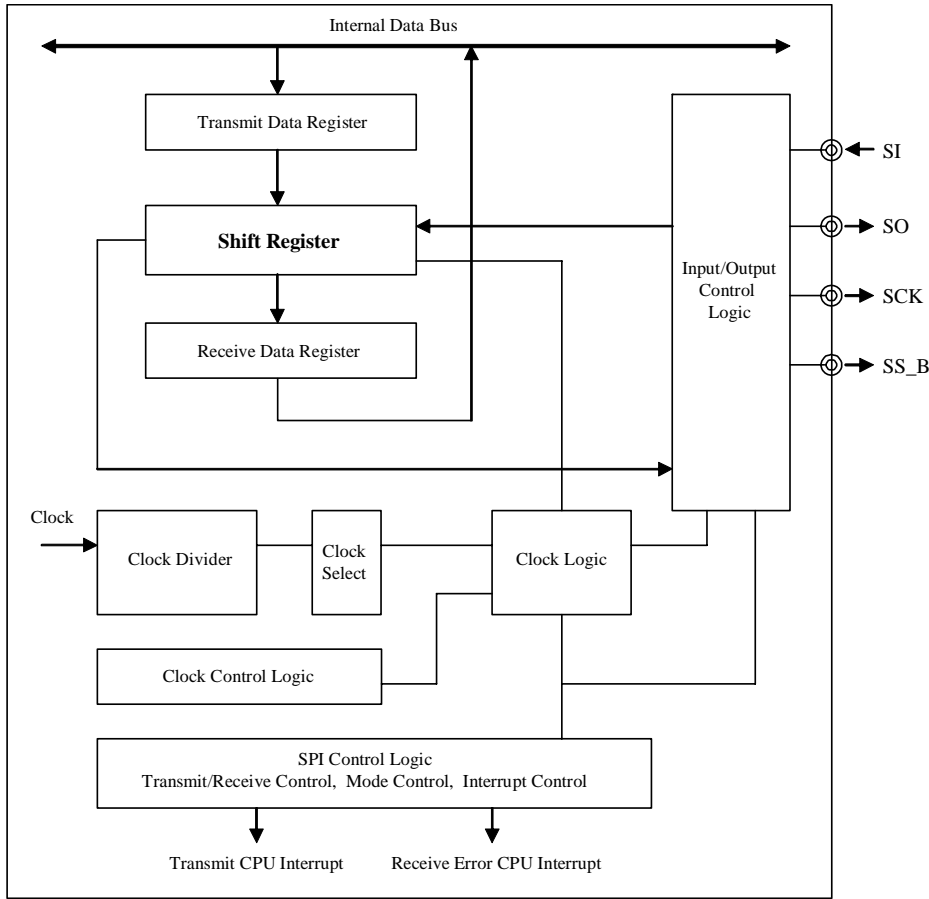
Figure 3. Clock-Selection Register Specifies CSI Transfer

The slave unit—whether another microcontroller or a peripheral device, such as a serial EEPROM—must provide interface logic to support all clocking methods (types one through four, as shown in the figure above). The master must be configured to communicate using the clocking method used by the slave.

1.3 Brief Overview of SPI Features

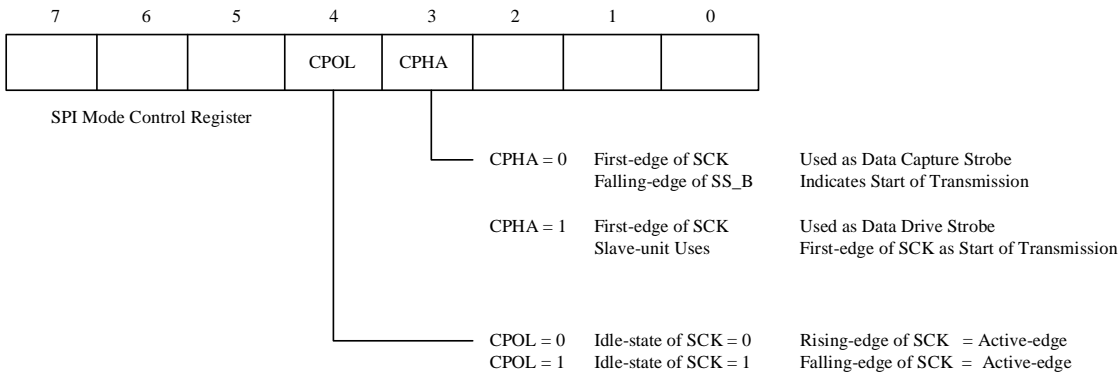
A typical SPI unit is similar to an NEC Electronics CSI unit.

Figure 4. Block Diagram of Typical SPI Unit



The SPI mode-control register specifies the transfer operation.

Figure 5. SPI Mode-Control Register Specifies Transfer Operations



When $CPHA = 0$, the first edge of SCK is the data-capture strobe for the first bit. Therefore, the slave unit must begin driving data before the first edge of SCK. The falling edge of SS_B (slave chip select) indicates the start of transmission. Between transmissions, SS_B must toggle HIGH and then LOW.

When $CPHA = 1$, the master begins driving data at the first edge of SCK. Therefore, the slave uses the first edge of SCK as the start-of-transmission signal. In this clocking mode, SS_B can remain LOW (active chip select state) between transmissions. This clocking method may be preferable for configurations with one master and one slave.

1.4 Comparison of NEC Electronics' CSI and SPI Transfer Operations

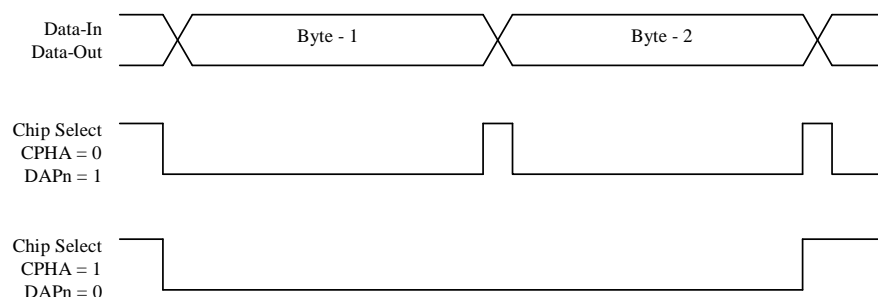
The NEC Electronics' CSI unit has a clock-selection register (CSICn), which specifies the clocking method, depending on the CKPn and DAPn bits. SPI, on the other hand, uses an SPI control register to specify the clocking method, depending on the CPOL and CPHA bits. These bits, located in the control registers, specify one of four possible clocking methods. The following chart compares NEC Electronics' CSI clocking methods with SPI.

Table 2. Clocking-Method Selection for CSI and SPI

NEC Electronics CSI Clocking Method			SPI Clocking Method		
CKPn	DAPn	Clocking Type Descriptions	CPOL	CPHA	Clocking Type Description
0	0	Type-1 Clocking Method	1	1	Idle-State Clock = 1
		Idle-State Clock = 1			First edge SCK is Data Drive Strobe
		First edge of SCK is Data Drive Strobe			
0	1	Type-2 Clocking Method	1	0	Idle-State Clock = 1
		Idle-State Clock = 1			First clock edge is Data Capture Strobe
		First clock edge is Data Capture Strobe			
1	0	Type-3 Clocking Method	0	1	Idle-State Clock = 0
		Idle-State Clock = 0			First clock edge is Data Drive Strobe
		First clock edge is Data Drive Strobe			
1	1	Type-4 Clocking Method	0	0	Idle-State Clock = 0
		Idle-State Clock = 0			First clock edge is Data Capture Strobe
		First clock edge is Data Capture Strobe			

In both NEC Electronics' CSI and SPI, when the first edge of SCK is the data-capture strobe, the chip select (SS_B) as the start-of-transmission signal. In this case, the chip select should be HIGH and then LOW between data transmissions.

When the first edge of SCK is the data-drive strobe, SCK also acts as the start-of-transmission signal. In this case, the chip select can remain active between data transmissions.

Figure 6. Using First Edge of Sck as Data Strobe

1.5 SD Memory Card System Features

SD memory cards are designed to provide high capacity and performance with built-in security features. SD cards can be:

- ◆ Flash memory
- ◆ One-time programmable memories
- ◆ ROM, including cards used for distribution of software, video or audio
- ◆ Special-purpose cards, such as wireless units (WiFi, Bluetooth)

SD cards communicate via a 9-pin interface consisting of clock, command, four data lines, and three power lines. Operating frequencies can range as high as 25 MHz. For further details, refer to the specifications from the Technical Committee, SD Card Association:

- ◆ SD memory card Physical Layer Specification, Version-1.01
- ◆ SD Specification, Part-E1 SDIO Simplified Specification, Version-1.10

1.5.1 SD Memory Card Hardware Interface

As mentioned earlier, the SD card system defines two alternative communication methods: SD and SPI. SD communication mode uses a wider bus, thereby achieving higher-speed data transfers. The SPI standard defines the physical link with a host microcontroller and is commonly used in many microcontroller-based designs. The SD card SPI interface uses the signals shown in the following table.

Table 3. SD Card SPI Signals

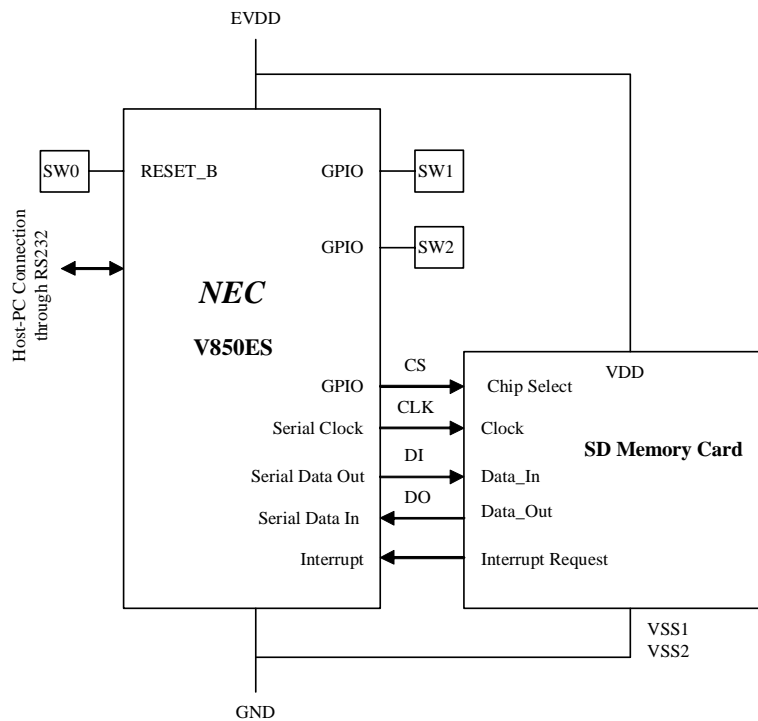
Signal	Symbol	Description
Chip select	CS	Host microcontroller to SD memory card chip-select signal
Clock	CLK	Host microcontroller to SD memory card clock signal
Data input	DI	Host microcontroller to SD memory card data input signal
Data output	DO	SD memory card to host microcontroller data output signal

2. Program Specification and Description

2.1 Initial Program Requirements

The demonstration program runs on a V850ES/KJ1+ board with an SD card interface. You input text via hyperterminal, and the text is written into an SD memory card as it is typed. The card contents are then read and displayed via hyperterminal.

Figure 7. Interfacing Demonstration Board to SD Memory Card



The SD card interface signals connect to a serial I/O port of a V850ES Series microcontroller. The microcontroller connects to a host PC through an RS232 (UART) interface. The hyperterminal program runs on the host PC. You control transfers with the switches on the demo board:

- ◆ Press SW1 to write text typed in hyperterminal into an SD memory card.
- ◆ Press SW2 to read the SD memory card contents and display them via hyperterminal.

2.1.1 Program Description

After start-up initialization, the program outputs the heading message to a console connected to the DB9 serial port at 9600-N-8-1. This console can be a PC running an application, such as Hyperterm or TeraTerm. Since the demo board is a DCE device (sends on pin 2, receives on pin 3), a null-modem adapter is used.

The program outputs a message requesting input of the sector number to use. You then enter the sector number to be read from or written to. If you want to write data to a sector, enter the data after typing the sector number and pressing return. You can enter up to 511 characters. If you enter more than 511 characters, the count resets to zero and you start over. The LEDs on the demo board show the number of characters entered.

To write the data, press switch 1. If you have not entered 511 characters, pad characters of 0x21 (exclamation marks) are added to fill the buffer. The display shows the hex value of the data.

To read the data, enter the sector number and press switch 2. The data buffer is filled with “z” characters before completing the read. The program remains in this loop forever, asking for sector numbers and reading or writing them.

For additional information on the setup to accommodate the MiniCube2, see Appendix E.

2.2 Software Module Descriptions

The demonstration program consists of the following major sections:

- ◆ Initialization code, called before the main() program starts; the code initializes the microcontroller’s clocks and peripheral sections, and initializes memory for running C code.
- ◆ The main program loop, which responds to your inputs via the terminal and switches
- ◆ Subroutines for CSI05 peripheral access (SD/MMC memory)
- ◆ Subroutines for UART3 access (terminal communications)
- ◆ Subroutines for LED display
- ◆ Subroutines for switch monitoring
- ◆ Subroutines for timer operation

2.2.1 Program Startup and Initialization

The following modules perform program startup and initialization:

- ◆ crete.s environment initialization
- ◆ SystemInit.c subsystem initialization
- ◆ system.s _Clock_Init
- ◆ inttab.s interrupt-vector table

Appendix A provides the reference system initialization flowchart, `SPI_1.0_system_initialize_flow`. Appendix B gives reference source-code listings for `crete.s`, `system.s`, `inttab.s` and `systeminit.c`.

For V850ES programs written in C, the environment must be set up before the program can run. This startup code is supplied in an assembly language startup file, generally named `crete.s`. This code specifies the reset vector, which determines where the program begins execution after a hardware reset. A hardware reset can be caused by power up, the reset switch or the MCU's watchdog timer. Before calling the main program, the startup code sets up the initial values in system registers and performs memory initialization or clearing.

On power up or reset, execution starts at the reset-interrupt vector. This vector is defined in `crete.s` as .section "RESET" and is linked to address 0x00000000—the start of the interrupt-vector table. This section holds an instruction to jump to `__start` and is linked at the start of the interrupt-vector table. In the routine `inttab.s` the section "RESET" is commented out so that there is not a conflict. The program uses the definition in `crete.s`.

The make file `850.dir` defines a section called "STARTUP = `crete.o`" that ensures this object is linked first.

When the Applilet generates a C program for the V850ES, the tool automatically generates the `crete.s` startup assembly-language file. This file provides definitions to allocate the `argc`, `argv` parameters that pass to the main program, allocates the stack space, and sets the stack register to point to that space. The startup routine sets the reset vector to jump to start and enables the on-chip debug mode. The `crete.s` file also includes a call to the `Clock_Init()` function to set the system clock and clears the `sbss` and `bss` memory areas. After all registers have been set up, a call is made to `SystemInit()`, which calls initialization routines for some (but not all) peripherals. On return from `SystemInit`, the startup routine calls the `main()` function. On return from `main`, a halt instruction executes.

The `SystemInit` functions called are:

- ◆ `PORT_Init()`
- ◆ `UART3_Init()`
- ◆ `CSIB5_Init()`
- ◆ `TMP0_Init()`

2.2.2 Main Program — NEC Electronics' CSI to SPI Serial-Communication Demo

Appendix A provides the flowcharts for `main.c` and `SPI_1.1_main_flow`. Appendix B has the source code for `main.c`.

After completing peripheral initialization (which includes calls to routines that initialize input-switch handling, LED output, and timer interrupts), the startup code calls the `main()` program. Before the call, the startup routine inserts a small delay to allow the UART to finish initialization. Startup of the UART subsystem can cause a glitch at the output, and if data is transmitted immediately after setup, the glitch looks like a start bit.

To display the program name and some operating instructions, a call is made to `uart3_tx_msg` with a pointer to the string to be displayed.

Calling the `SDmemory_Init()` function puts the SD/MMC memory into SPI mode. This function returns the initialization status and, if initialization fails, displays the message “main - memory card init status”. The program then aborts.

If initialization is successful, the `SDReadStatus()` function verifies that the memory is communicating and accepting commands.

The `main()` program then enters an endless loop. In this loop `main` resets the pointers and counters, puts the start-data token in the write buffer, displays the buffer count on the LEDs, and asks you to enter a sector number for the sector to be read or written. Your input for the sector number is retrieved by calling `get_sector()`. If the sector is not valid, the loop restarts.

If the sector is valid, the program enters an inner loop that continually checks for either a character received from UART3 or a switch pressed.

Calling `Check_UART3_Receive()` checks for your input. If a character is received, it is placed in the send buffer and echoed back to the console display. The LED is updated with the total number of characters entered. You can enter up to 512 characters, the sector size of the memory, and then wrap-around occurs. If you press SW1 when the send buffer is empty, the buffer is padded out with 0x21. The contents are written to the memory card, along with the data token and a dummy CRC value. The buffer value is dumped to the console.

Pressing SW2 initializes the receive buffer “z”. The requested sector is then read from the memory card into the receive buffer, and then dumped to the console.

2.2.3 SD Memory Card Functions

Appendix A provides reference flowcharts for SD/MMC memory card functions in `SPI_2.0sdmemory_flow`. Appendix B provides program source-code listings for `sdmemory.c`.

The SD/MMC memory-card functions are:

- ◆ `send_pad()`
- ◆ `build_cmd()`
- ◆ `SDmemory_Init()`
- ◆ `SDmemory_CMD_R16()`
- ◆ `SDmemory_CMD16()`
- ◆ `err_val()`, `err_text()`
- ◆ `R1_Initiate()`, `R2_Initiate()`
- ◆ `do_crc7()` stub
- ◆ `SDmemory_R_query()`
- ◆ `SDmemory_DT_query()`

- ◆ SDmemory_DR_query()
- ◆ SDReadSector()
- ◆ SDWriteSector()
- ◆ SDReadStatus()

2.2.3.1 SD/MMC Initialization

For this demonstration, the memory card must be reset and switched into SPI mode. These actions are done by deselecting the card and sending 10 pad characters out the SPI port. These characters serve as a clock signal that the card uses to complete internal power-up reset processing. The card must not be selected during this time. Because send operations are interrupt driven, the program monitors a done flag to determine when the SPI transmit completes. Once the pad characters are transmitted, a CMD0 message is sent to the card to put it into reset. When this message has been sent and a proper response received, a CMD1 message puts the card into SPI mode. The program then monitors for a proper R1 message response.

2.2.3.2 Write Memory-Card Sector

Calling SDWriteSector() writes 512 bytes of data to the specified sector. After selecting the device by lowering the chip-select line, a CMD24 message is built and sent. The CMD24 message contains the destination block address, which is the sector number multiplied by 512. After the message has been sent, the card responds with an R1 message. Count NWR pad characters are then sent, as specified by Reference 8 (listed in Appendix D of this application note) Table 5-11. The data is then sent, preceded by a start token and followed by two dummy CRC characters. Then a check is made for receipt of a data-response token. When the response token has been received, the routine sends NEC count pad characters (defined in Reference 8, Table 5-11) and then deselects the device.

This description does not cover all error possibilities. Refer to the source code and flowcharts for more detail.

2.2.3.3 Read Memory Card Sector

The SDReadSector() function requests reading a memory-card sector into a specified buffer. The device is selected and command CMD17 is built and sent, along with the specified block address. The block address is the sector number multiplied by 512. The card responds with an R1 message. If the response does not indicate an error, the card starts sending pad characters and looks to receive a data token. The data token indicates that the start of data follows. After receiving the data token, the function sends 514 pad characters to clock in the data. Then the function sends NEC count pad characters (as defined in Reference 8, Table 5-11; this reference is listed in Appendix E of this application note). The function then deselects the device and returns the status of the read sector request.

2.2.3.4 SD/MMC 16-Byte Response Command

The SDmemory_CMD_R16 function sends either CMD9 send card specific data (CSD) or CMD10 send card identification data (CID) messages, and then receives the 16-byte response message.

2.2.4 Serial-Interface Functions

Appendix A provides the flowchart for the SPI_3.0_serial_interface. Appendix B provides the source code for serial.c and serial_user.c.

2.2.4.1 Serial-Interface Initialization

- ◆ UART3_Init()
- ◆ UART3_UserInit()

The alternate function of port PMC8 selects the UART3 (Universal Asynchronous Receive Transmit) function. This UART is set up to provide 8 data bits, no parity bit, and 1 stop bit, and to send the least-significant bit first. The baud rate is based on a 20-MHz clock and set to divide down for 9600 baud. Receive and transmit interrupts are used but at the lowest priority. Initialization enables the UART interrupts, receive and transmit. The user initialization consists of setting up the round-robin buffer put and get pointers to index the start of the receive buffer.

2.2.4.2 Serial-Interface Transmit

- ◆ UART3_SendData()
- ◆ MD_INTUA3T()
- ◆ uart3_tx_msg()

The serial-transmit operation is interrupt driven and initiated by a call to UART3_SendData(). The pointers to the string and number of characters to be sent are input parameters. On entry, this routine enables the UART transmit port, saves the input parameters, clears the done flag, and writes the first byte of the string to the UART transmit register. The routine increments the string pointer by one and decrements the number of characters decremented by one. Control then returns to the caller and the remainder of the string is sent as part of interrupt processing.

When character transmission completes, the function calls interrupt vector MD_INTUA3T to check the remaining character count. If no characters remain to be sent, the done flag is set and the program exits the interrupt service. If characters remain to be sent, the routine writes the next character to the UART transmit register, increments the data pointer, and decrements the number of characters remaining to send. Then the UART transmit interrupt service routine is exited.

2.2.4.3 Serial-Interface Receive

- ◆ UART3_ReceiveData()
- ◆ MD_INTUASR()
- ◆ UART3_Receive()
- ◆ Check_UART3_Receive()

To receive data via the serial connection, the UART receiver and the receive interrupt are enabled.

When a UART receives a character, the UART generates an interrupt and vectors program execution to the MD_INTUA3R interrupt-service routine. Interrupts are enabled immediately so as not to block other interrupts. The routine reads the UART status register and checks for errors. If there were no receive errors,

the routine reads the receive register and places the data into the round-robin buffer by calling the function `UART3_Receive()`.

2.2.5 SPI-Interface Initialization

- ◆ `CSIB5_Init()`

Port PMC6 is configured to function as the SPI interface. All operation is stopped, interrupts are turned off and cleared. CSIB5 is configured to operate MSB first, with 8-bit data transfers in a single-transfer mode. The routine selects a transmit clock speed of $f_{xx}/64$, which is 312.5 kHz. Finally, the routine enables the SPI receive and transmit registers. For this demonstration, the SPI interface operates in polled mode, and the interrupts are not enabled.

- ◆ `CSIB5_SendData()`

A single function handles sending and receiving data, since the SPI interface requires that you send data to it. The interface sends the specified number of bytes from the specified address, and receives the same number of bytes and places them in an SPI receive buffer.

- ◆ `CSIB5_select_SPI()`
- ◆ `CSIB5_deselect_SPI()`

2.2.6 Timer functions

The `SPI_5.0_timer_interface` flowchart is in Appendix A. Appendix B gives the source code for `timer.c` and `timer_user.c`.

2.2.6.1 Timer Initialization

- ◆ `TMP0_Init()`
- ◆ `TMP0_User_Init()`
- ◆ `TMP0_Start()`

The 16-bit timer, P, provides the interval (periodic) timer interrupt. The timer uses the internal clock, $f_{xx}/64$, for input. Operating as an interval timer, P restarts once it reaches the set count. When the main function calls `TMP0_Start`, the timer starts.

2.2.6.2 Timer Interface

- ◆ `MD_INTTP0CC0()`
- ◆ `SetMsecTimer()`
- ◆ `CheckMsecTimer()`
- ◆ `delay()`

Operations that are performed periodically are:

- ◆ LED multiplexing
- ◆ Switch-input monitoring
- ◆ Delay-count service

The timer-counter value increments until it matches the interval-count register, then the timer-counter generates an interrupt. Program execution vectors to the interrupt-service routine at MD_INTTP0CC0. The interrupt-service routine calls `sw_isr()` and `led_mux_drive()`.

2.2.7 Port Functions (Including Switch Input and LED Output)

Appendix A provides flowcharts for the `SPI_4.0_port_interface`, `SPI_4.1_led_interface` and `SPI_4.2_switch_interface`. Appendix B provides source code for `port.c`, `led_vjj2.c` and `sw_vjj2.c`.

2.2.8 Port Initialization

◆ `PORT_Init()`

Port initialization consists of setting the registers to their default values and, for each port, setting the function, mode and mode-control registers. Setting the default port value only has meaning if the port bit is set for output.

2.2.8.1 LED Driver

- ◆ `dump_led_digit()`
- ◆ `led_init()`
- ◆ `led_out_digit1()`, `led_out_digit2()`, `led_out_digit3()`, `led_out_digit4()`
- ◆ `led_dp_digit1()`, `led_dp_digit2()`, `led_dp_digit3()`, `led_dp_digit4()`
- ◆ `led_L1()`, `led_L2()`, `led_L3()`
- ◆ `led_colon()`
- ◆ `led_num_digit1()`, `led_num_digit2()`, `led_num_digit3()`, `led_num_digit4()`
- ◆ `led_hex()`
- ◆ `led_dig_bcd()`
- ◆ `led_mux_drv()`

The LITEON LTC-4627JR 7-segment, 4-digit LED unit on the demo board is a common-anode, multiplexed device. Thus, the segments are connected in parallel, and all anodes that make up a digit connect together. To display a particular segment, a ONE must be output on the port driving the segment, and a ZERO must be output on the port sinking the current.

Data to be displayed on the LEDs is maintained in array `led_digit[]`. Various functions are provided to modify the digits, either raw or decoded. Other functions are provided to turn decimal points and colon on or off, and to control other special LEDs, designated L1, L2 and L3. Special functions convert and display either hex values or BCD (binary coded decimal).

The routine `led_mux_drv()` performs the actual display. The interval timer periodically calls this routine to display the next digit. The routine does this by first turning off all of the sink lines in port P6. The routine then reads the value to be displayed from the array `led_digit[]` using the current index set by the periodic interrupt, and outputs that data to the lower part of port 9 P9L. The routine then selects the digit by pulling the appropriate bit in port P6 LOW.

2.2.8.2 Switch-Input Interface

- ◆ `sw_init()`
- ◆ `sw_chk()`
- ◆ `sw_set_debounce()`
- ◆ `sw_get()`
- ◆ `sw_isr()`

Mechanical switches can make and break many times as they touch together or separate. Debouncing prevents processing every switch contact change by making the switch act more like a state change. Thus, debouncing builds in some hysteresis. A debounced switch must remain open or closed for a specified number of samples before the state of the switch is declared open or closed, thus eliminating bounce.

3. Applilet Selections

The Applilet tool allows the selection and building of a basic code framework for beginning your development project.

The following selections in the Applilet tool are made for the initial code generation.

Figure 8. System-Foundation Settings

System

Foundation setting | Startup setting

Clock generator operation mode
☒ Main clock operation mode
☐ Sub-Clock operation mode

Sub-clock setting
☒ Use
☐ Unused

Main clock setting
☒ Use
☐ Unused

PLL function setting
☒ PLL function ON
☐ PLL function OFF

System clock control in PLL mode
☒ fxx = 4*fx
☐ fxx = 8*fx

Enable watchdogtimer2
☐ Enable watchdogtimer 2
☒ Unused/disable watchdogtimer 2

Internal oscillator setting
☐ Disable internal oscillator

PLL lock setting
 PLL lock time selection(us) 1630

Oscillator frequency setting
 Main oscillator(MHz) 5
 Internal oscillator(KHz) 200
 Sub-resonator(KHz) 32.768

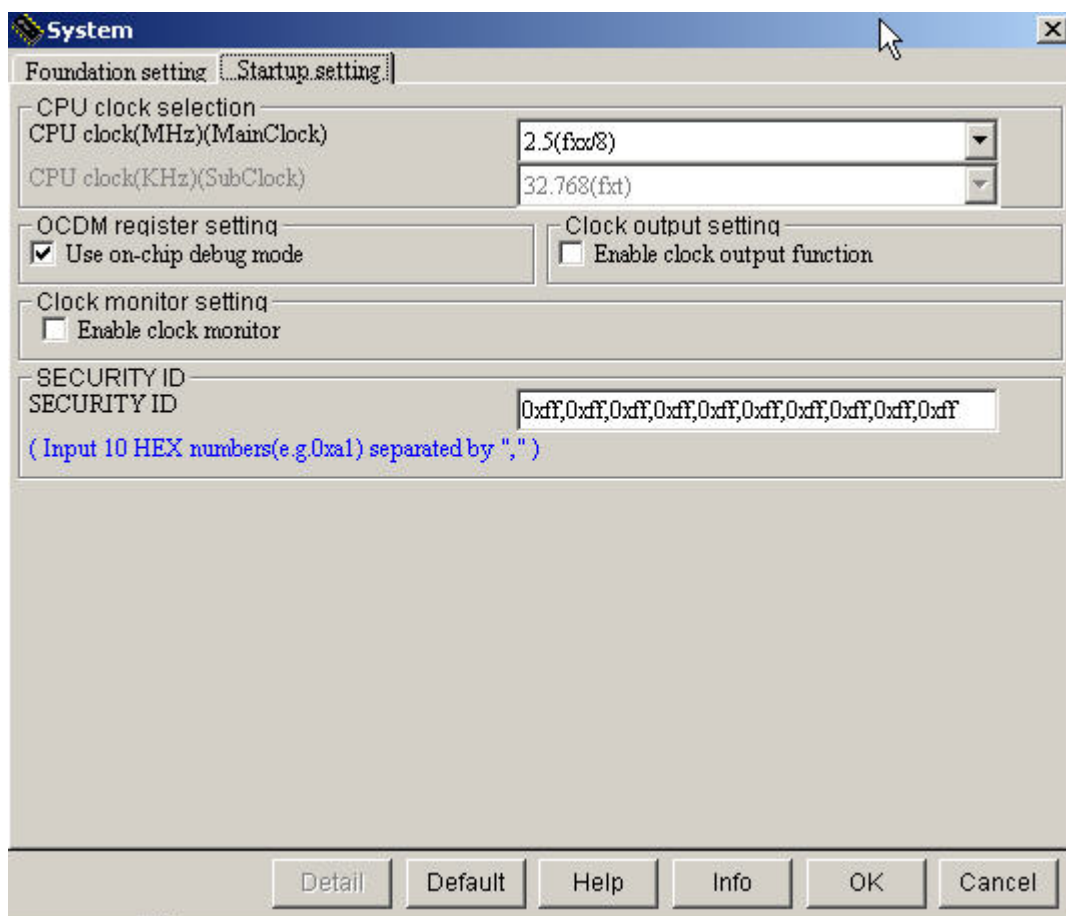
Oscillation stabilization time
 Stable time(ms) 13.11

Detail Default Help Info OK Cancel

The main clock is used because the demo board provides an external 5-MHz crystal. The PLL function is turned ON to multiply the 5-MHz clock, $f_{xx} = 4 \times f_x$, providing a 20-MHz system clock. The demo does not use watchdog timer 2.

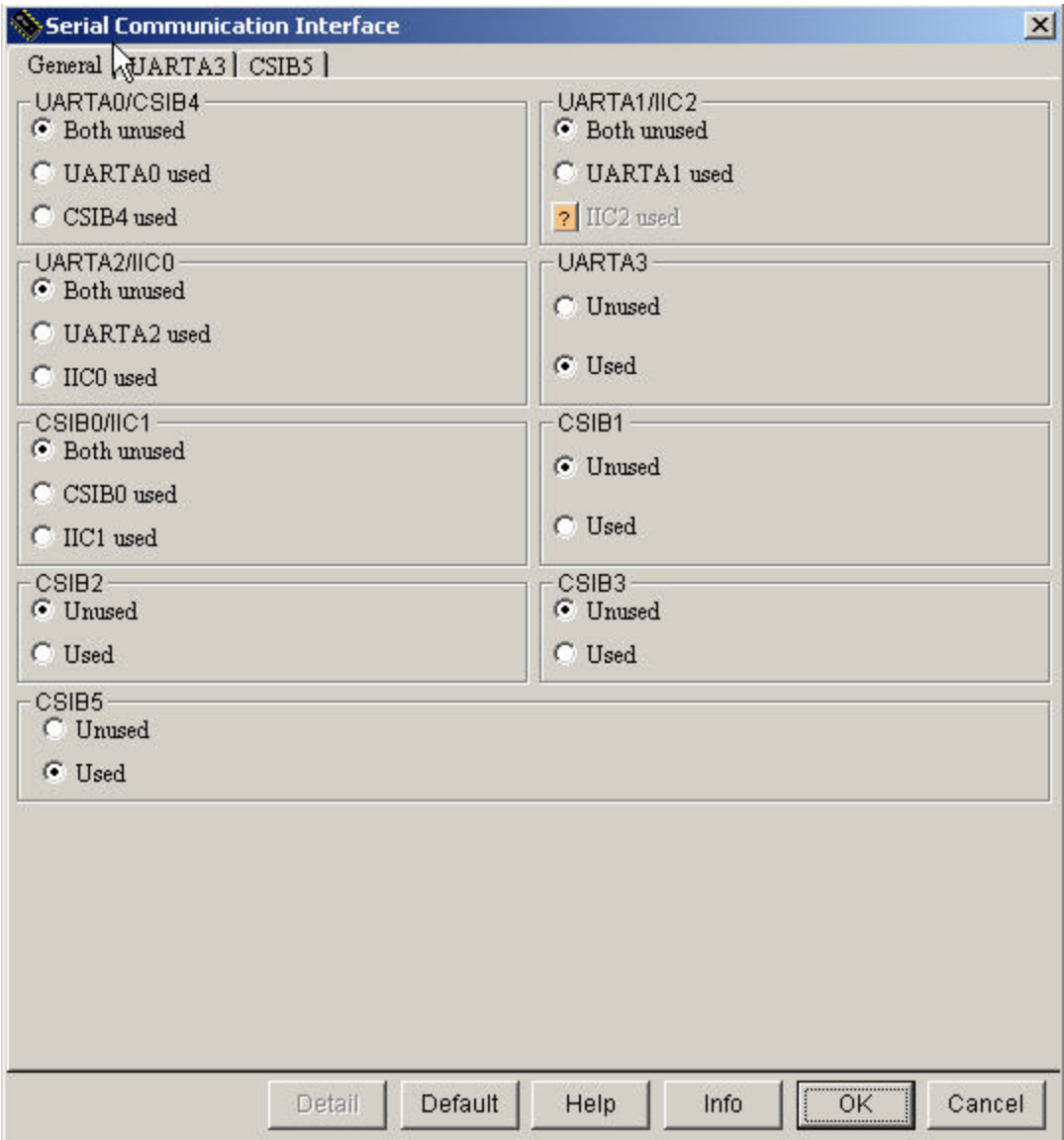
The next screen capture shows the CPU clock selection. This screen also shows the selection of on-chip debug mode. Leave the security ID at the default values of 0xff.

Figure 9. System Startup Settings



The next screen shows how you pick the serial devices you wish to use. The demo uses UART3 and CSIB5, as the demo board was built to use these interfaces.

Figure 10. Serial-Communication Interface Selections



3.1 Configuring Applilet for CSI (CS15)

This application involves sending and receiving, with the data format of 8 bits, MSB, single transfer, Type 1 for SPI. This interface is the master, so it supplies the clock. Using a slower speed avoid problems. Interrupts are not used.

Figure 11. Serial-Communications CS1B5 Settings

The screenshot shows the 'Serial Communication Interface' dialog box with the 'CS1B5' tab selected. The dialog is divided into several sections with various settings:

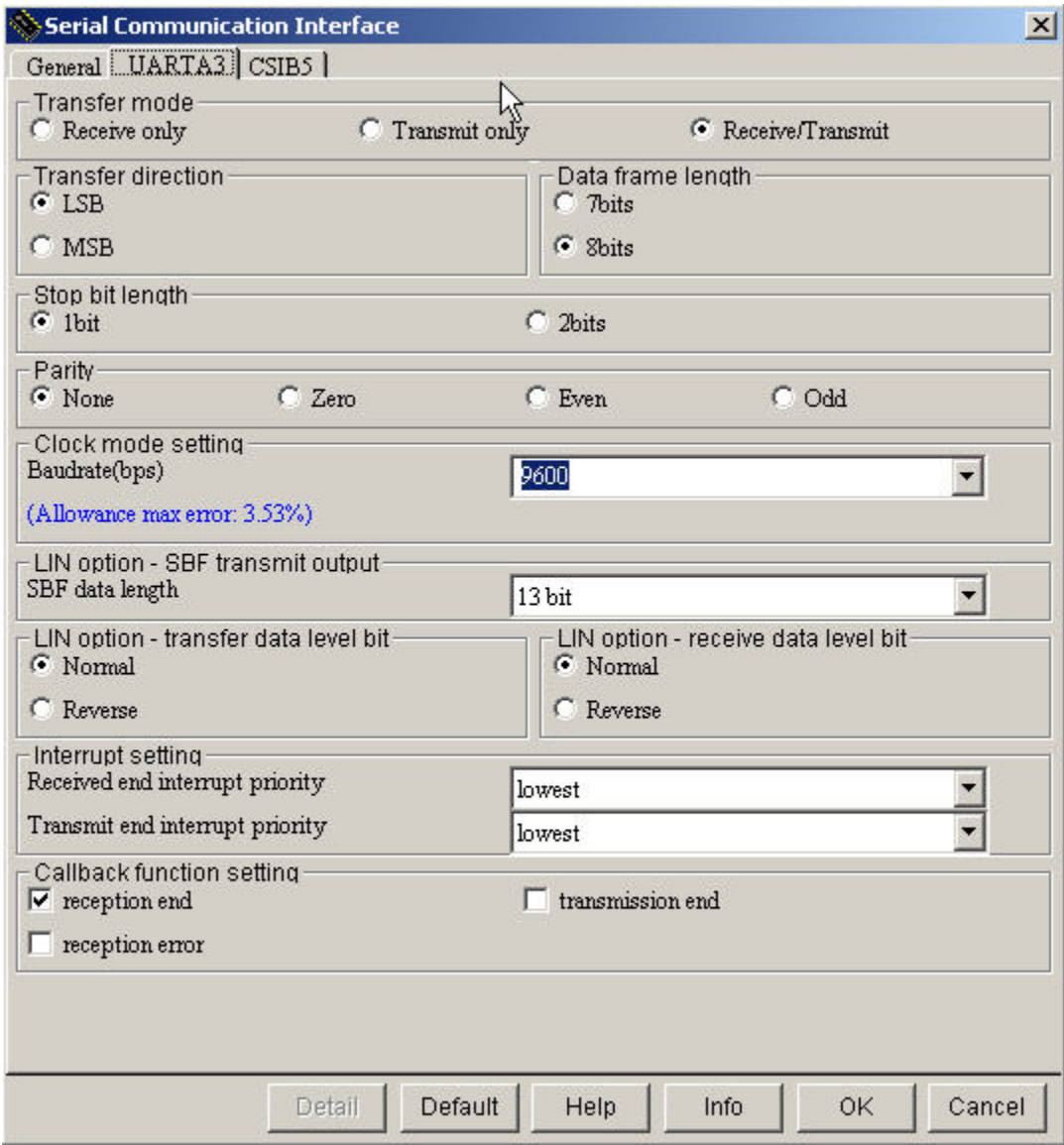
- General:** Includes tabs for 'General', 'UARTA3', and 'CS1B5'.
- Transfer mode:**
 - ☒ Enables reception
 - ☒ Enables transmission
- Data length:**
 - Data length: 8 bits (dropdown menu)
- Data direction:**
 - ☒ MSB
 - ☐ LSB
- Specifies transfer mode:**
 - ☒ Single transfer mode
 - ☐ Consecutive transfer mode
- Specification of serial clock and data phase:**
 - ☒ Type 1
 - ☐ Type 2
 - ☐ Type 3
 - ☐ Type 4
- Clock mode setting:**
 - Transmission/reception timing: Internal clock - master mode (dropdown menu)
 - Baudrate(bps): 620000 (dropdown menu)
 - fbrgm baudrate(bps): 921600 (dropdown menu)
 - (Allowance max error: 3.53%)
- Interrupt setting:**
 - Reception completion interrupt priority: level2 (dropdown menu)
 - Transmission write enable interrupt priority: lowest (dropdown menu)
- Callback function setting:**
 - ☐ Reception end
 - ☒ Transmission end
 - ☐ Overrun error

At the bottom of the dialog are buttons for 'Detail', 'Default', 'Help', 'Info', 'OK', and 'Cancel'.

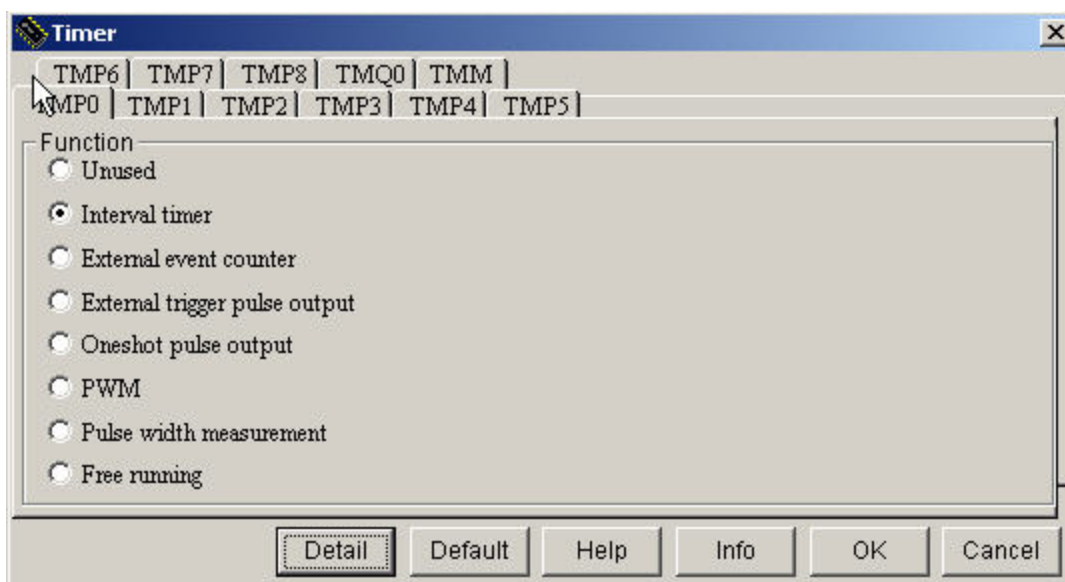
3.2 Configuring Applilet for UART3

The following settings are chosen for normal asynchronous serial communications.

Figure 12. Serial-Communications Interface UARTA3



The demo uses the 16-bit timer TMP0 as an interval timer.

Figure 13. Configuring Applilet for Timer 00 (TMP0)

Set up the interval timer as shown below. Since the timer is used for a human interface (switch checking and LED multiplexing), a rather long interval should be used. The interrupt can be the lowest priority because these functions are the least important.

Figure 14. Setting Up Interval Timer

Interval timer

Count clock

☐ Auto

☐ fix/2

☐ fix/8

☐ fix/32

☐ fix/128

☐ TIP00 rising edge

☐ fix

☐ fix/4

☐ fix/16

☒ fix/64

☐ TIP00 falling edge

☐ TIP00 both edge

Ext clock(KHz) 100

Value scale

Value scale msec

Interval timer

Interval value0 100

CCR1 setting Not Used

Interval value1 50

Output setting

☐ TOP00 pin output enable

TOP00 pin output level setting Output normal

☒ TOP01 pin output enable

TOP01 pin output level setting Output normal

Interrupt setting

☒ TMP0 and CCR0 match, generate an interrupt(INTTP0CC0)

Priority lowest

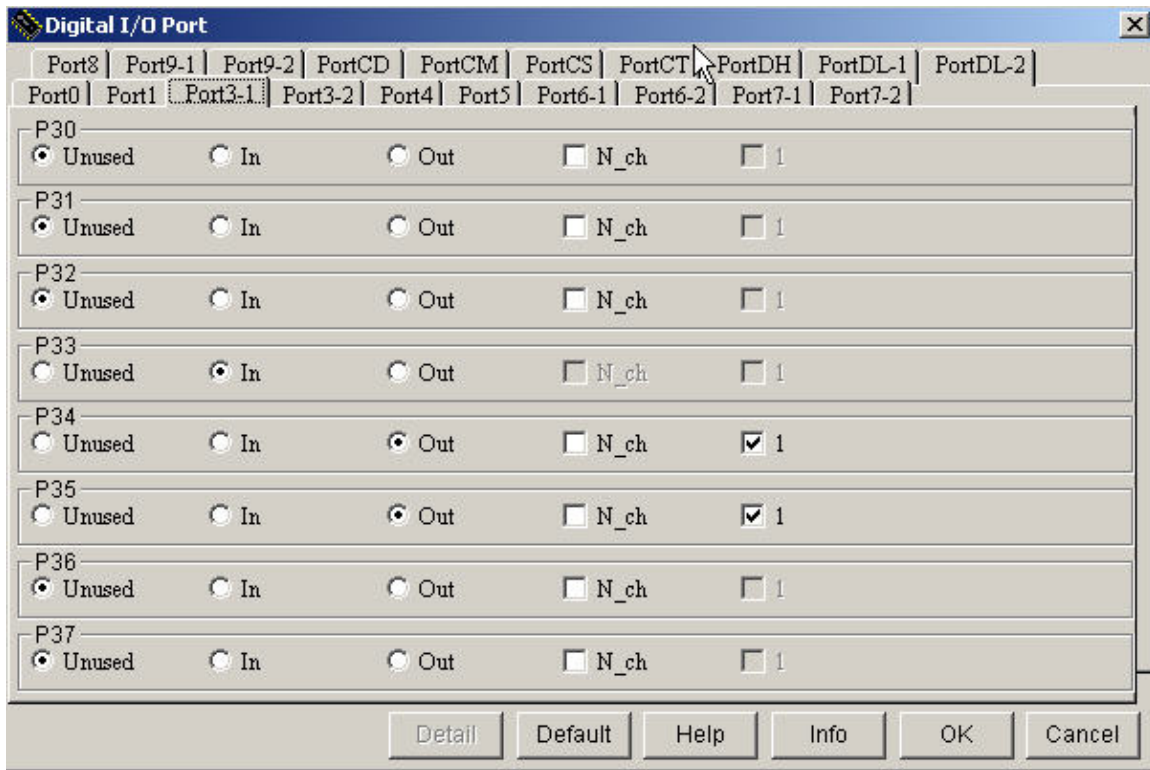
☐ TMP0 and CCR1 match, generate an interrupt(INTTP0CC1)

Priority lowest

Help Info OK Cancel

When setting up the I/O ports (shown below), Port 3, bit 4, is the SPI chip select for a standard Zigbee interface. Port 3, bit 5, is the chip select for the SD memory card.

Figure 15. Configuring Applilet for I/O Ports

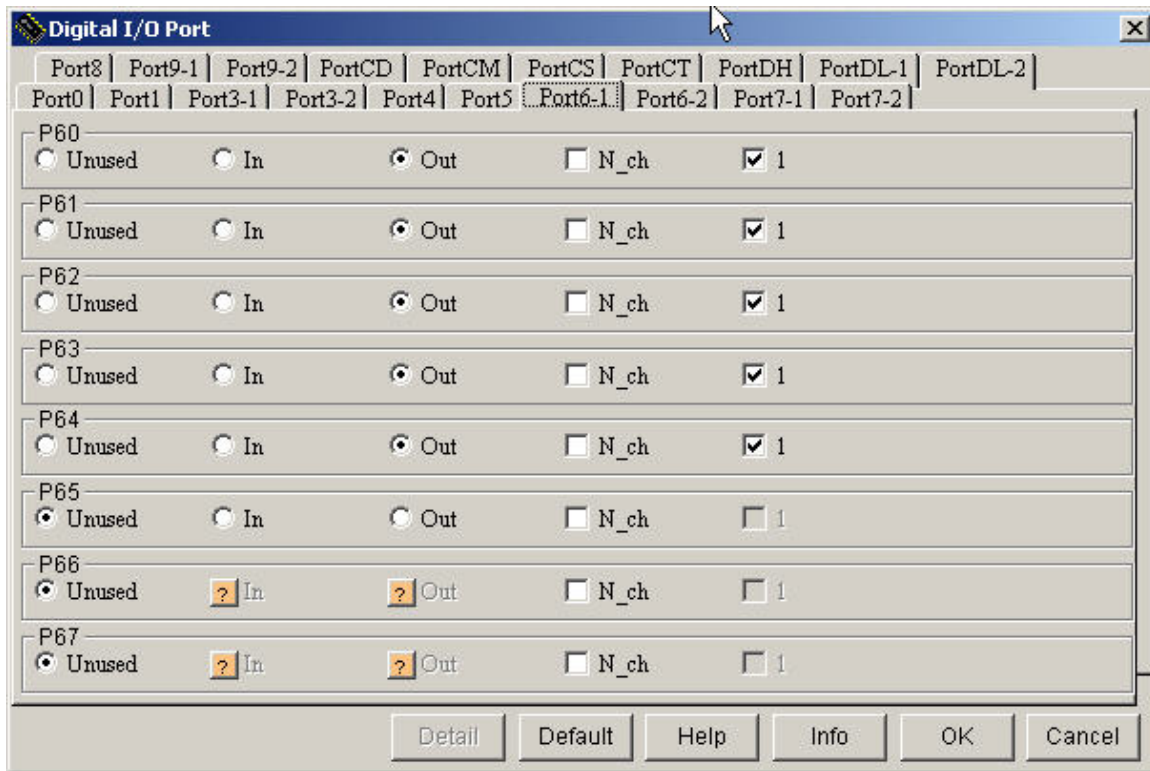


For controlling the LEDs (as shown below), the Port 6 bits are as follows:

- ◆ Bit 0 is digit 1, the common anode.
- ◆ Bit 1 is digit 2.
- ◆ Bit 2 is digit 3.
- ◆ Bit 3 is digit 4.
- ◆ Bit 4 is the colon and top dot.

Initialize these bits to ONE, which sets the off state.

Figure 16. Configuring Port 6



For further LED setup, the Port 9 bits are:

- ◆ Bit 0 is segment A and L1.
- ◆ Bit 1 is segment B and L2.
- ◆ Bit 2 is segment C and L3.
- ◆ Bit 3 is segment D.
- ◆ Bit 4 is segment E.
- ◆ Bit 5 is segment F.
- ◆ Bit 6 is segment G.
- ◆ Bit 7 is the decimal point.

Initialize these bits to ZERO, not ONE, as shown in the figure below.

Figure 17. Configuring Port 9

Digital I/O Port

Port0	Port1	Port3-1	Port3-2	Port4	Port5	Port6-1	Port6-2	Port7-1	Port7-2
Port8	Port9-1	Port9-2	PortCD	PortCM	PortCS	PortCT	PortDH	PortDL-1	PortDL-2

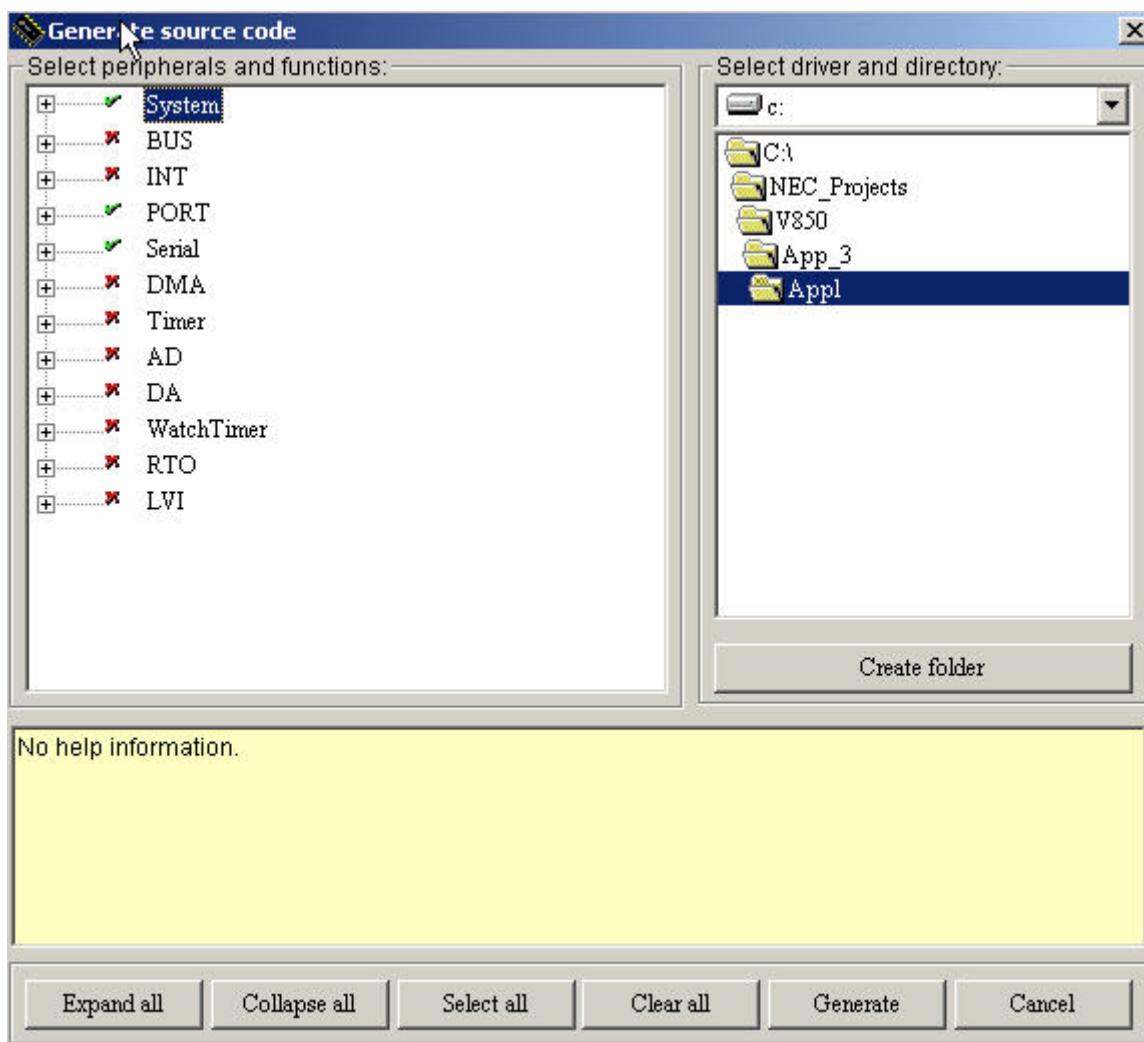
Port	Unused	In	Out	N_ch	1
P90	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
P91	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
P92	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
P93	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
P94	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
P95	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
P96	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
P97	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Detail Default Help Info OK Cancel

3.3 Generating Code with Applilet

After making the selections described above, you are ready to generate your base code. Press the **generate** button to create the code for the selected peripherals and system initialization.

Figure 18. Generating Code with Applilet



The list of files generated by the Applilet appears below. Double-click on the SDmem.prw file to bring up the project manager.

Figure 19. Files Generated by Applilet



When opened, the project manager asks you to select which tools it should use, as shown in the example below.

Figure 20. Selecting Tools in Project Manager

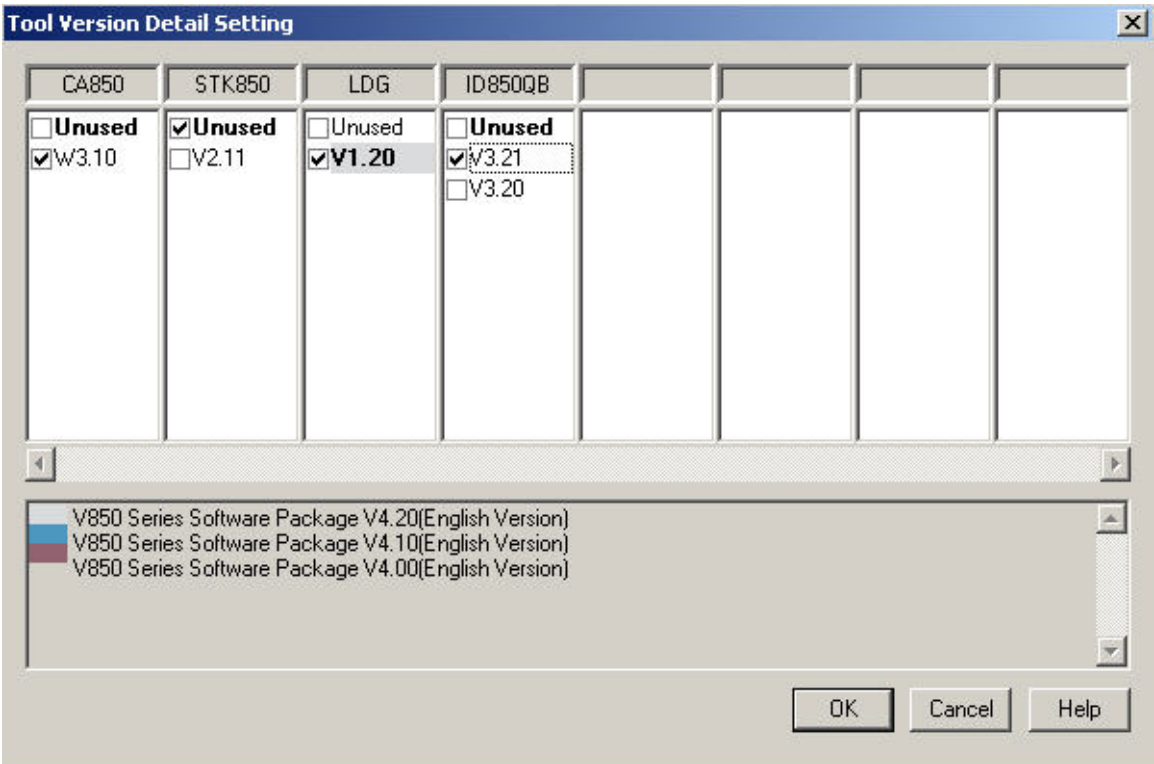


Figure 21. Tool Set in Applilet

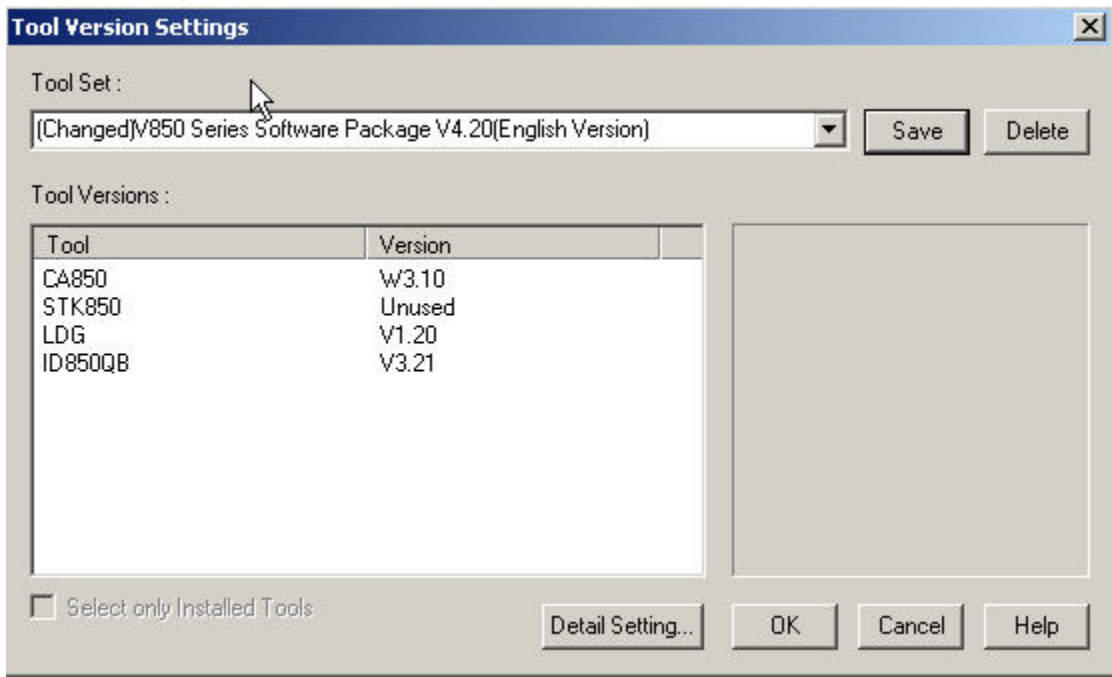
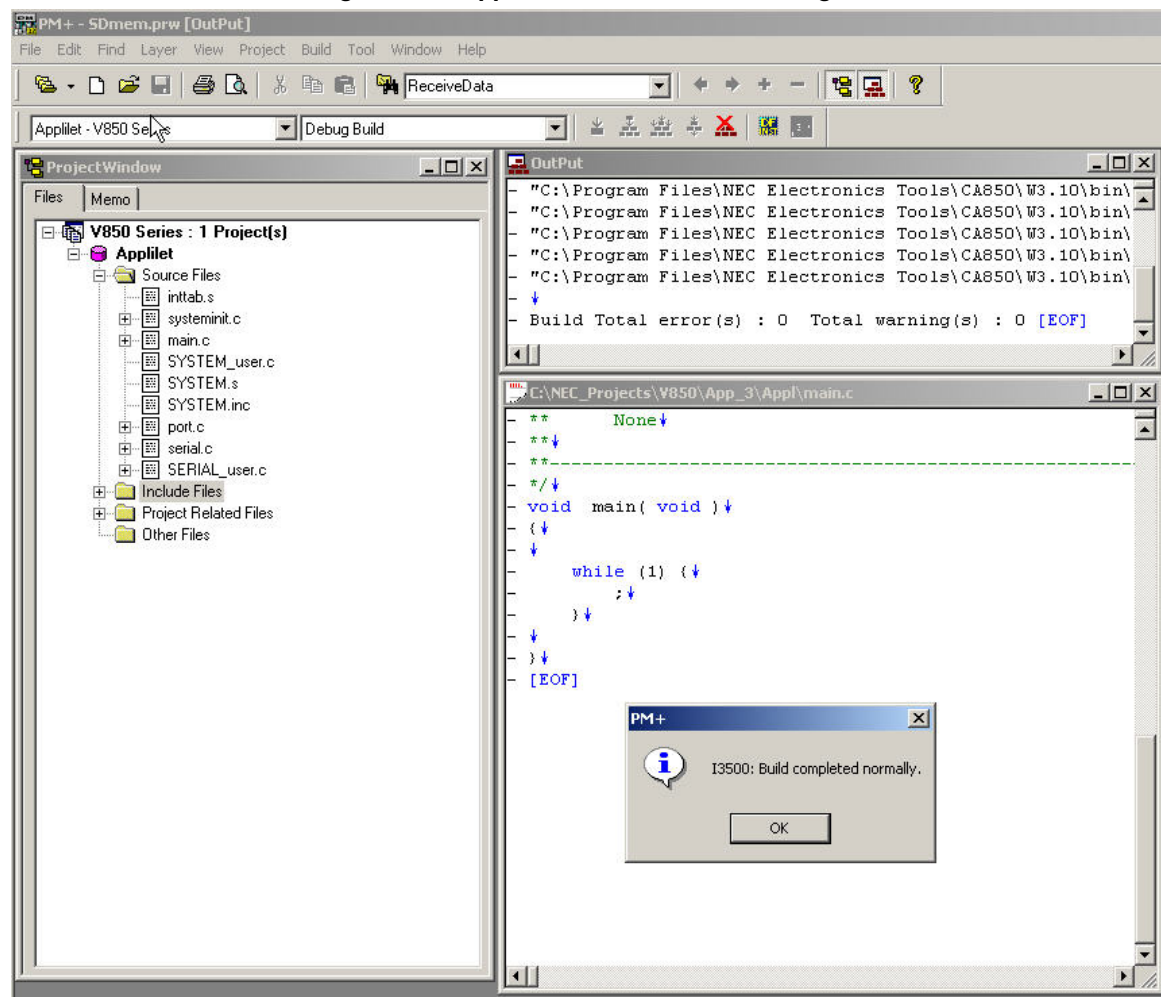


Figure 22. Applilet Interface After Selecting “Build”



The following screen capture shows the program, downloaded via debugger and MiniCube2, running on the demo board.

Figure 23. Fig. 23 Screen Capture of Program Running on Demo Board

```

ID850QB : mdt.prj
File Edit View Option Run Event Browse Jump Window Help

Source (main.c)
Search... << >> Watch Quick... Refresh Close

65 void main( void )
66 {
67     char msg_n1[] = {"\r\n"};
68     const char msg1[] = {"\r\n\r\nSD Memory Demonstration program  <
69     const char msg2[] = {"\r\n\r\nDemonstrating reading and writing of SD/M
70     const char msg3[] = {"\r\n\r\n enter sector number to use (00-99) "};
71     const char msg4[] = {"\r\n\r\n enter text to be written to SD memory on
72     const char msg5[] = {"\r\n\r\n (512 character per sector limit)\r\n\r\n
73     const char msg6[] = {"\r\n\r\n press SW1 to write data to SD memory\r\n\r\n
74     const char msg7[] = {"\r\n\r\n press SW2 to read data back from SD memor
75     const char msg_sdmr[] = {"\r\n\r\nSDmemory_Read sector %d\r\n\r\n"};
76     const char msg_sdmw[] = {"\r\n\r\nSDmemory_Write sector %d\r\n\r\n"};
77     const char msg_a[] = {"main - memory card init status 0x%02x\r\n\r\n"};
78     const char msg_b[] = {"main - memory card specific data request sta
79     const char msg_c[] = {"main - memory card identification data requ
80     const char msg_d[] = {"main - sector %d read status 0x%02x\r\n\r\n"};
81     const char msg_e[] = {"main - sector %d write status 0x%02x\r\n\r\n"};
82     char msg_buf[120];
83
84     MD_STATUS status;
85     MD_STATUS mem_stat, mem_stat9, mem_stat10;
86     unsigned char sw_val;
87     UCHAR data[20];
88     int err,i;
89     int line;
90     int sector;
91     USHORT size, done, SD_status;
92     unsigned char buffer1[BUF_SIZE];
93     unsigned char buffer2[BUF_SIZE];
94
95     WT_Start(); /* watch timer start up */
96     sw_init(); /* initialize switch variables */
97     led_init(); /* initialize LED display */
98     TMP0_Start(); /* start timer for switch debouncing, led mux
99     CSIB5_deselect_SPI(); /* bring all spi device select lines high */
100
101
102     delay(250); /* the setup of uart3 can put glitches on the lin
103                /* allow some time for it to settle before output
104
105     uart3_tx_msg((char *)msg1);
106     uart3_tx_msg((char *)msg2);
107     uart3_tx_msg((char *)msg4);
108     uart3_tx_msg((char *)msg5);
109     uart3_tx_msg((char *)msg6);
110     uart3_tx_msg((char *)msg7);
111
112     CSIB5_send_done = 0;
113
114     mem_stat = SDmemory_Init(); /* initialize SD memory access */
115     if(mem_stat != MD_OK) {
116         sprintf(msg_buf, msg_a, mem_stat); /* dbg SD memory init sta
117         uart3_tx_msg(msg_buf); /* dbg */
118         /* consider abort message and hang */
119     }

```

main.c#66 main 00000004 RUN

4. Demonstration Platform

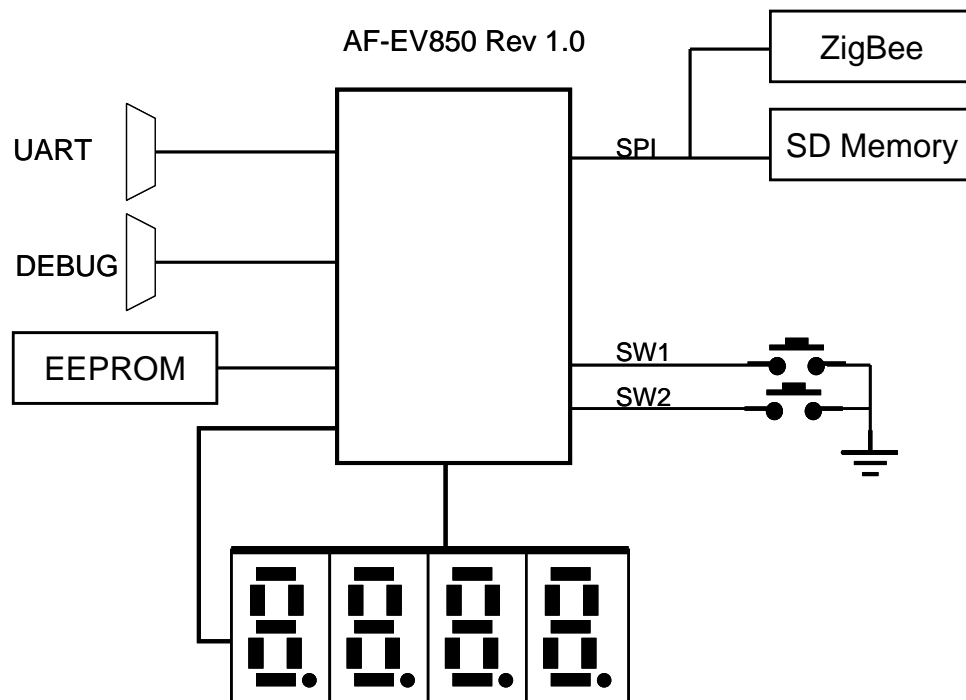
The demonstration uses a development board from NEC Electronics. You may be able to duplicate the same hardware using off-the-shelf components along with the NEC Electronics microcontroller of interest.

Demo board features:

- ◆ NEC Electronics' 850ES/JJ2 D70F3721GJ microcontroller
- ◆ USB FTDI chip
- ◆ MAXIM MAX232 RS232 driver for UART
- ◆ 4-digit, 7-segment LED
- ◆ A/D potentiometer
- ◆ MiniCube2 interface

The following figure shows a block diagram of the board.

Figure 24. Demonstration Board Block Diagram



[illegible]

```
enter sector number to use (00-99) 10
```

SDmemory_Read sector 10

```
main - sector 10 read status 0x12
```

* * * * *

[illegible]

enter sector number to use (00-99)

6. Software Modules

The following files make up the software modules for the demonstration program. The table shows which files were generated by the Applilet and which of those needed modification to create the demonstration program. Appendix B contains the listings for these files.

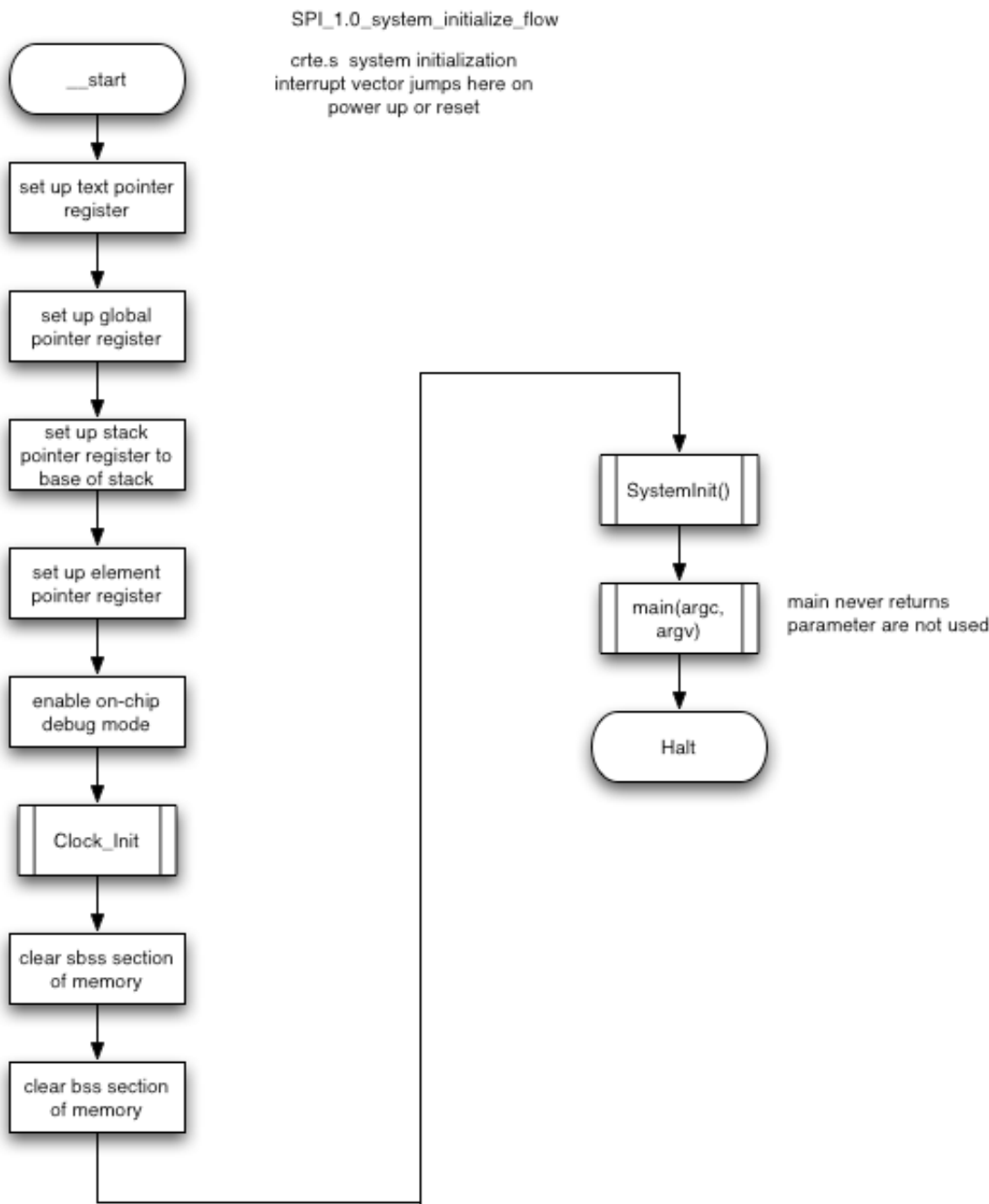
Table 4. Demonstration Program Software Modules

File	Generated by Applilet	Modified by User
crete.s	Applilet	modified
system.s	Applilet	
system.inc	Applilet	
inttab.s	Applilet	modified for MiniQube2
systeminit.c	Applilet	
macrodriver.h	Applilet	modified
main.c	Applilet	modified
sdmemory.c		
sdmemory.h		
serial.c	Applilet	modified
serial.h	Applilet	
port.c	Applilet	
port.h	Applilet	
led_vjj2.c		
led_vjj2.h		
sw_vjj2.c		
sw_vjj2.h		
timer.c	Applilet	
timer_user.c	Applilet	modified
timer.h	Applilet	

7. Appendix A — Flow Charts

7.1 System_initialize_flow

Figure 25. SPI_1.0_system_initialize_flow



7.2 Main_flow

Figure 26. SPI_1.1.0_main_flow

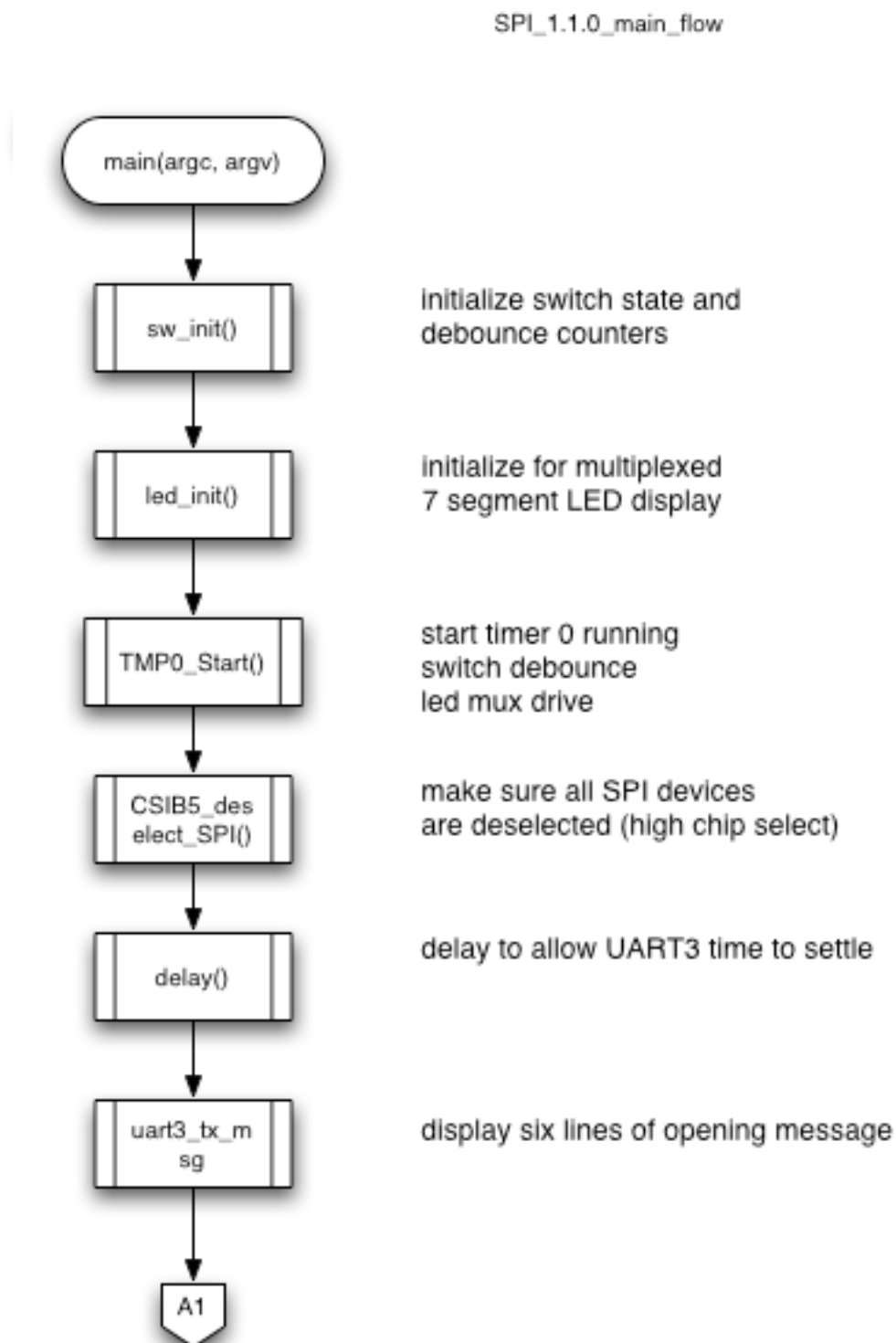


Figure 27. SPI_1.1.1_main_flow

SPI_1.1.1_main_flow

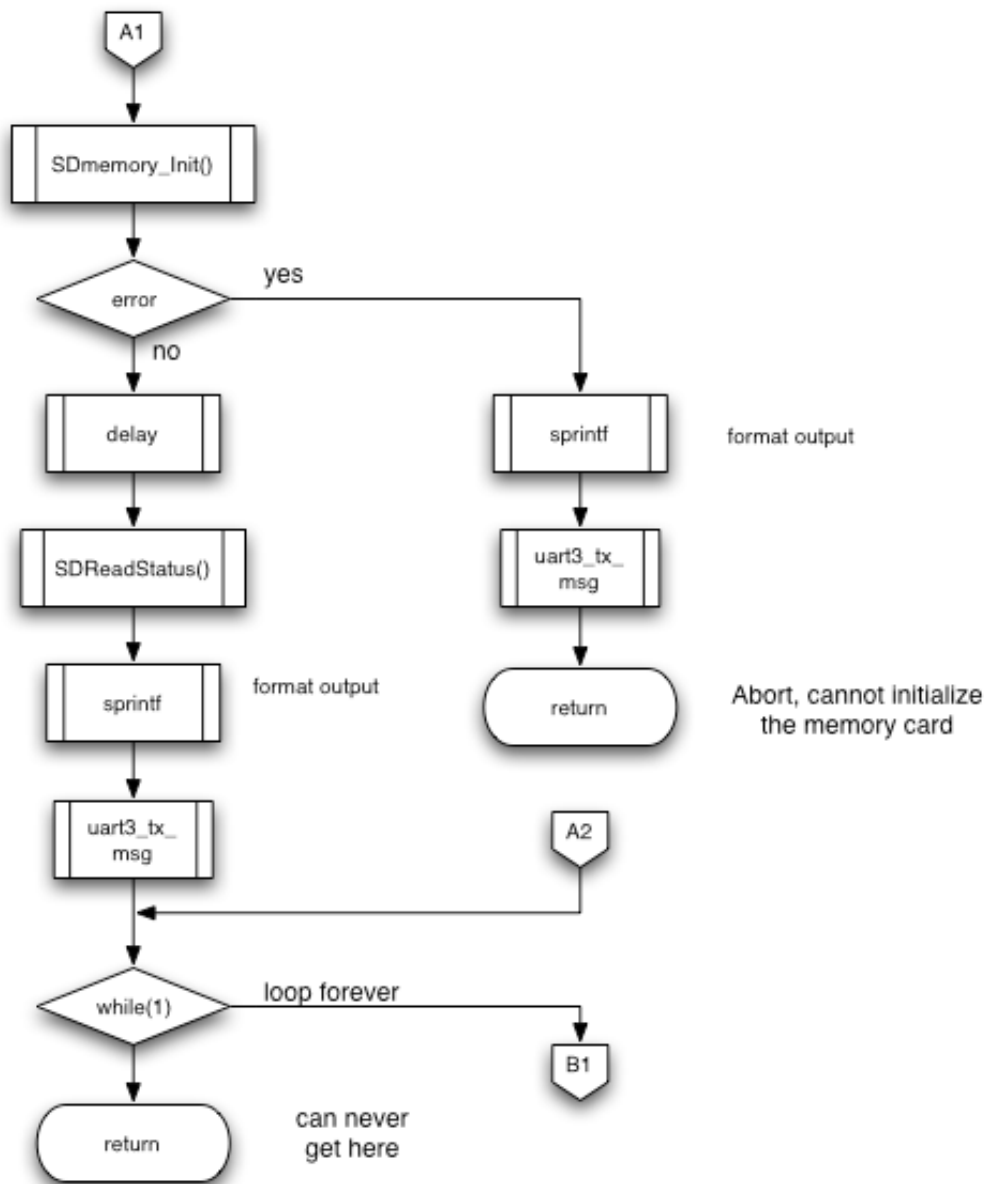


Figure 28. SPI_1.1.2_main_flow

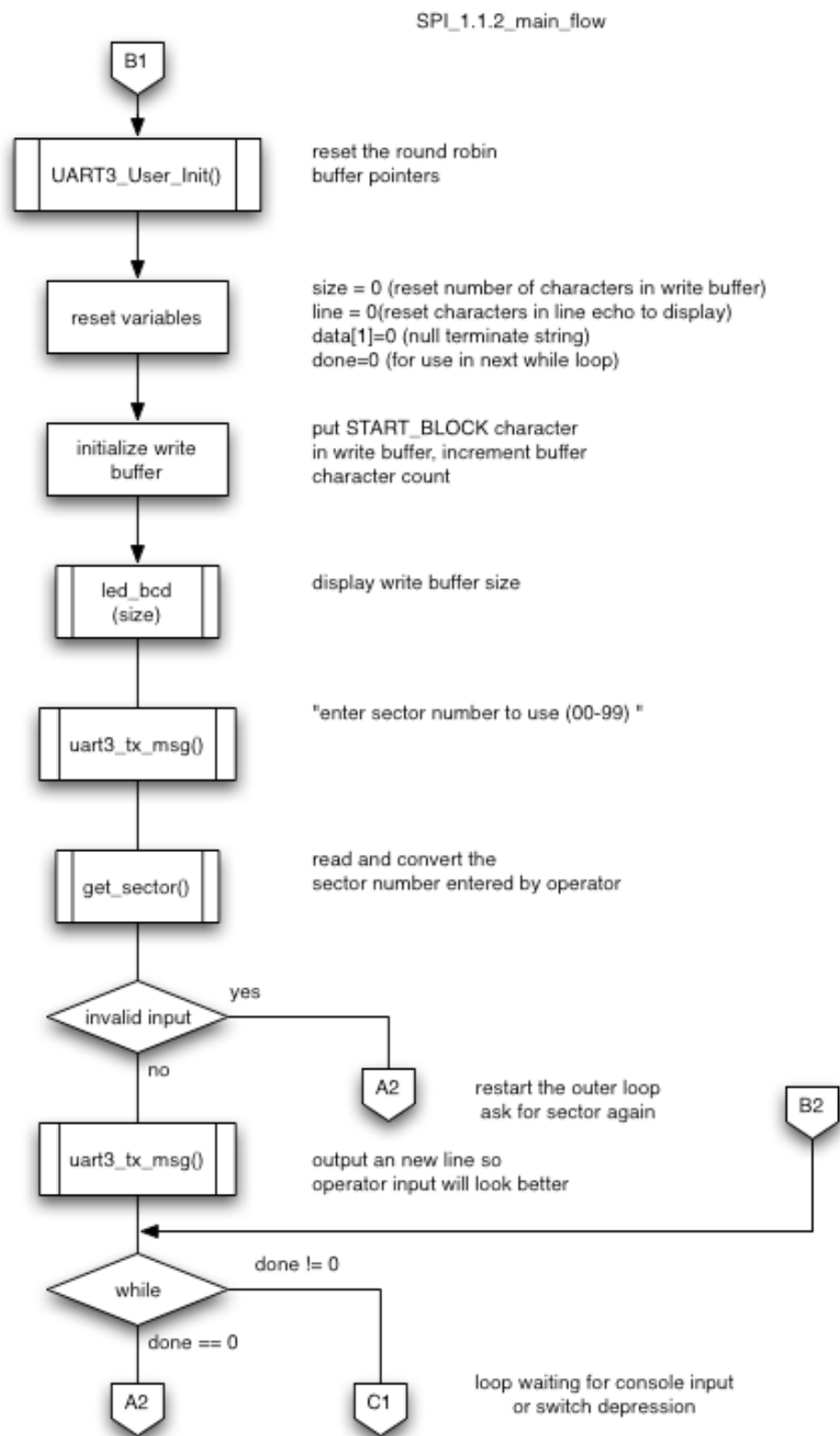


Figure 29. SPI_1.1.3_main_flow

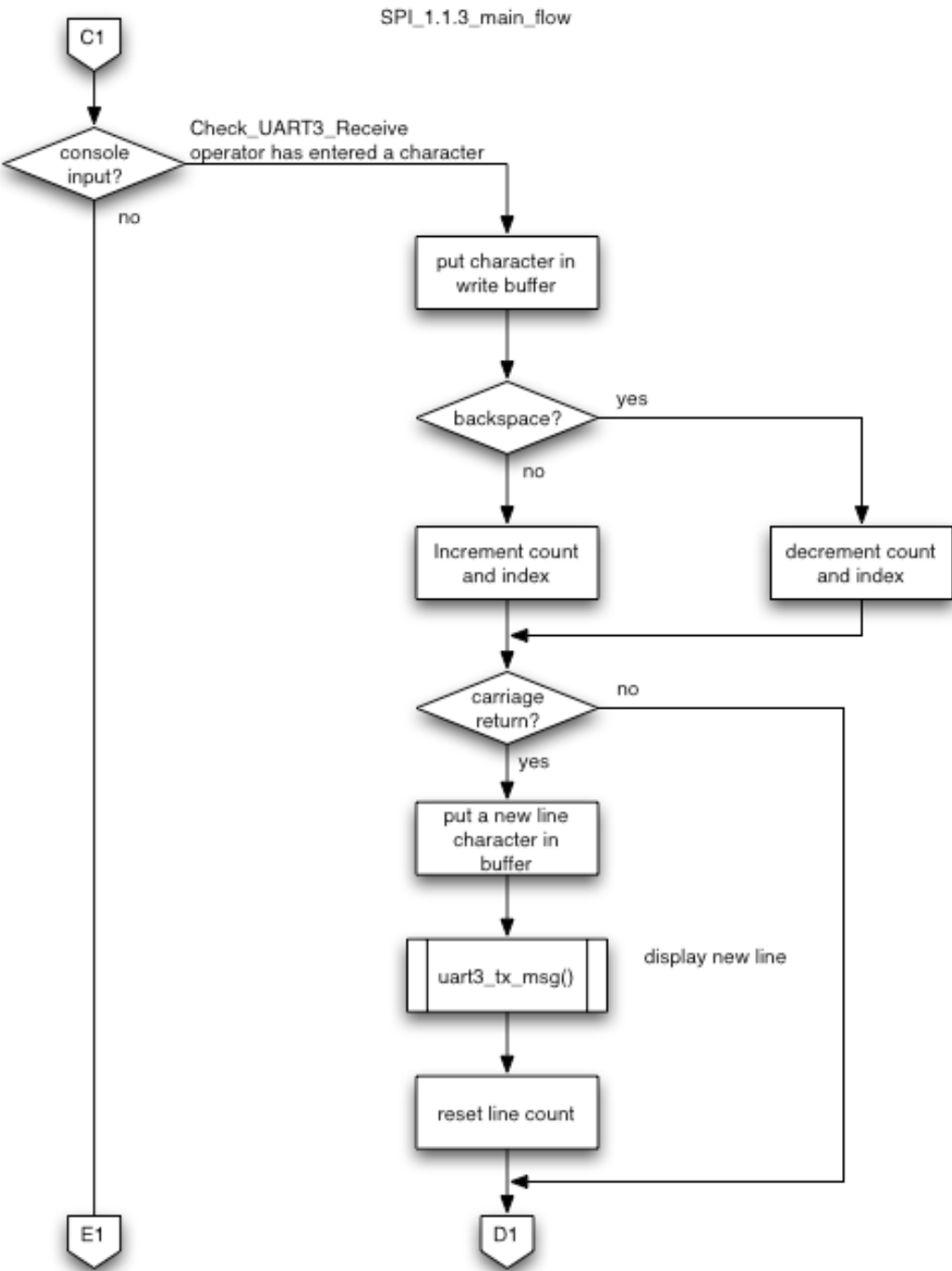


Figure 30. SPI_1.1.4_main_flow

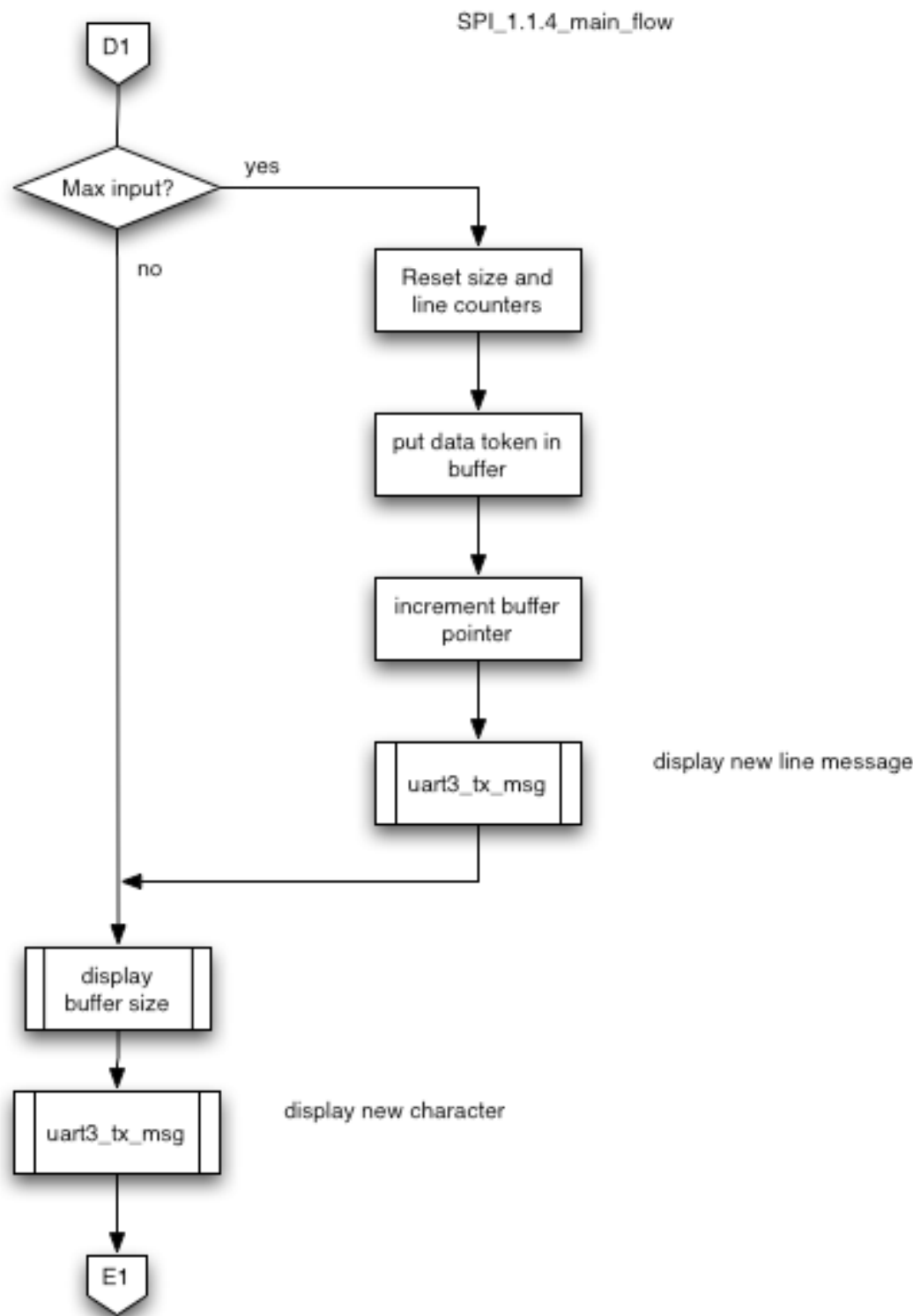


Figure 31. SPI_1.1.5_main_flow

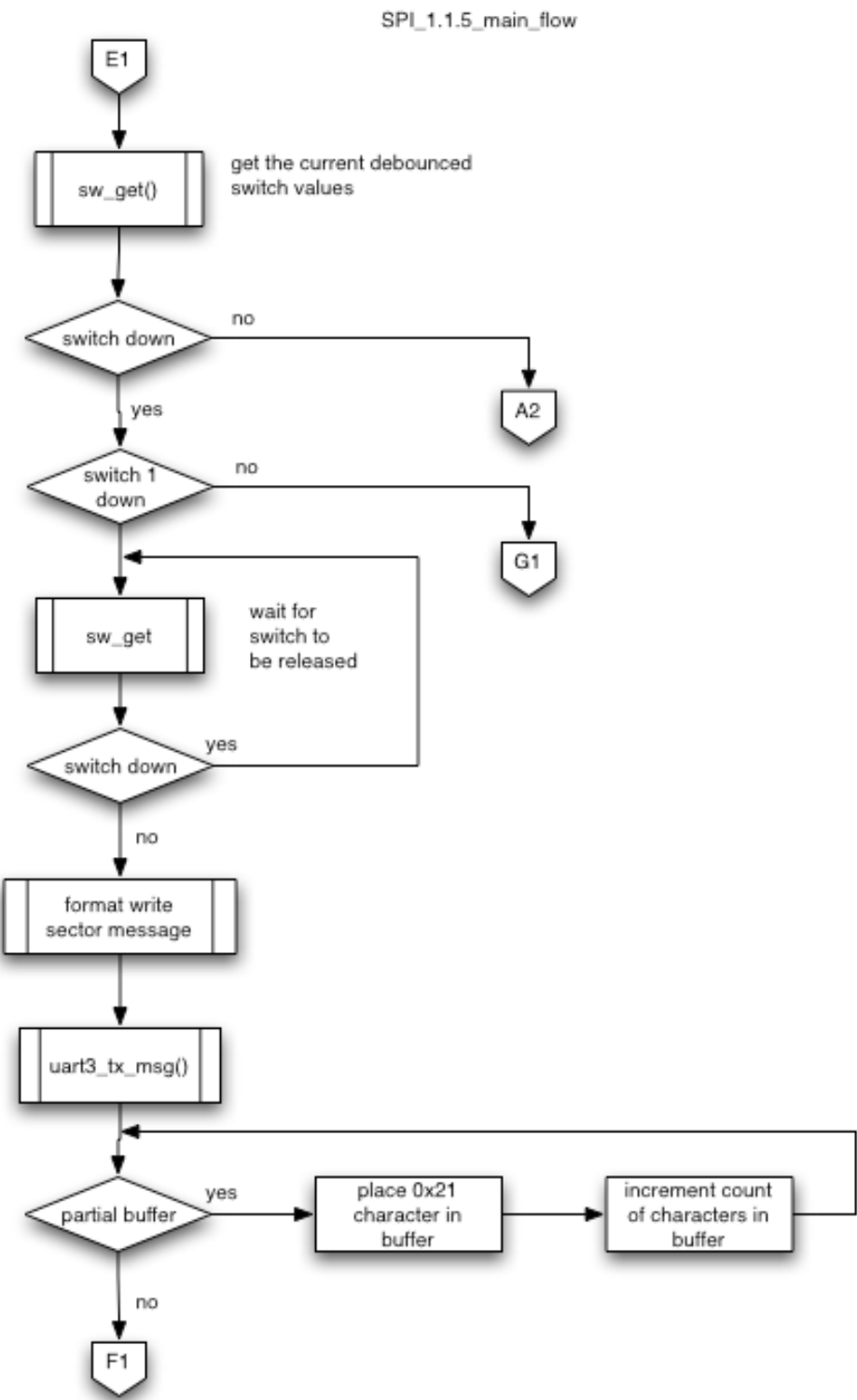


Figure 32. SPI_1.1.6_main_flow

SPI_1.1.6_main_flow

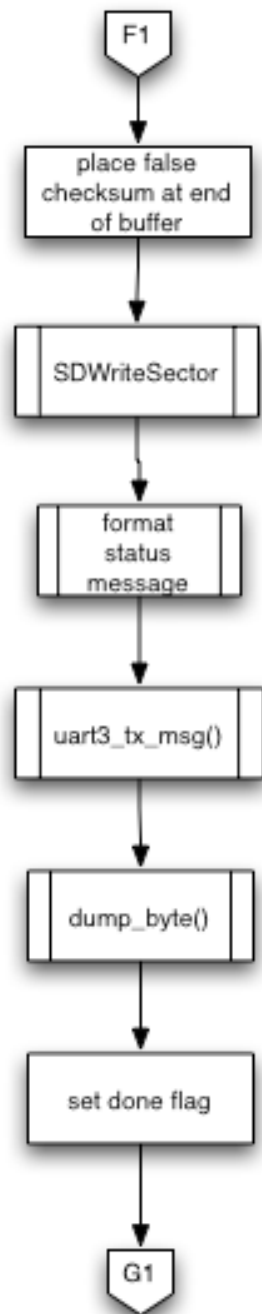


Figure 33. SPI_1.1.7_main_flow

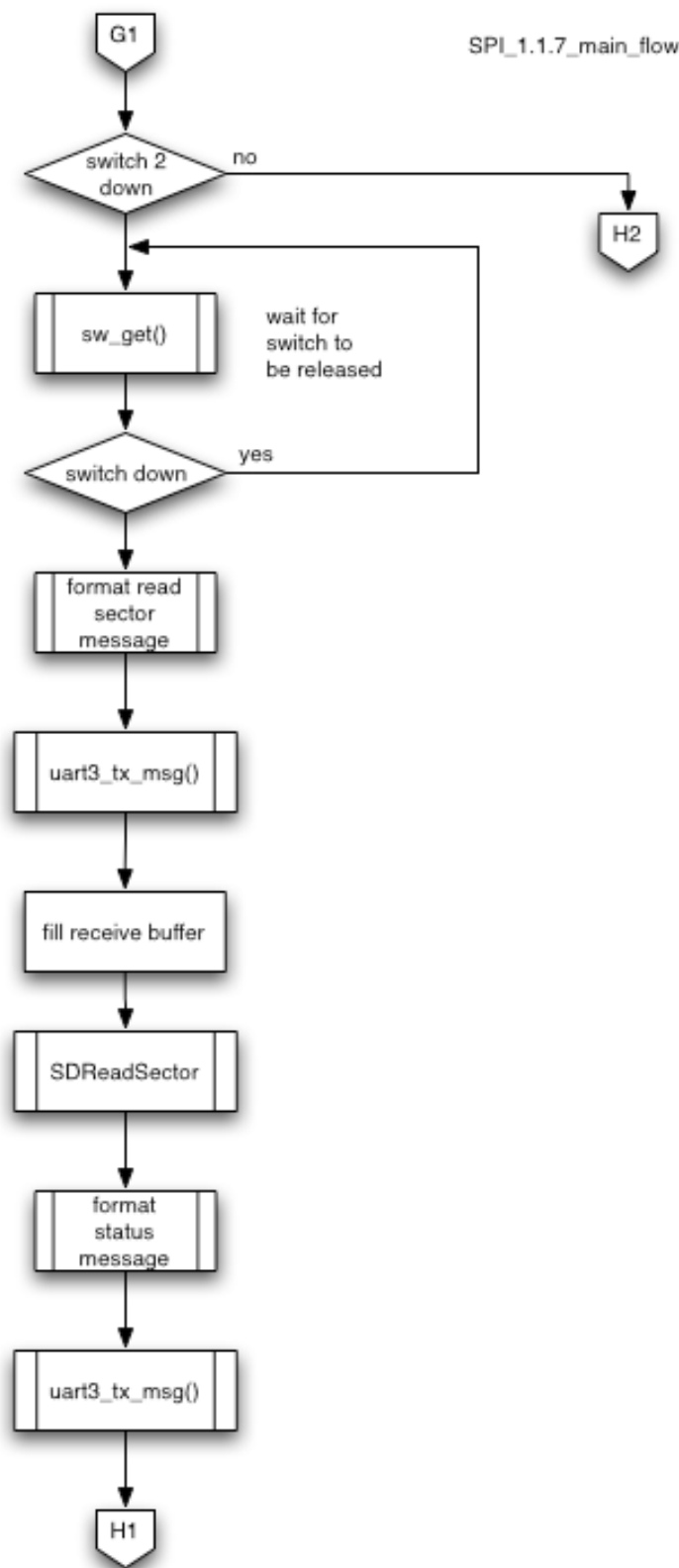


Figure 34. PI_1.1.8_main_flow

SPI_1.1.8_main_flow

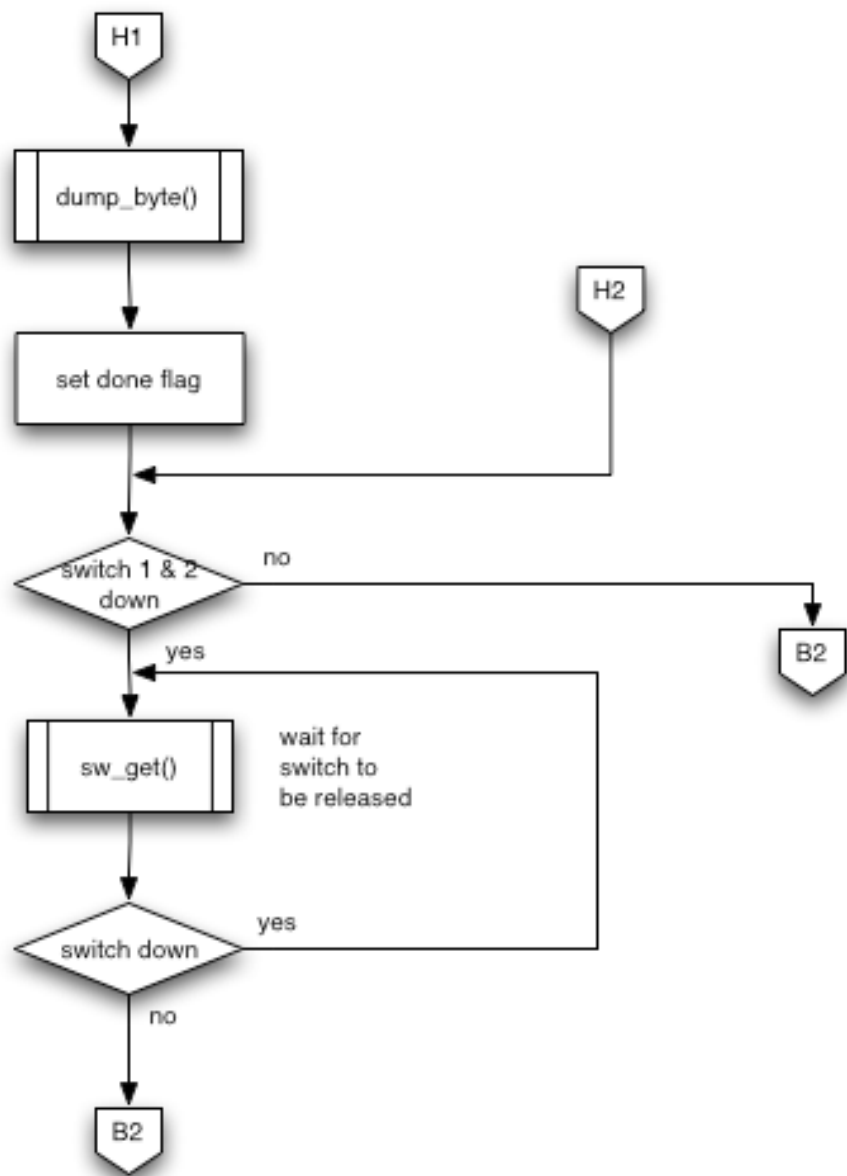


Figure 35. SPI_1.1.9_main_flow

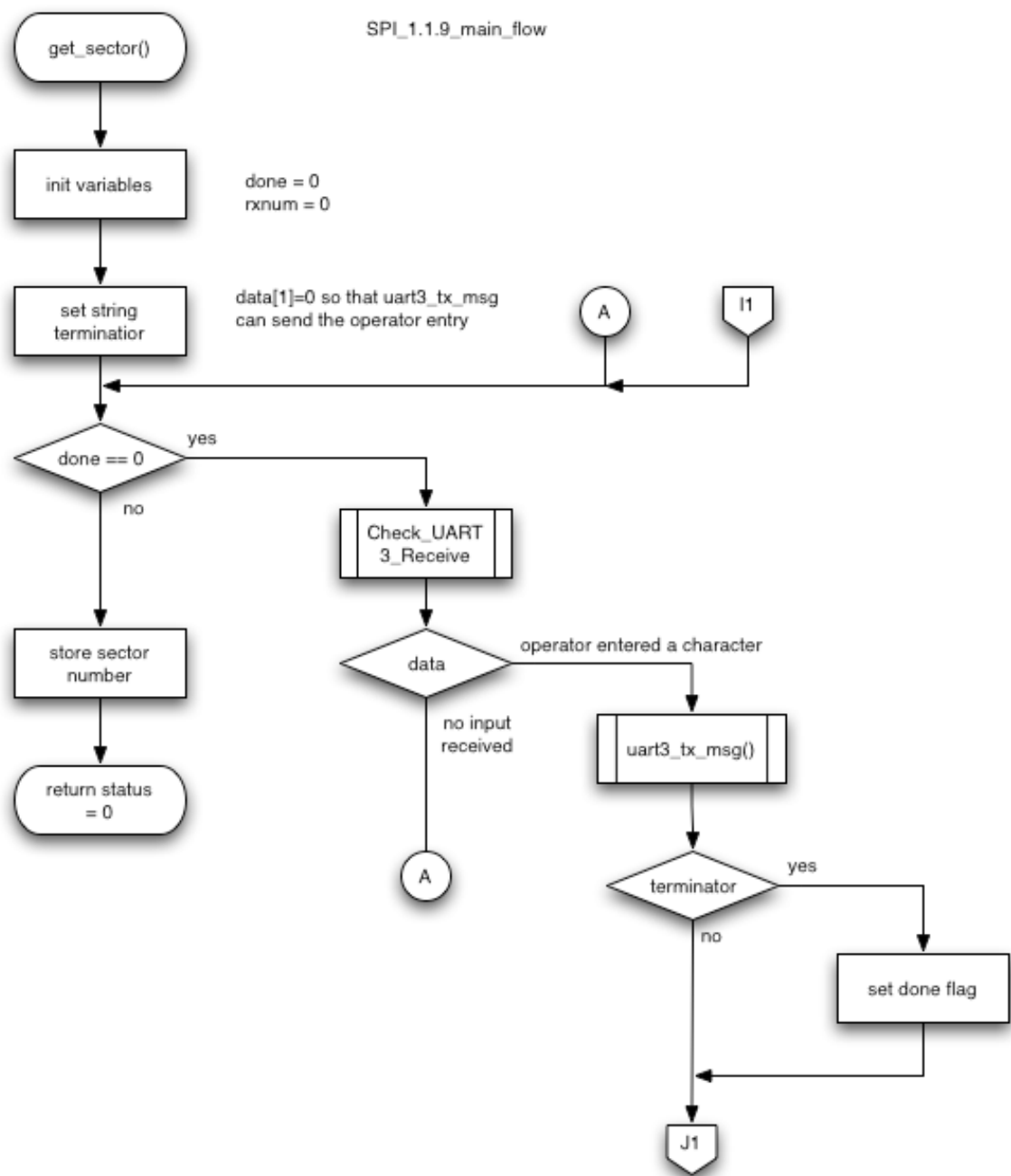


Figure 36. SPI_1.1.10_main_flow

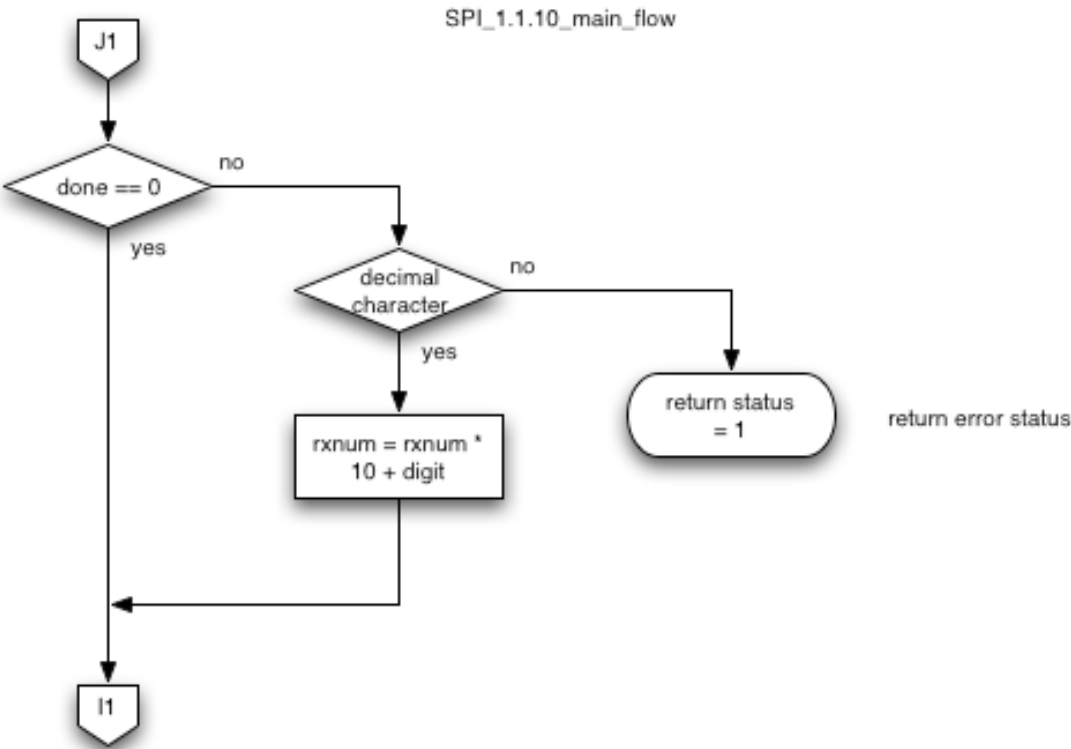


Figure 37. SPI_1.1.11_main_flow

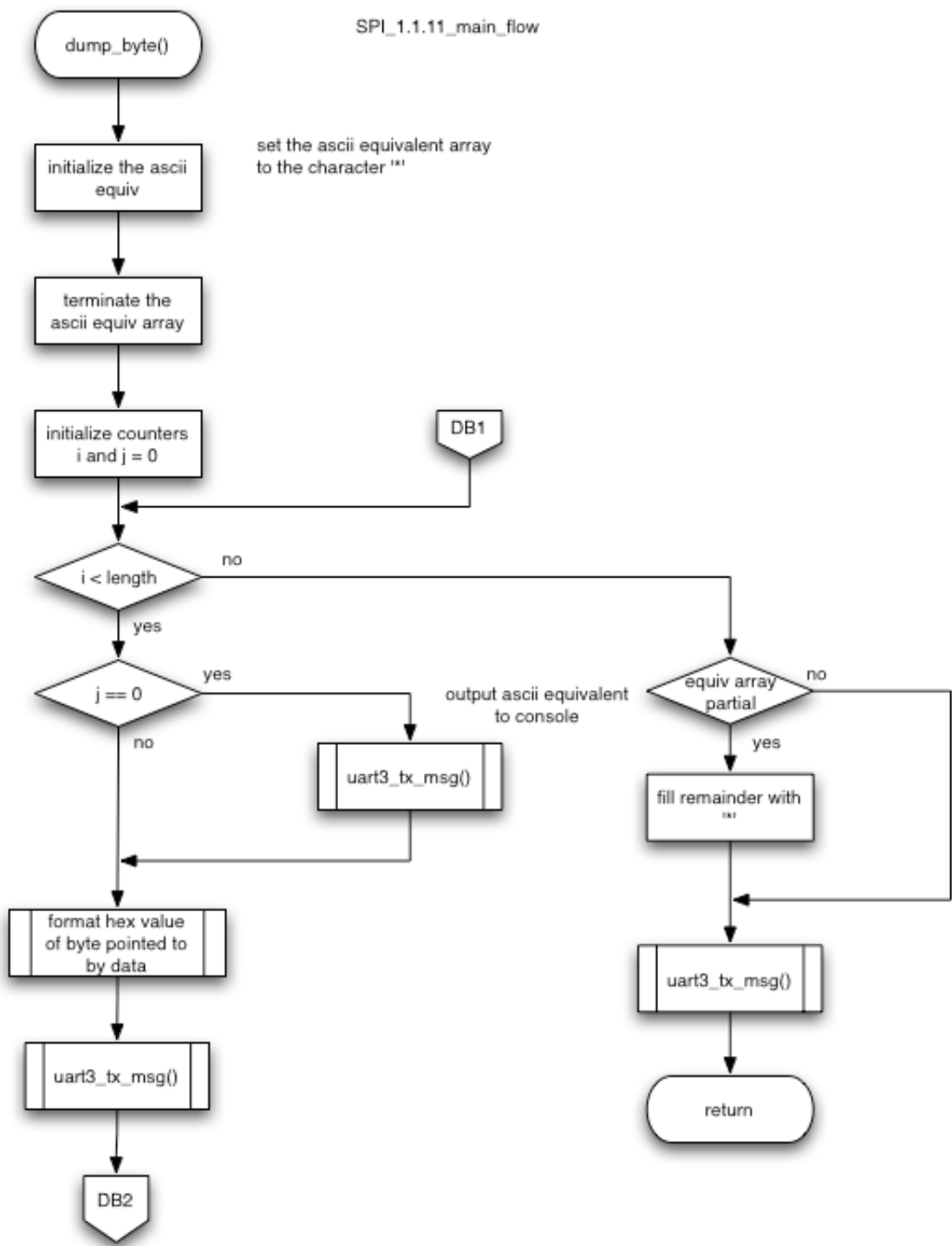
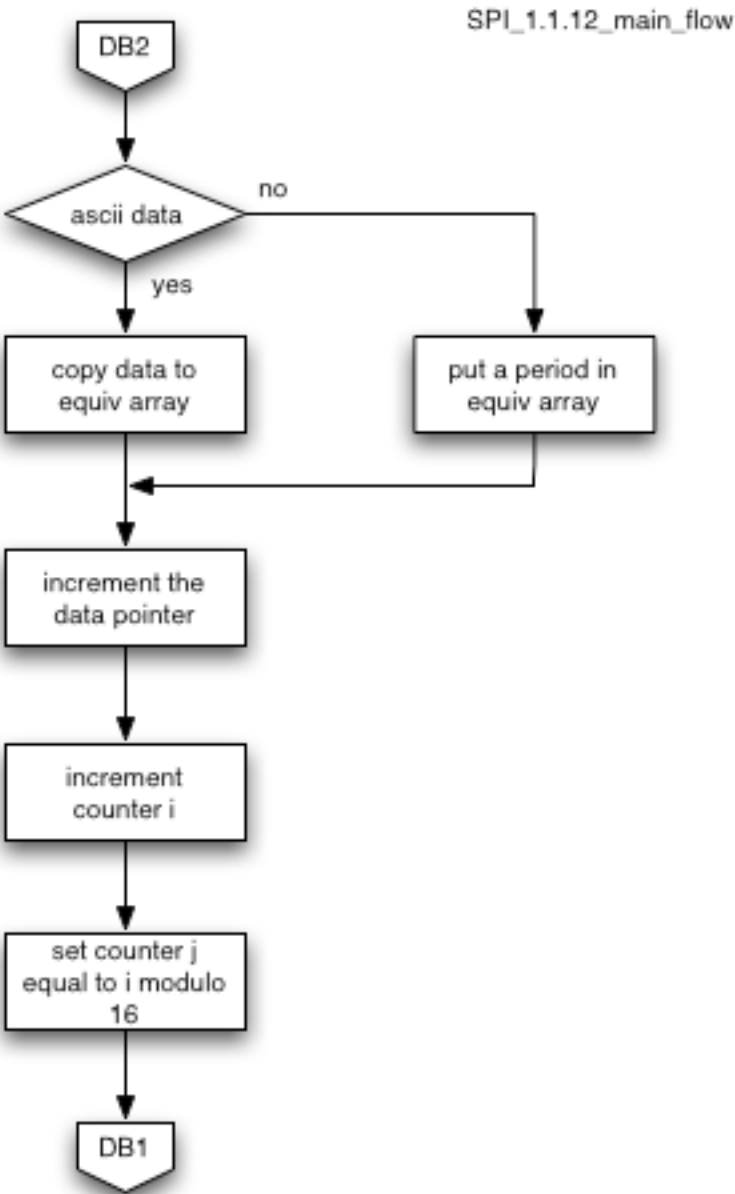


Figure 38. SPI_1.1.12_main_flow



7.3 Sdmemory_flow

Figure 39. SPI_2.0.0_sdmemory_flow

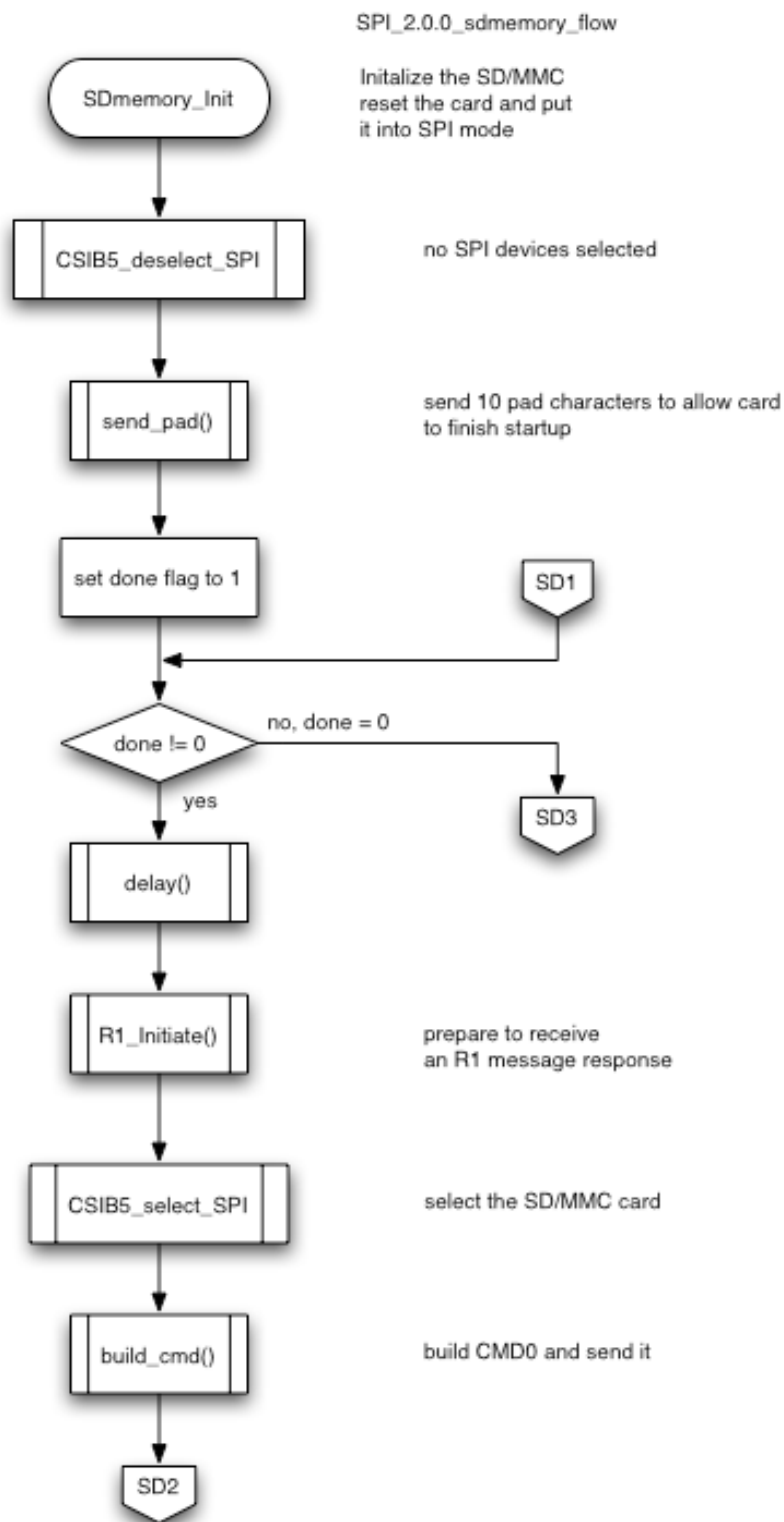


Figure 40. SPI_2.0.1_sdmemory_flow

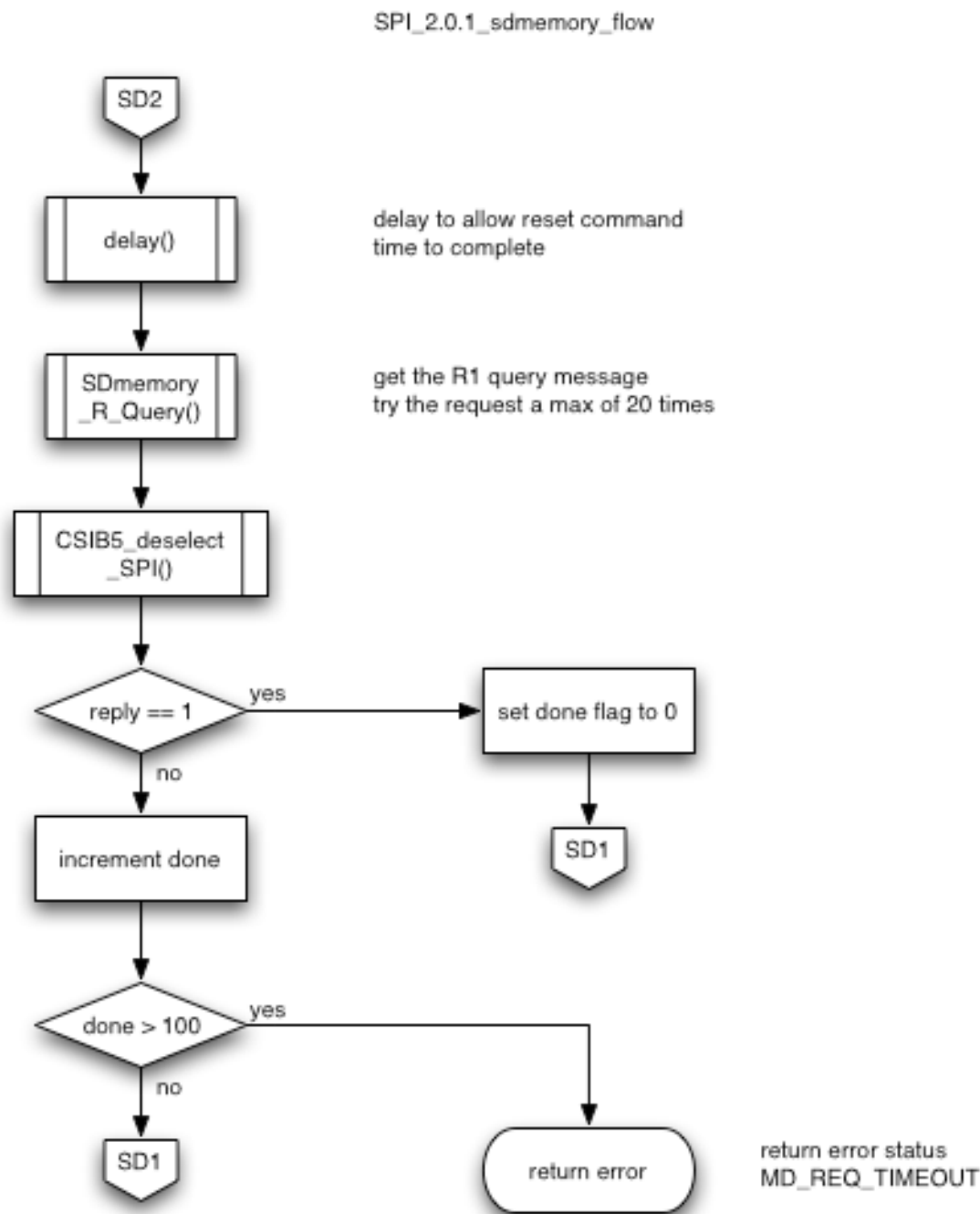


Figure 41. SPI_2.0.2_sdmemory_flow

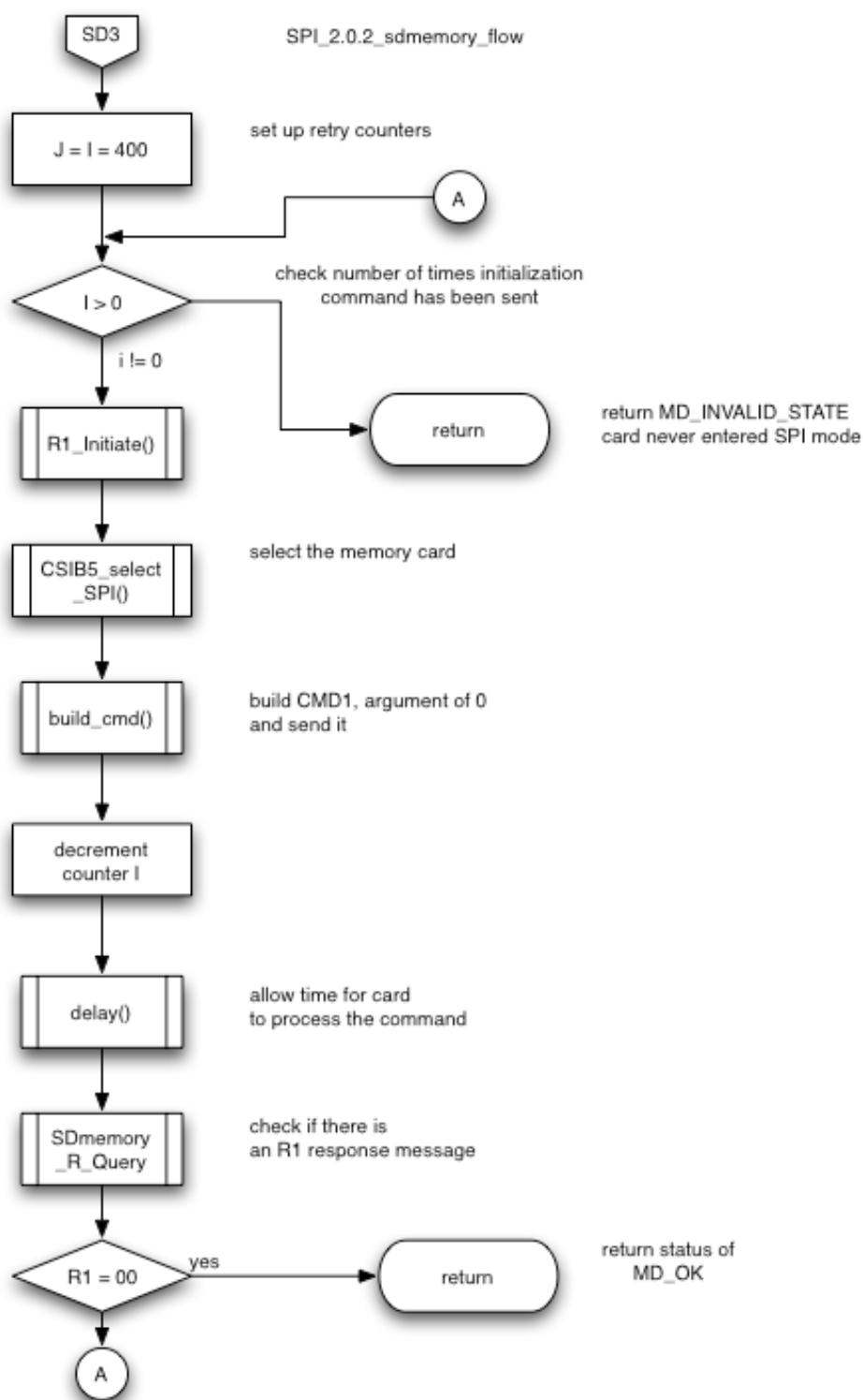


Figure 42. SPI_2.0.3_sdmemory_flow

SPI_2.0.3_sdmemory_flow

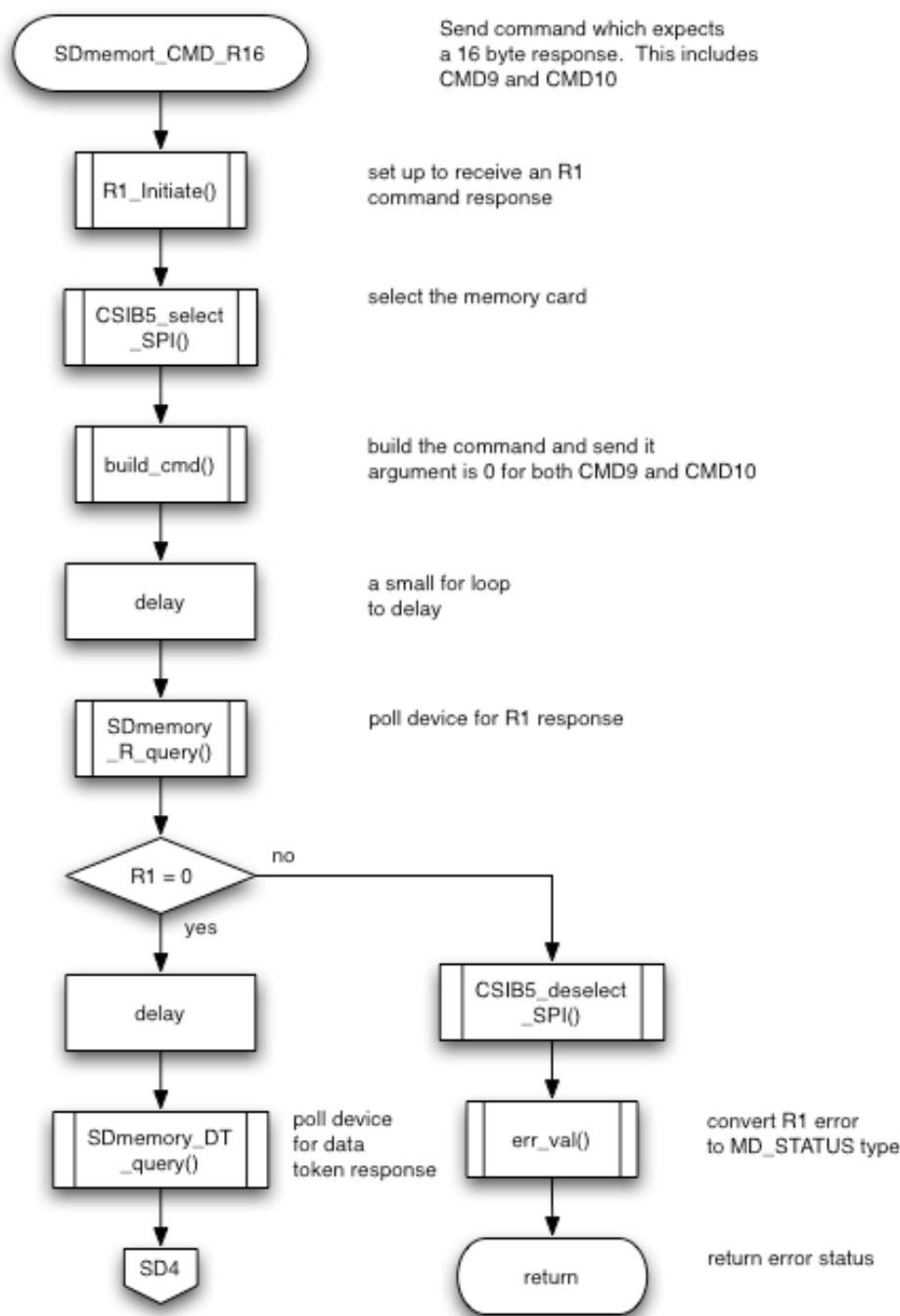


Figure 43. SPI_2.0.4_sdmemory_flow

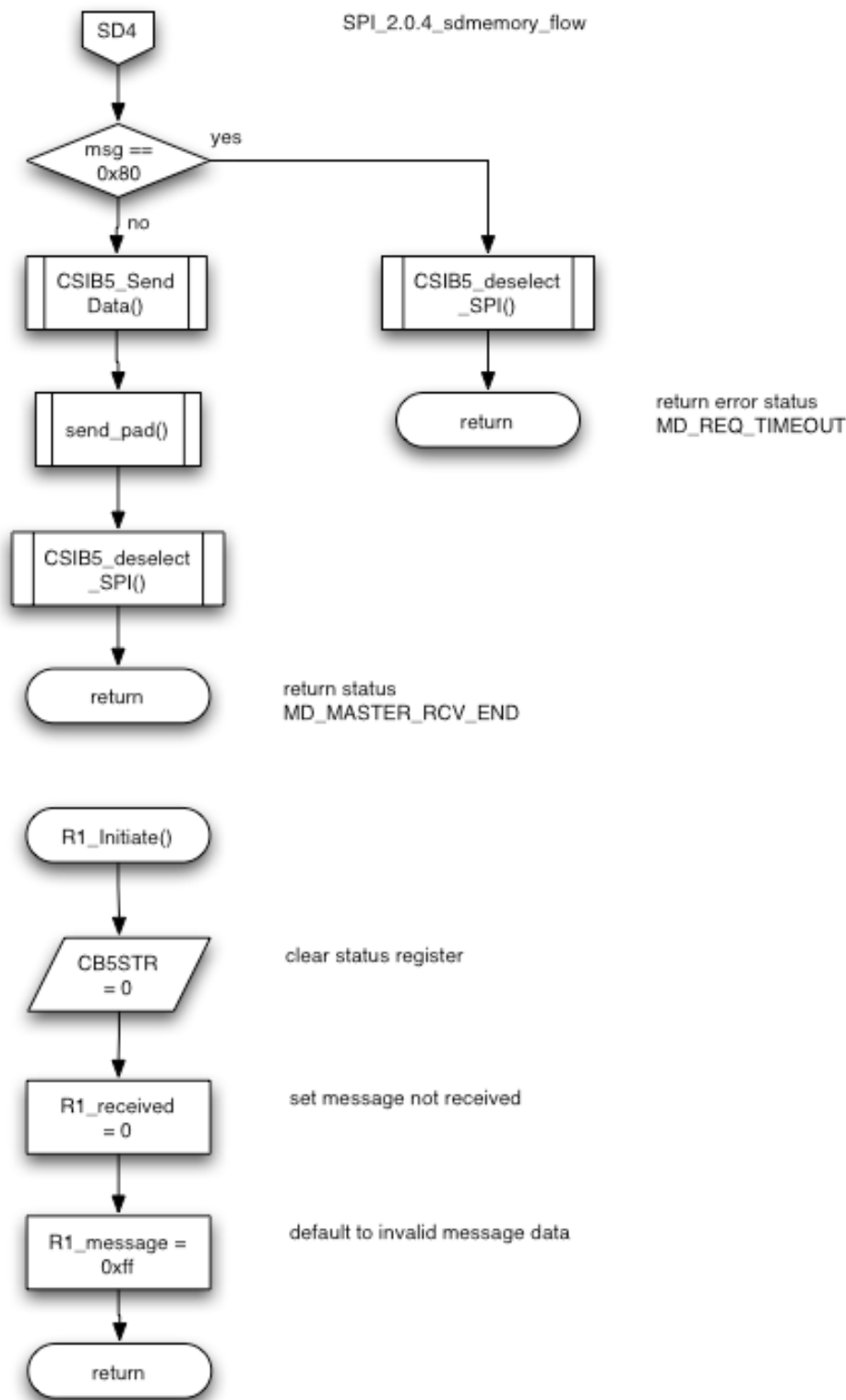


Figure 44. SPI_2.0.5_sdmemory_flow

SPI_2.0.5_sdmemory_flow

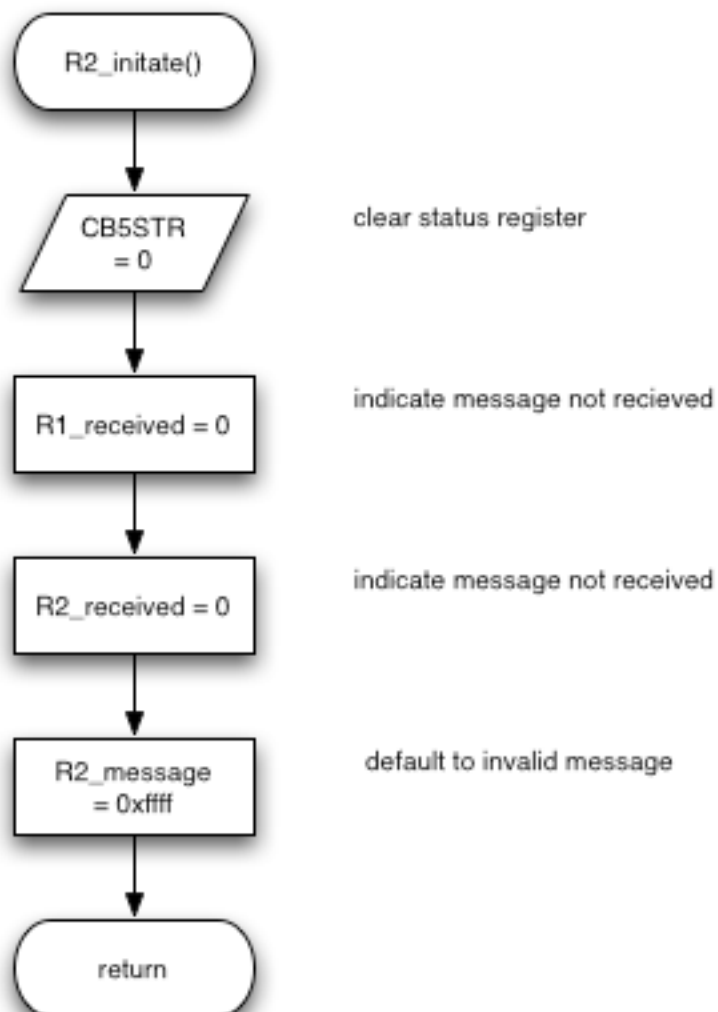


Figure 45. SPI_2.0.6_sdmemory_flow

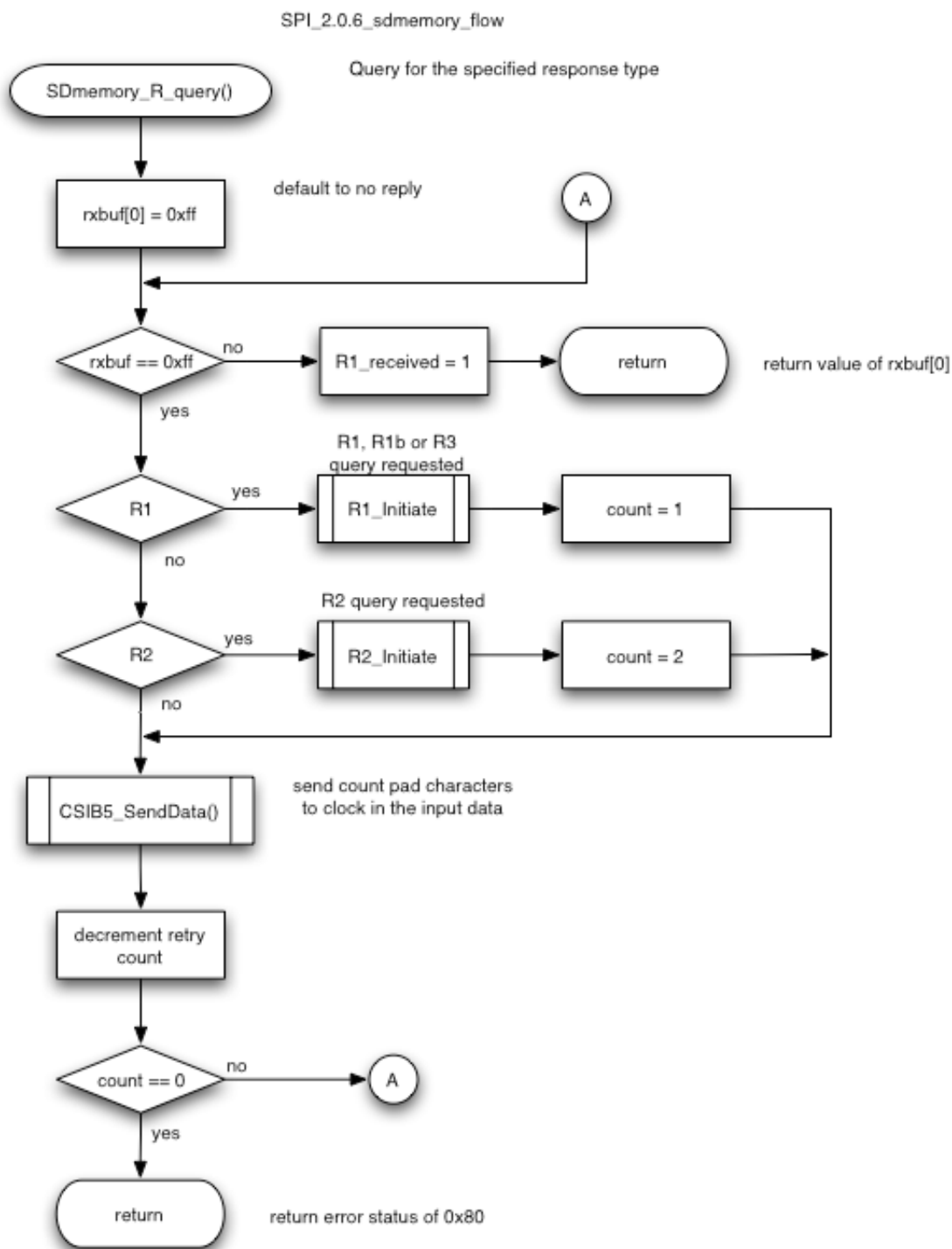


Figure 46. SPI_2.0.7_sdmemory_flow

SPI_2.0.7_sdmemory_flow

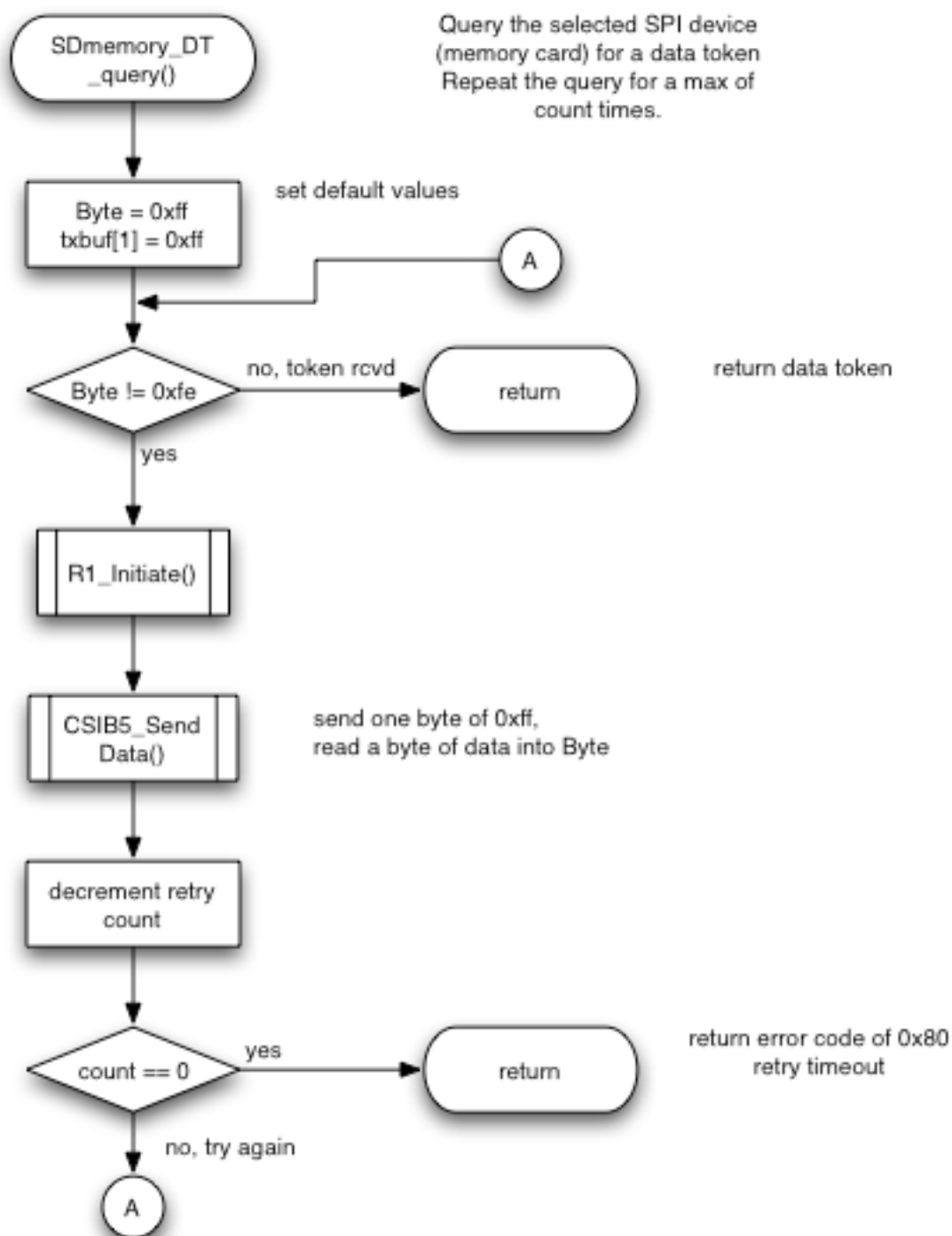


Figure 47. SPI_2.0.8_sdmemory_flow

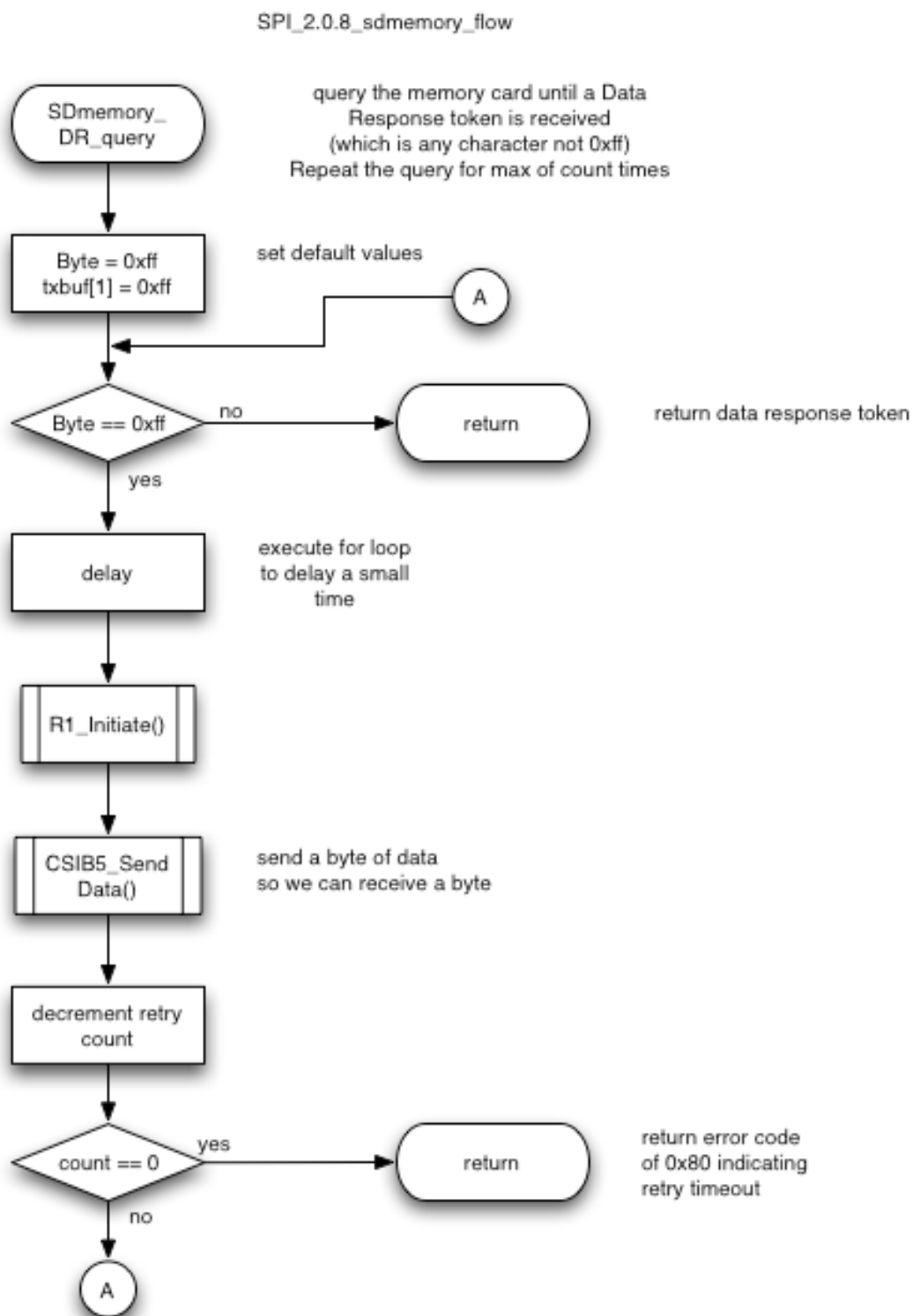


Figure 48. SPI_2.0.9_sdmemory_flow

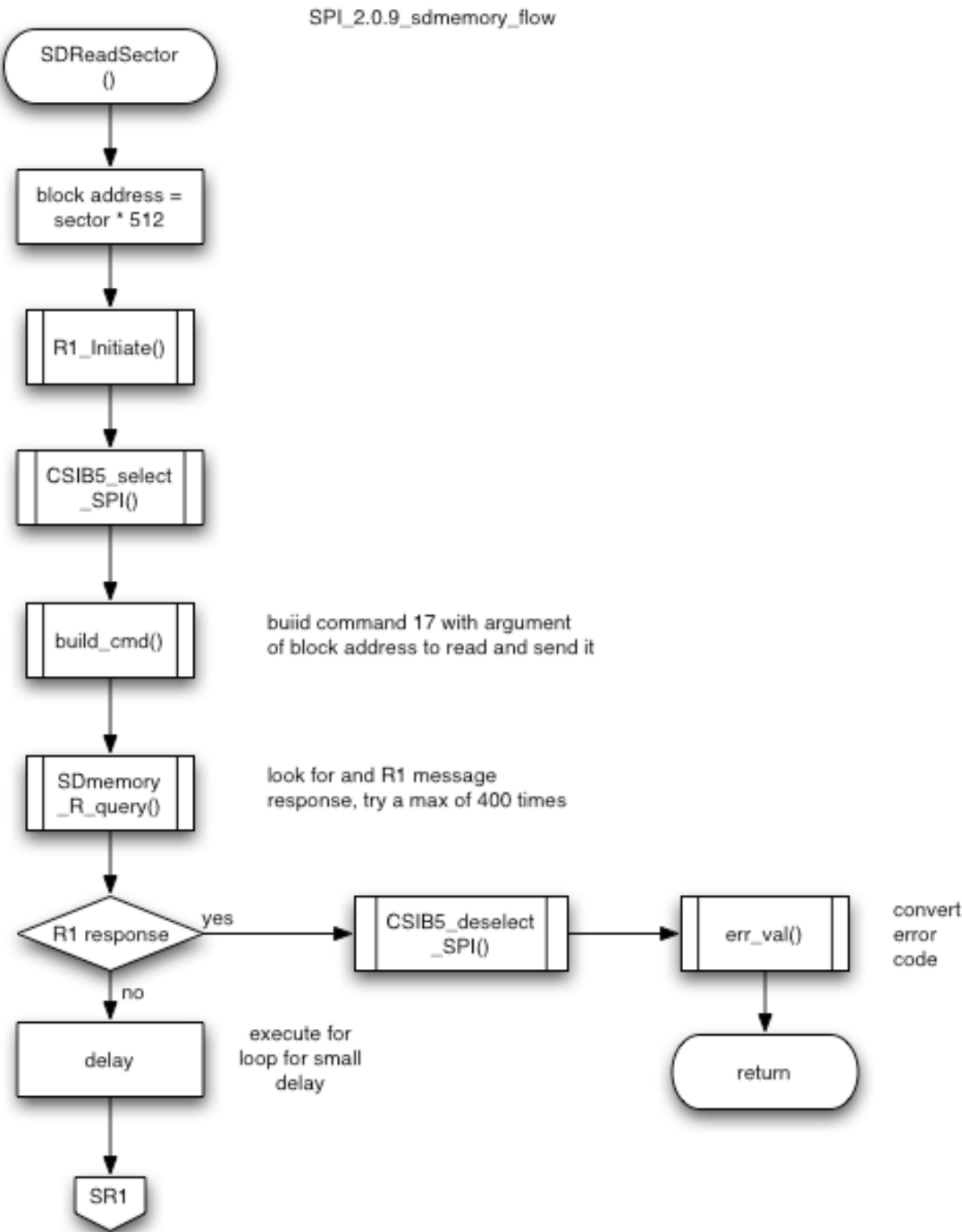


Figure 49. SPI_2.0.10_sdmemory_flow

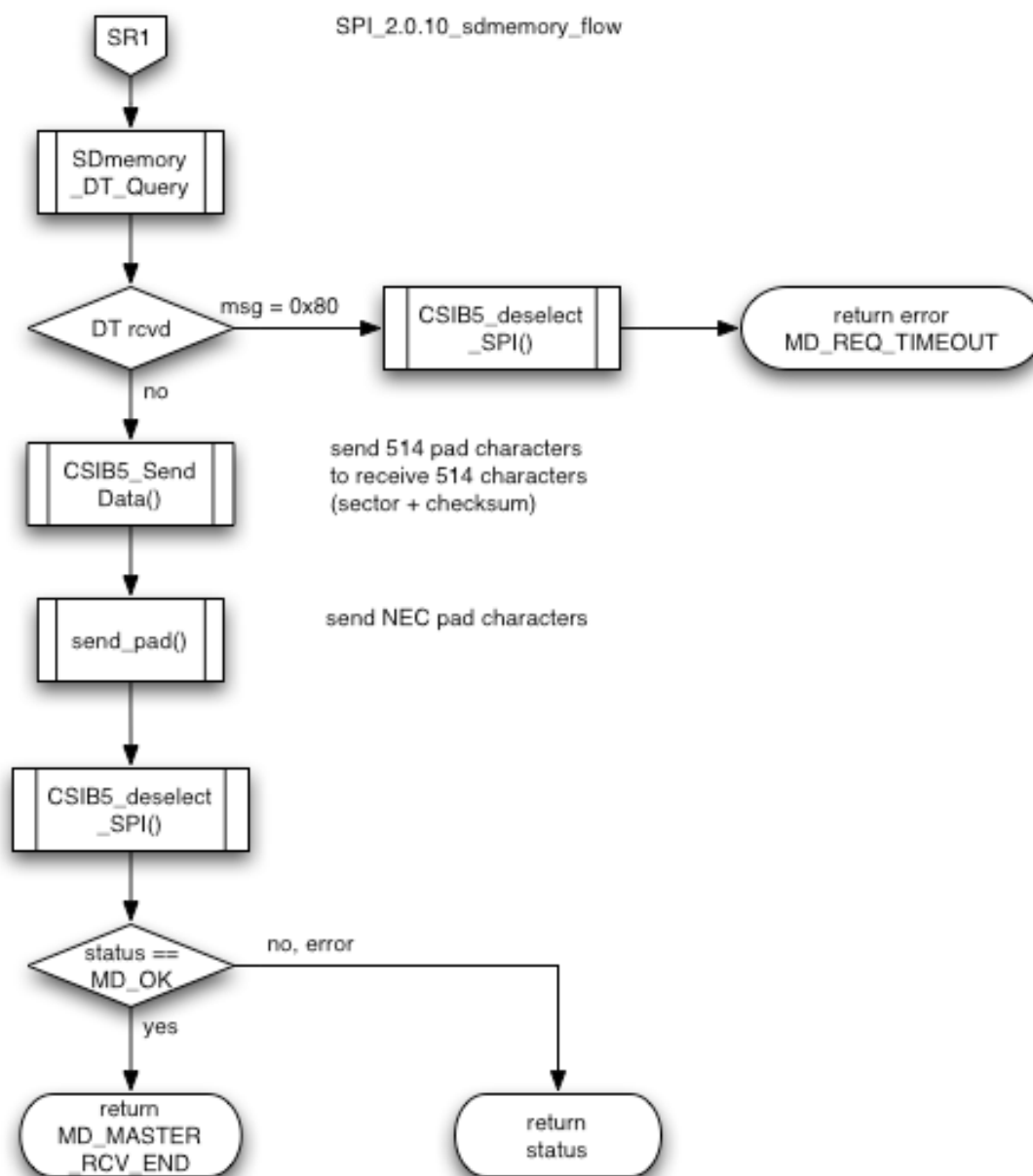


Figure 50. SPI_2.0.11_sdmemory_flow

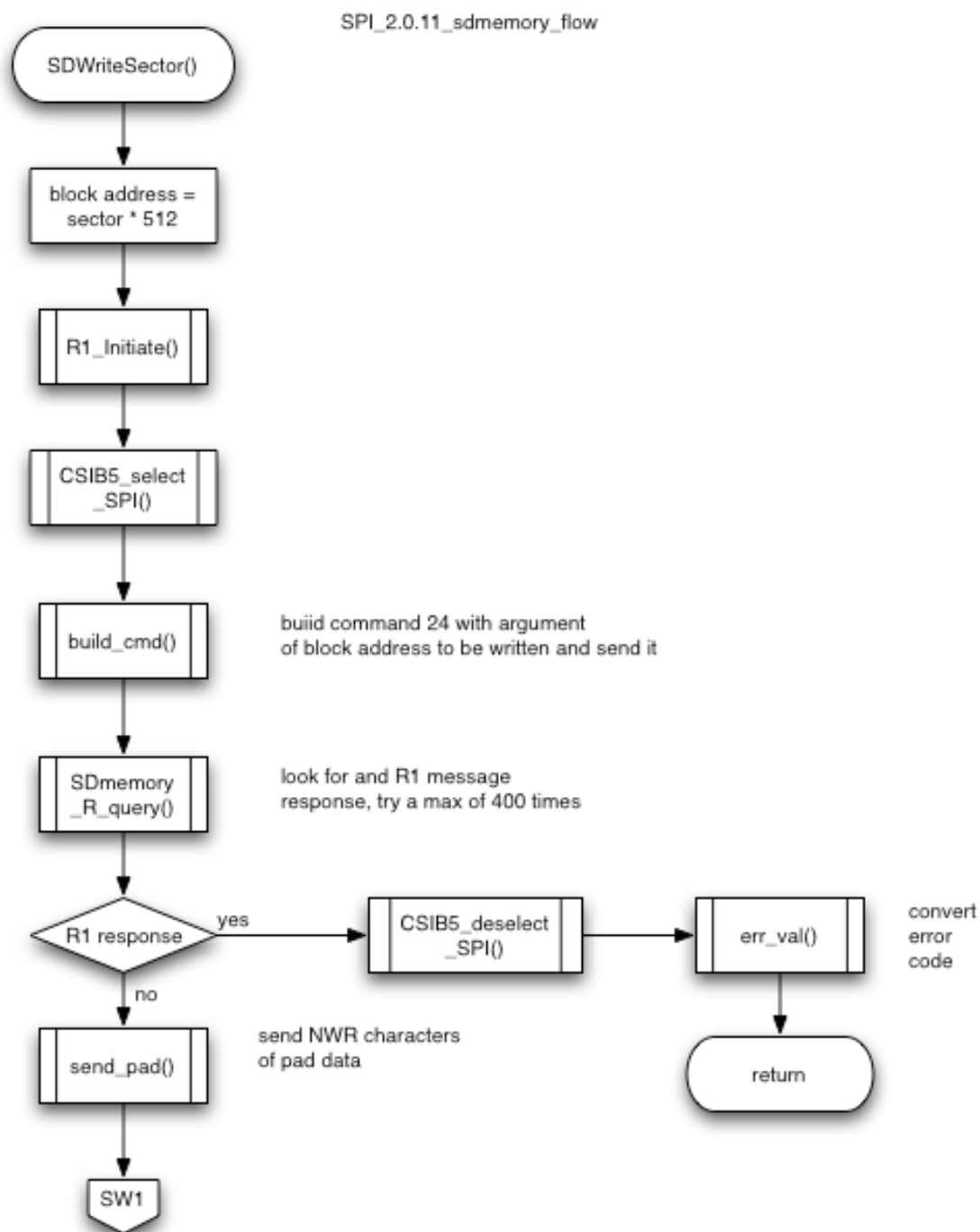


Figure 51. SPI_2.0.12_sdmemory_flow

SPI_2.0.12_sdmemory_flow

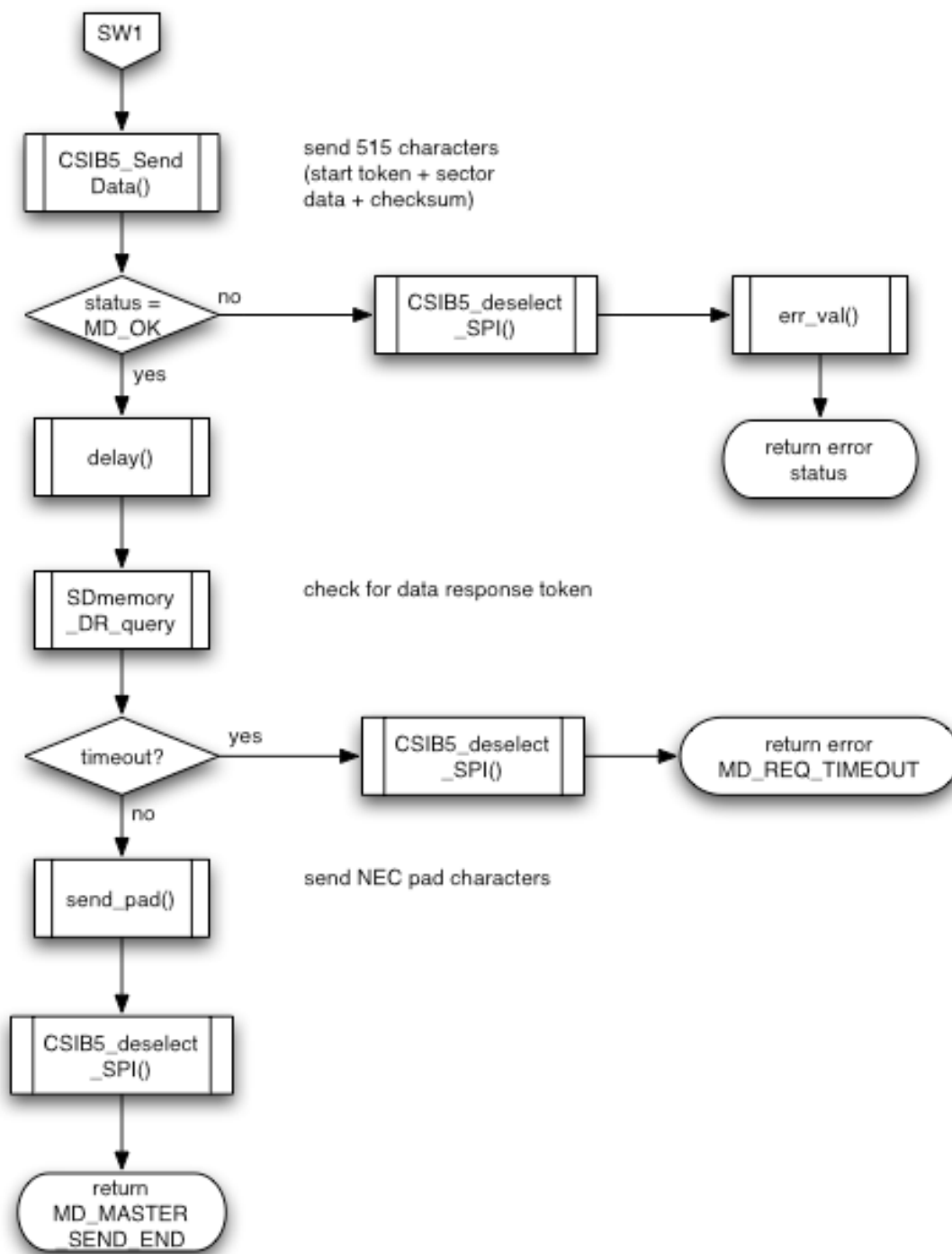


Figure 52. SPI_2.0.13_sdmemory_flow

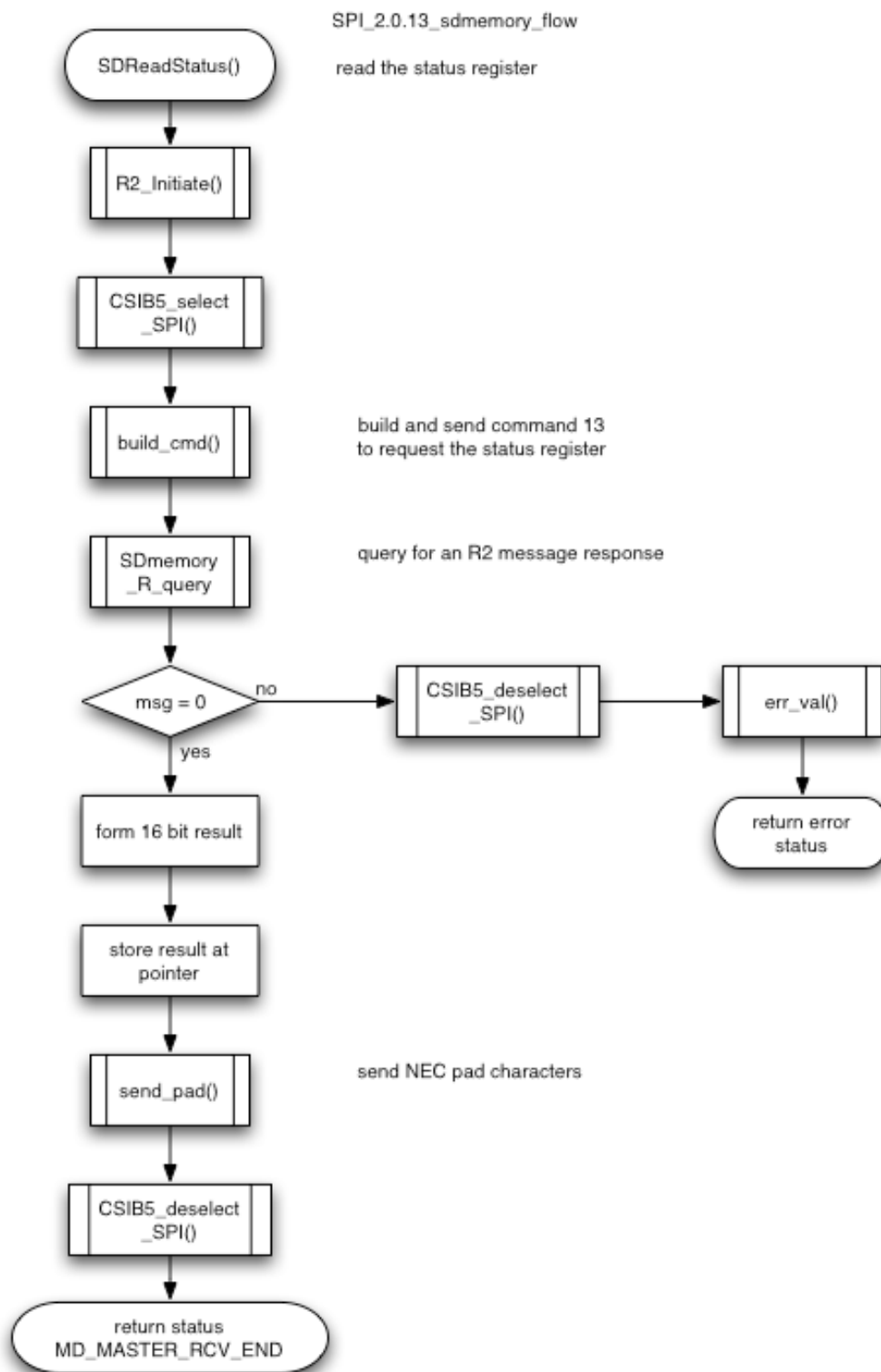


Figure 53. SPI_2.0.14_sdmemory_flow

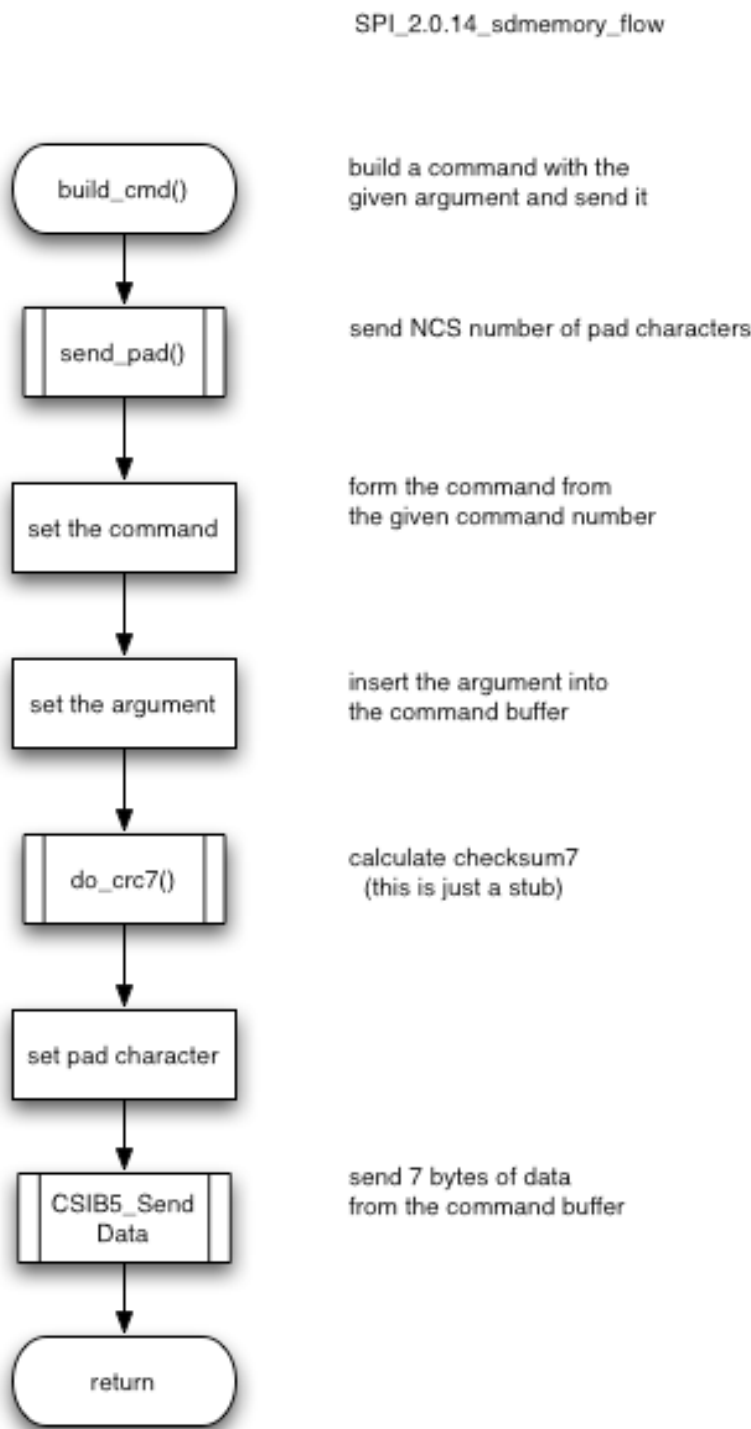


Figure 54. SPI_2.0.15_sdmemory_flow

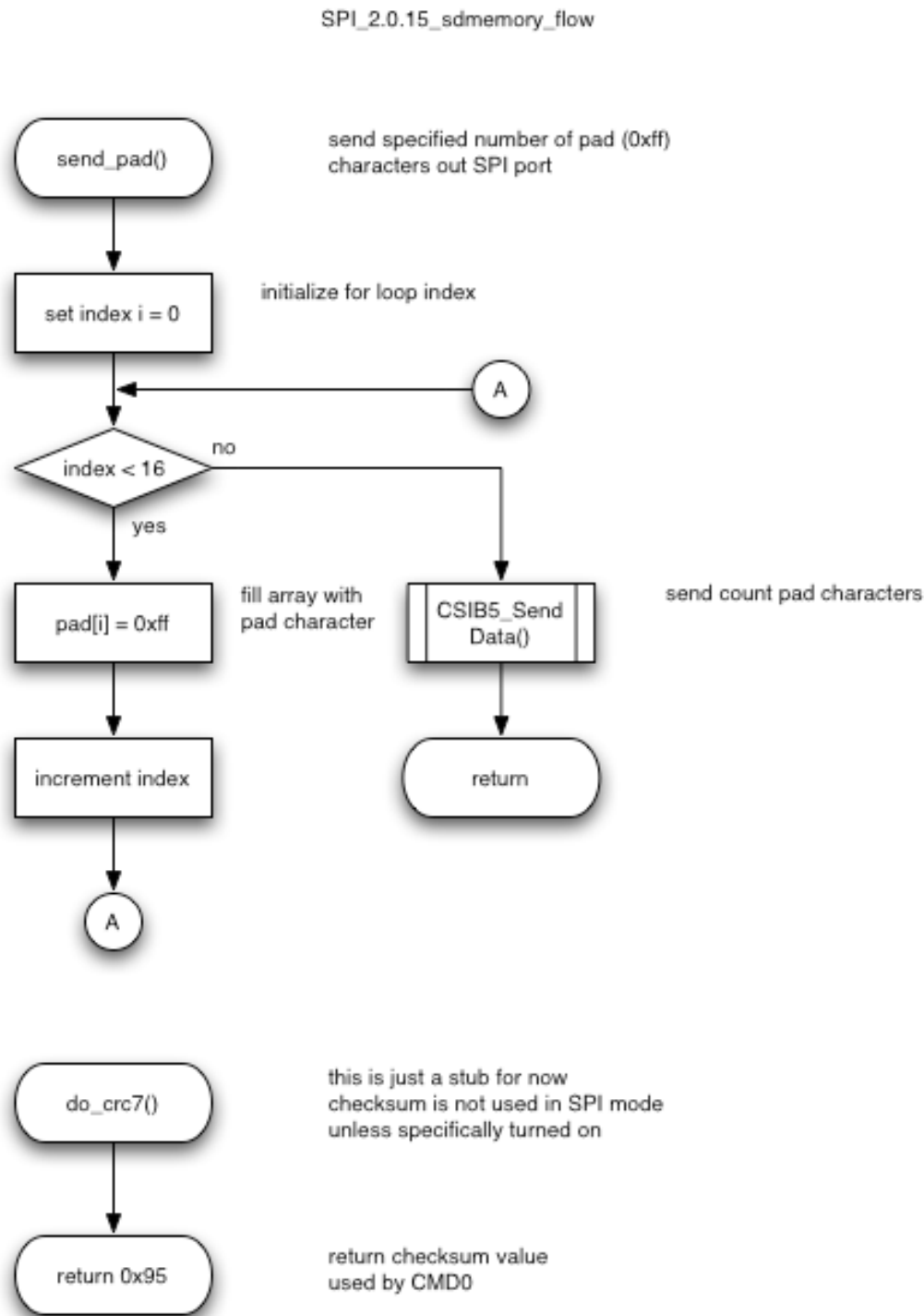


Figure 55. SPI_2.0.16_sdmemory_flow

SPI_2.0.16_sdmemory_flow

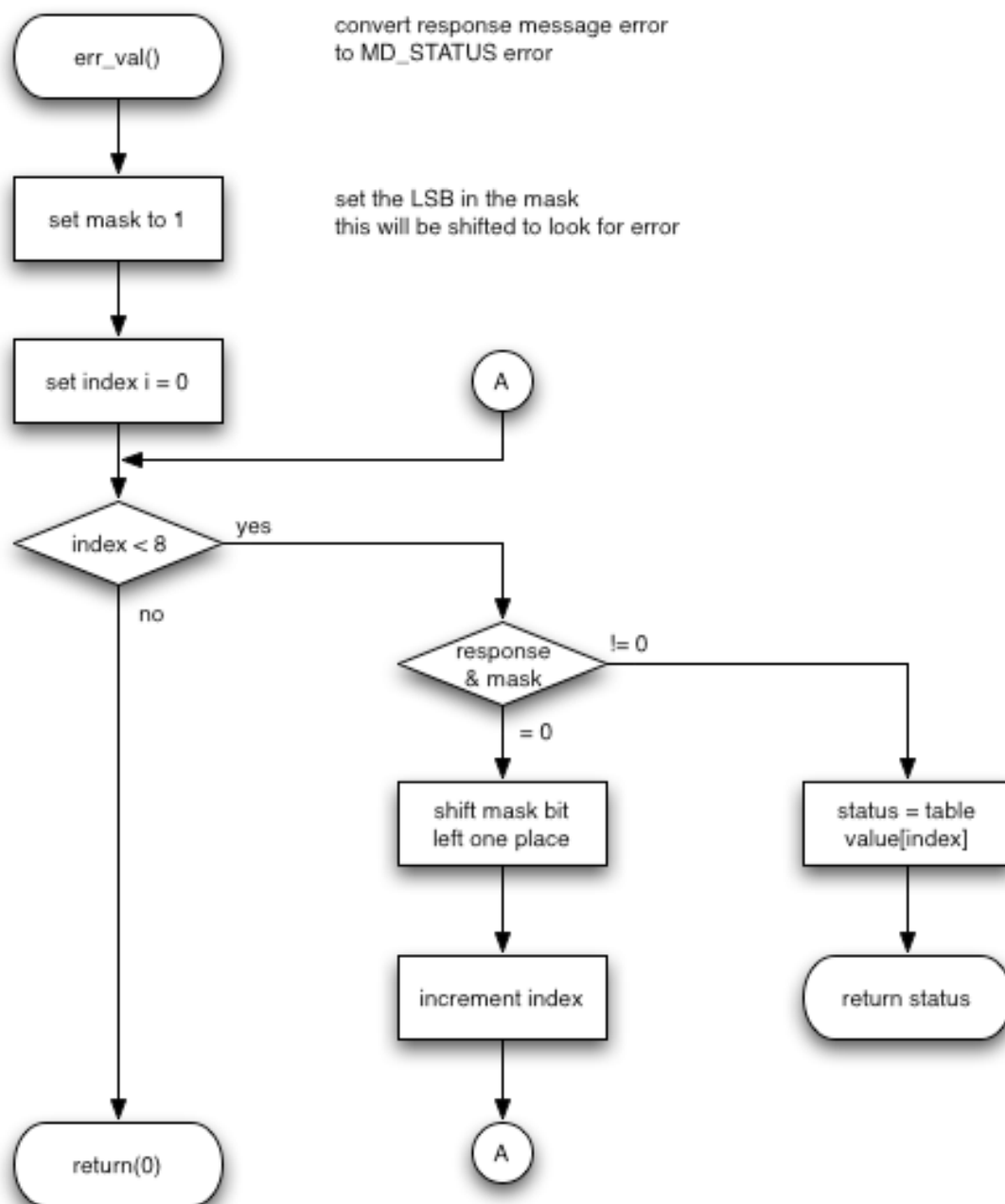
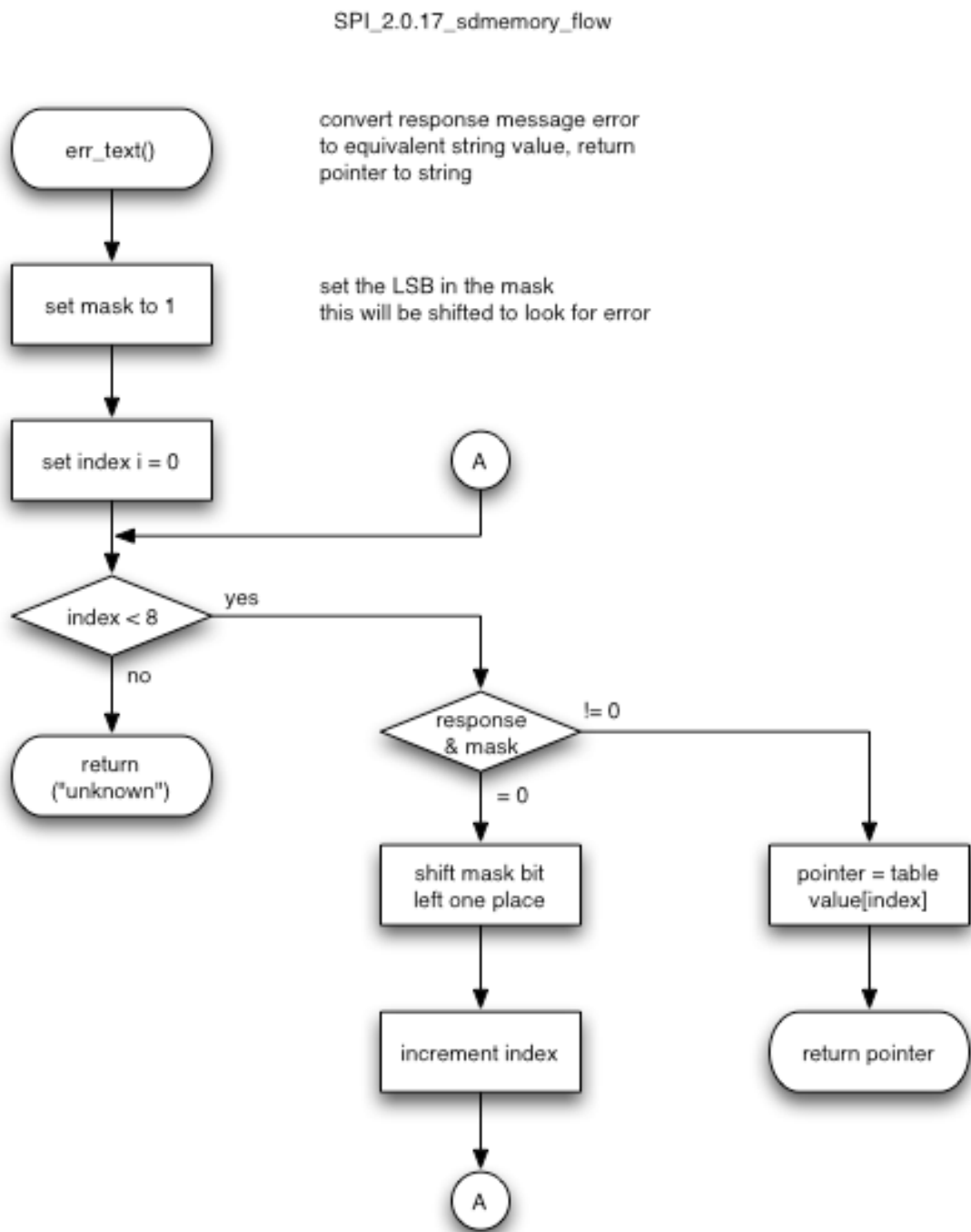


Figure 56. SPI_2.0.17_sdmemory_flow



7.4 Serial_interface

Figure 57. SPI_3.0.0_serial_interface

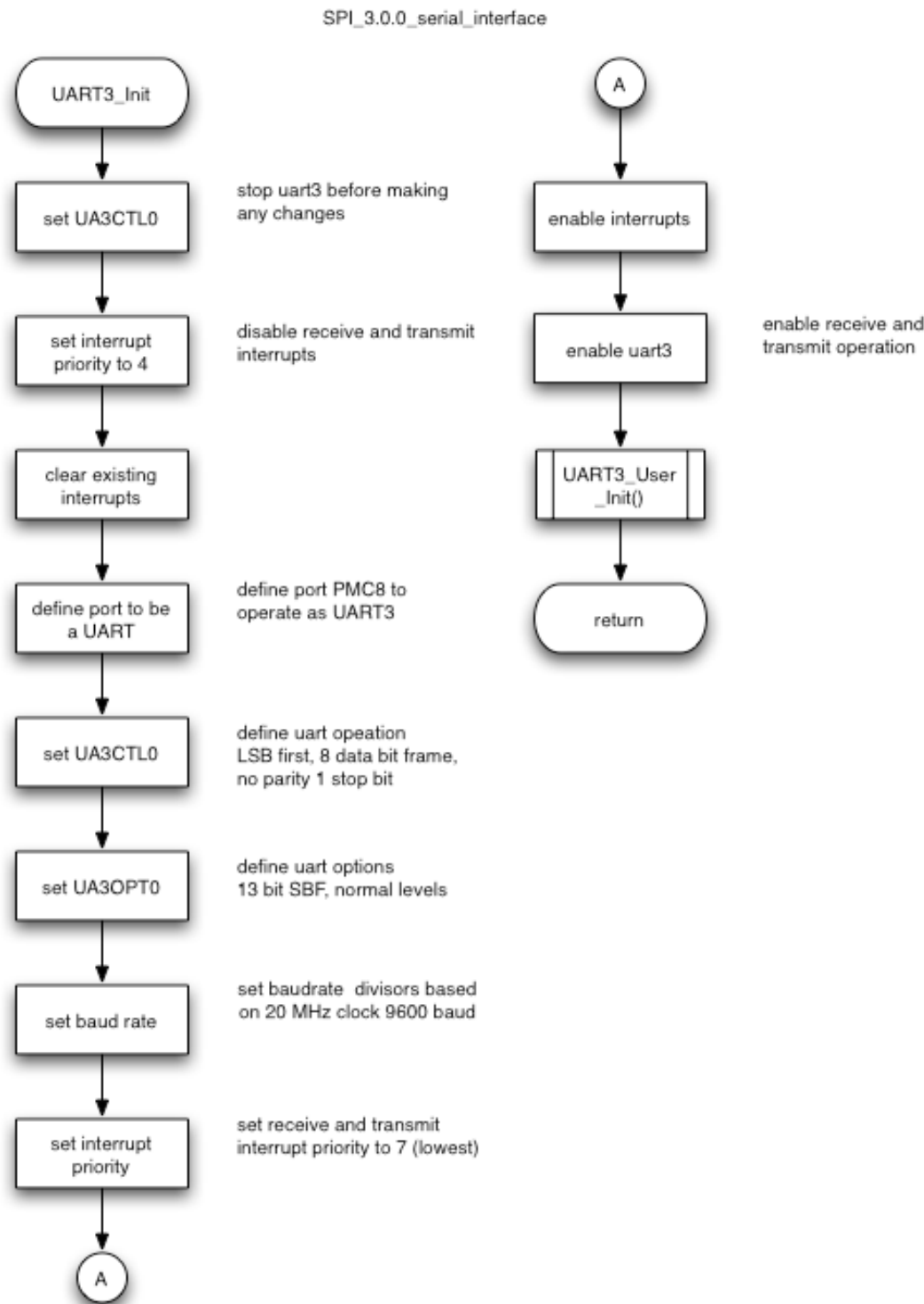


Figure 58. SPI_3.0.1_serial_interface

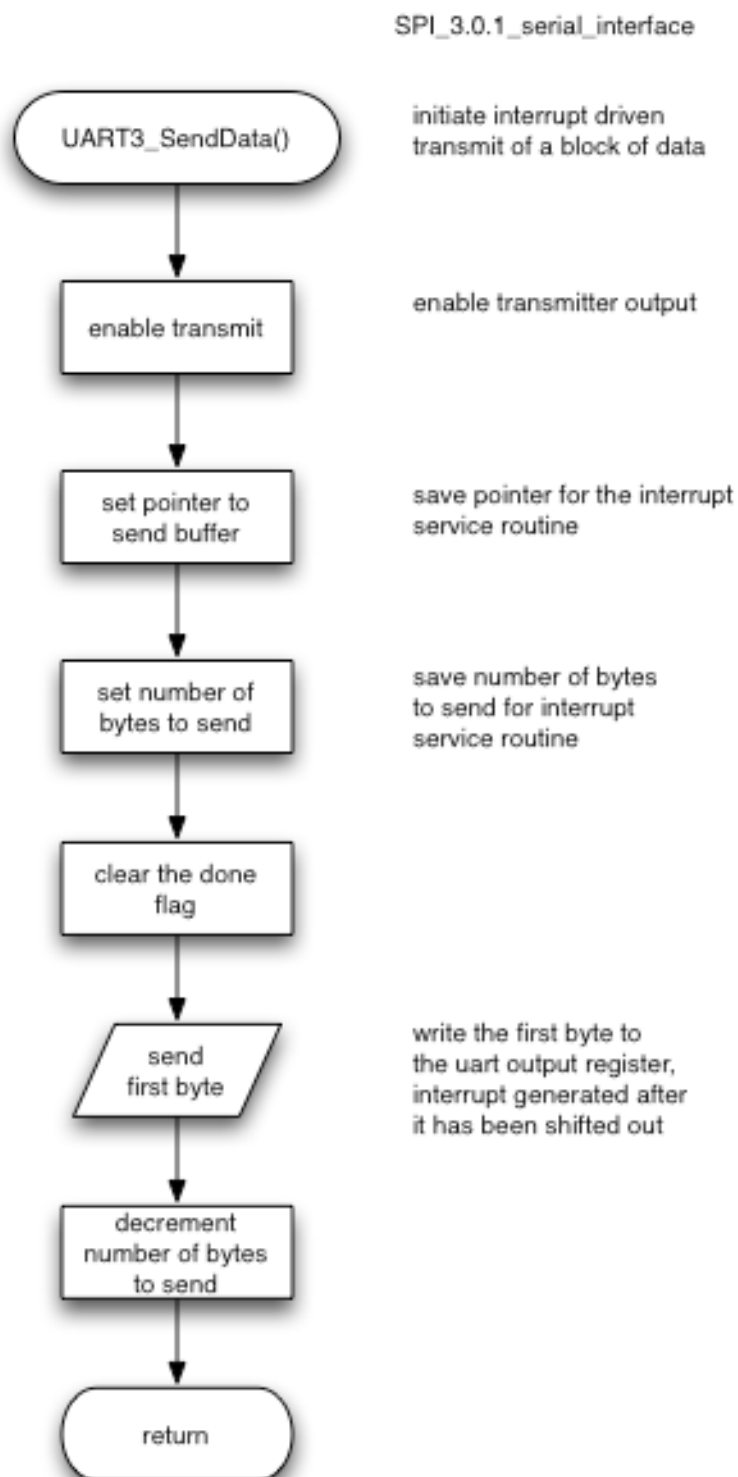


Figure 59. SPI_3.0.2_serial_interface

SPI_3.0.2_serial_interface

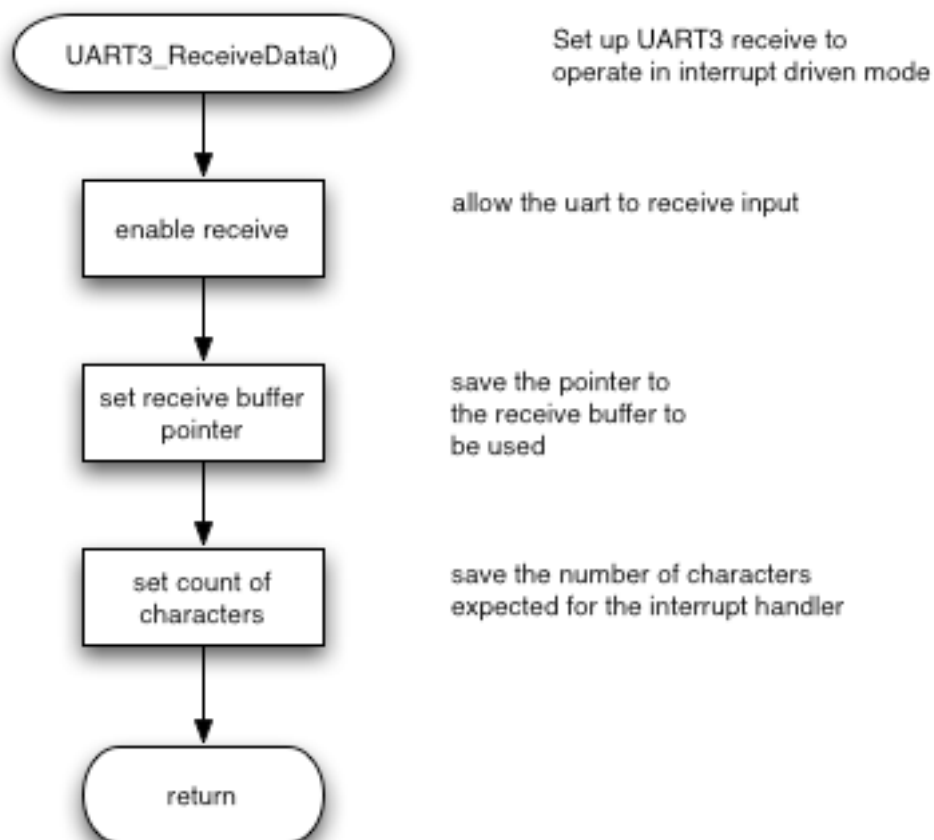


Figure 60. SPI_3.0.3_serial_interface

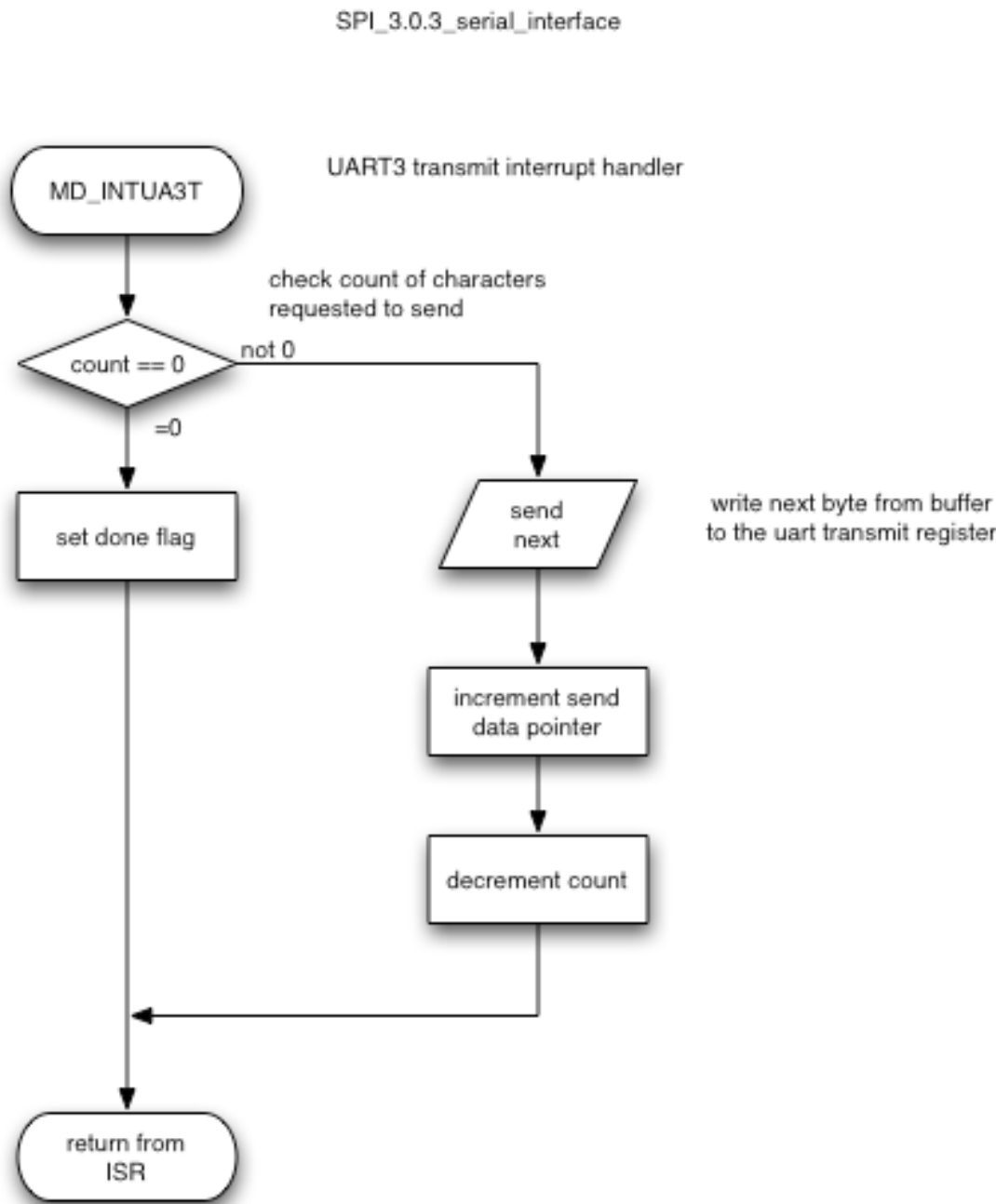


Figure 61. SPI_3.0.4_serial_interface

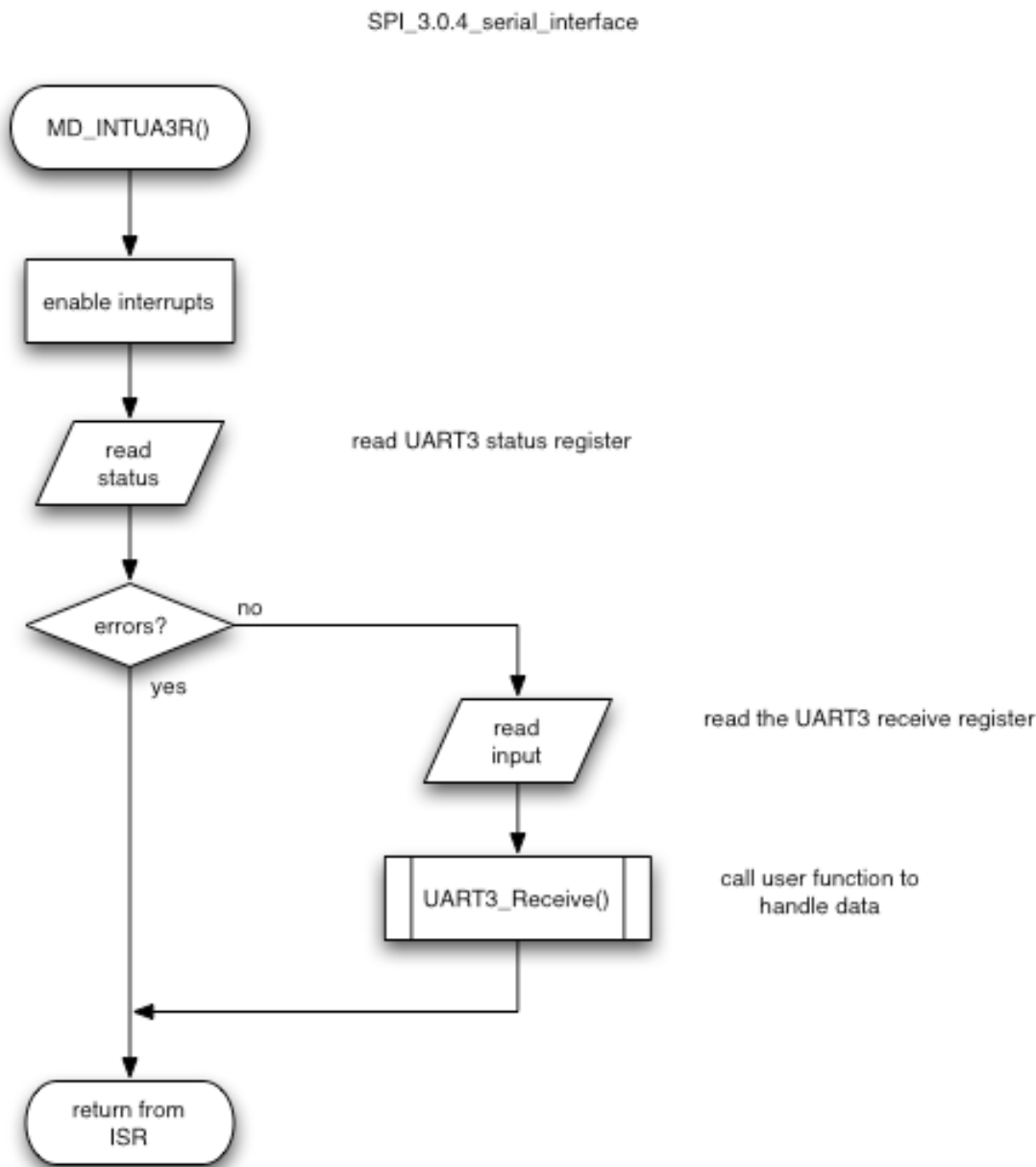


Figure 62. SPI_3.0.5_serial_interface

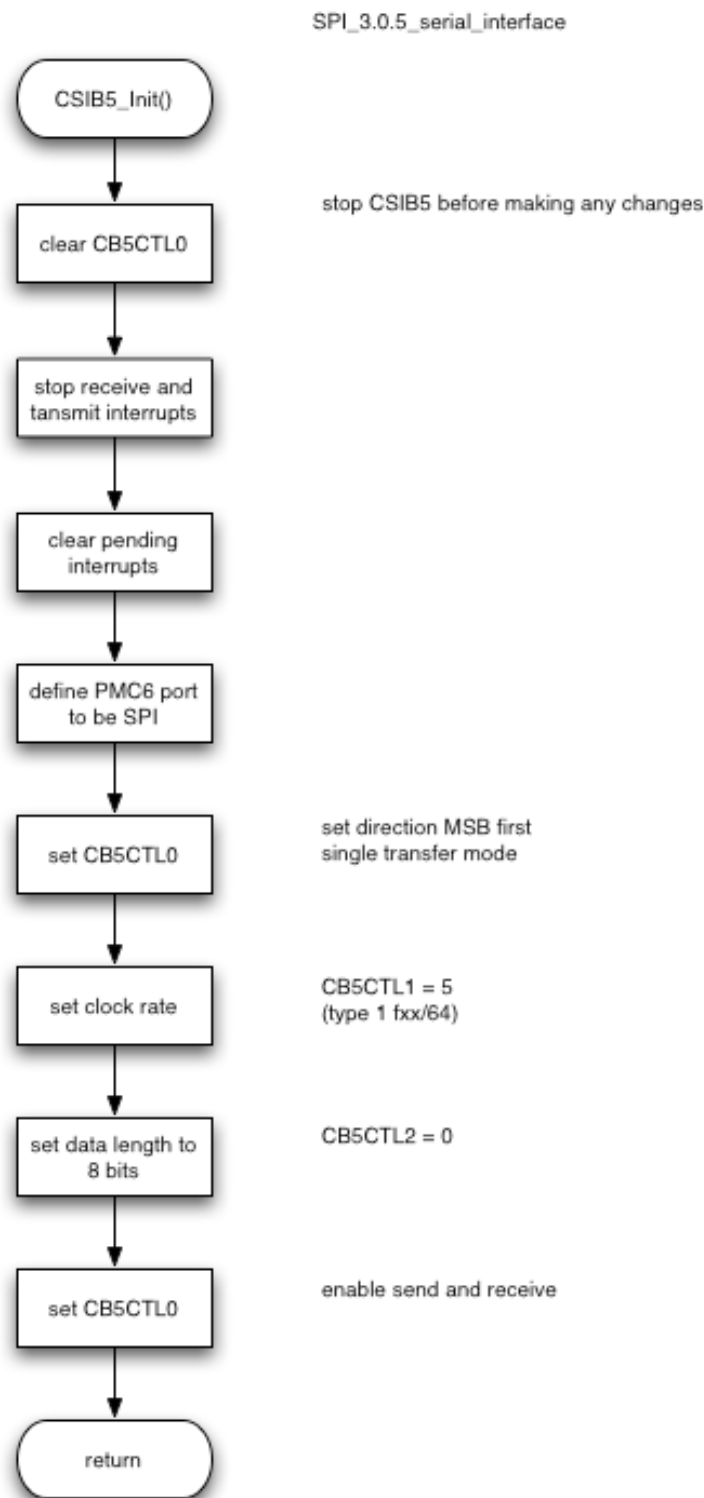


Figure 63. SPI_3.0.6_serial_interface

SPI_3.0.6_serial_interface

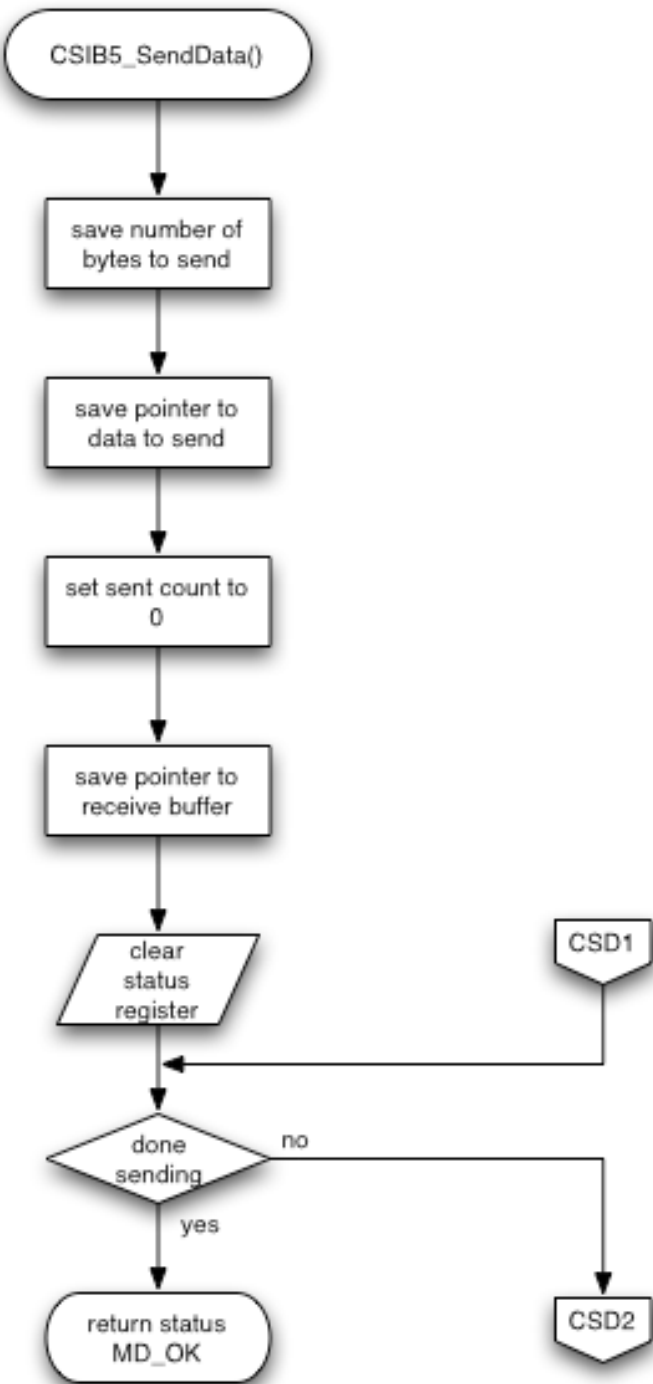
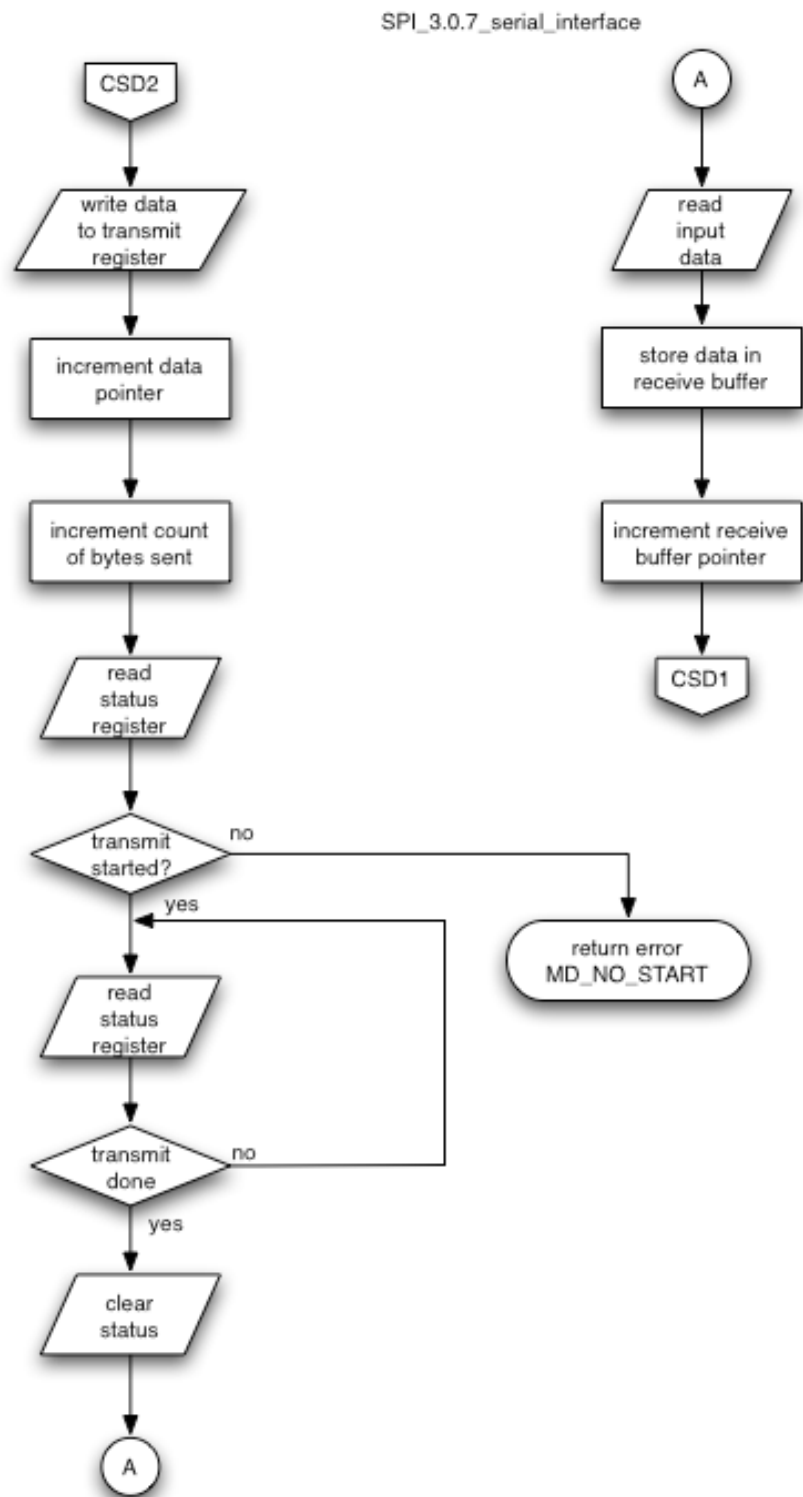


Figure 64. SPI_3.0.7_serial_interface



7.5 Serial_interface_user

Figure 65. SPI_3.1.0_serial_interface_user

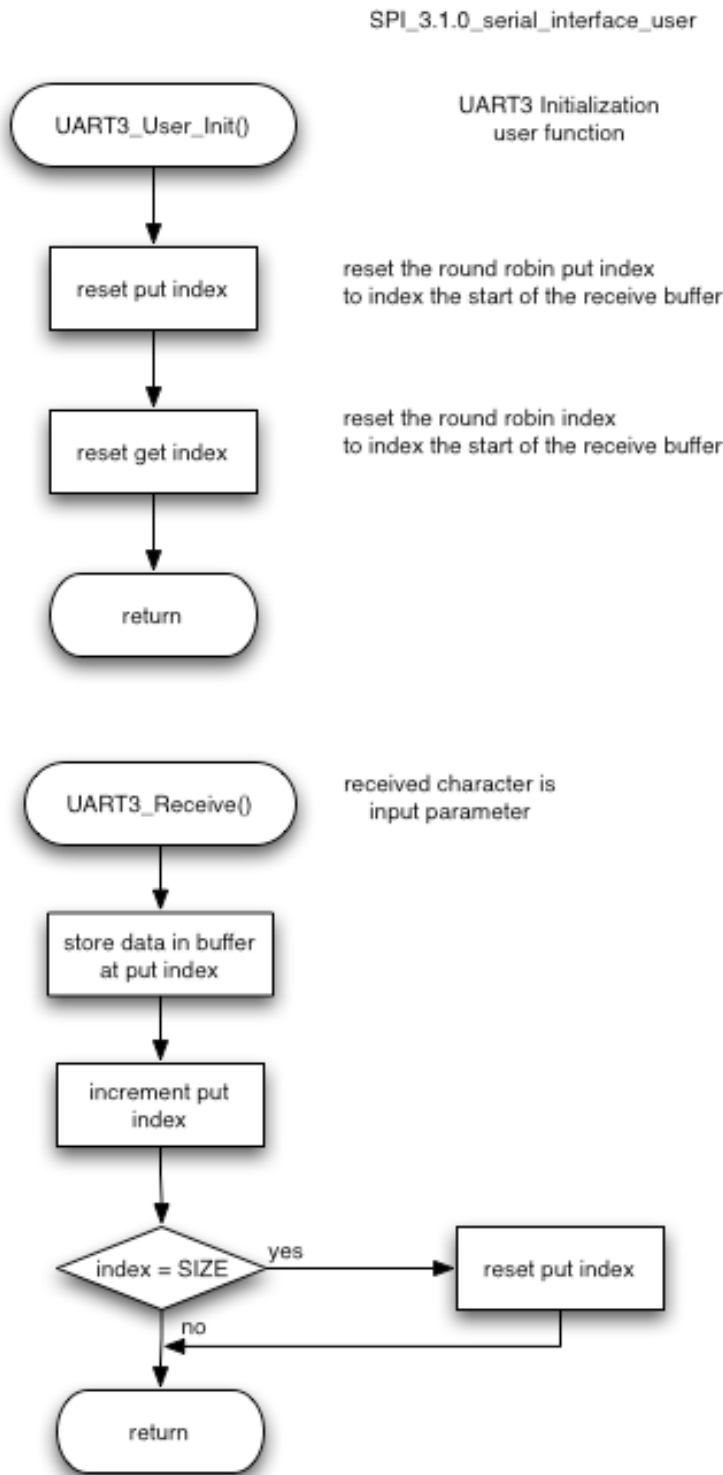


Figure 66. SPI_3.1.1_serial_interface_user

SPI_3.1.1_serial_interface_user

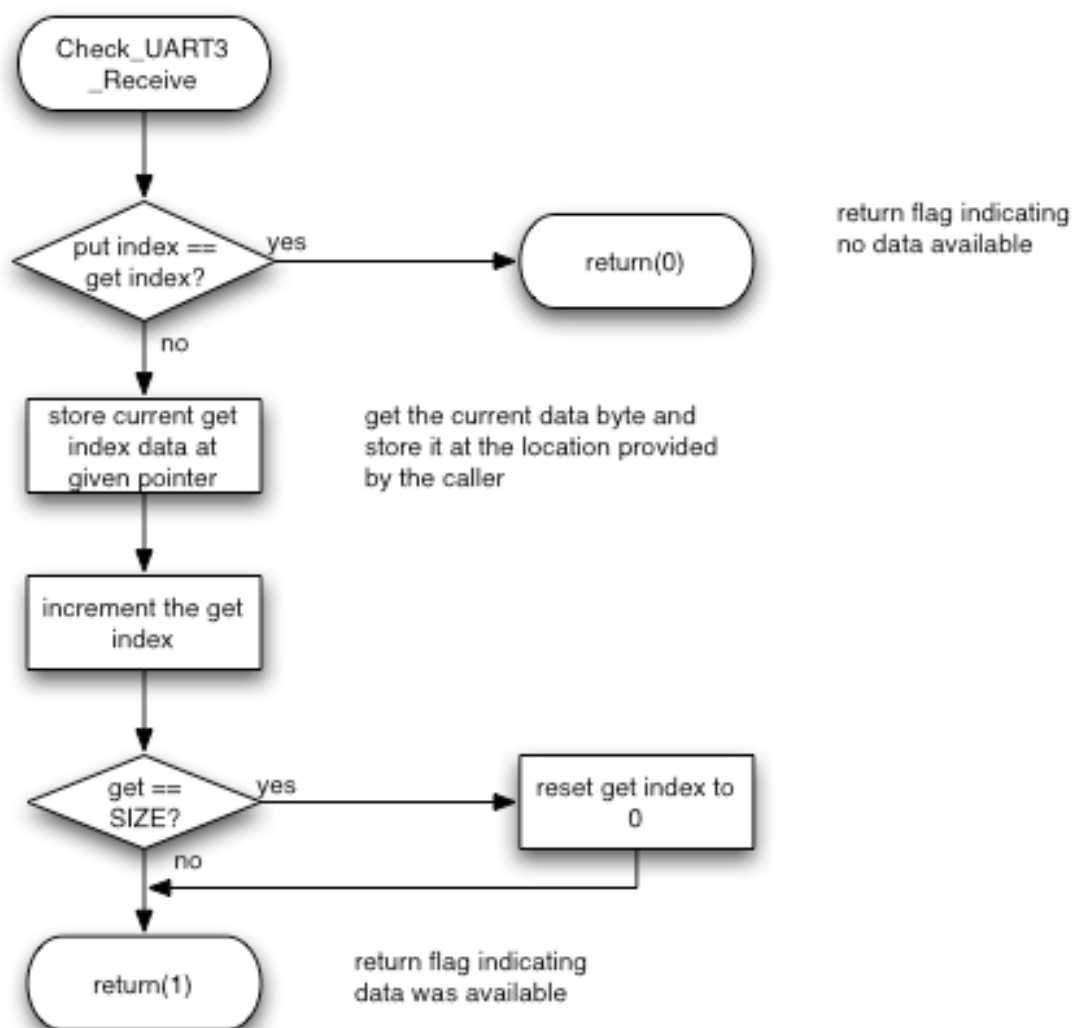


Figure 67. SPI_3.1.2_serial_interface_user

SPI_3.1.2_serial_interface_user

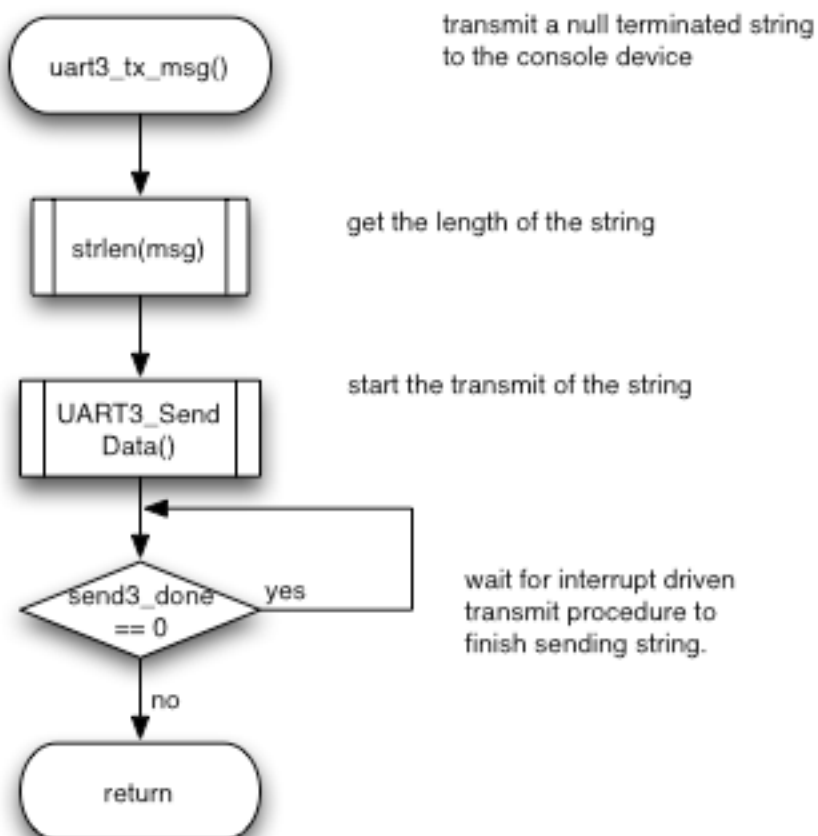
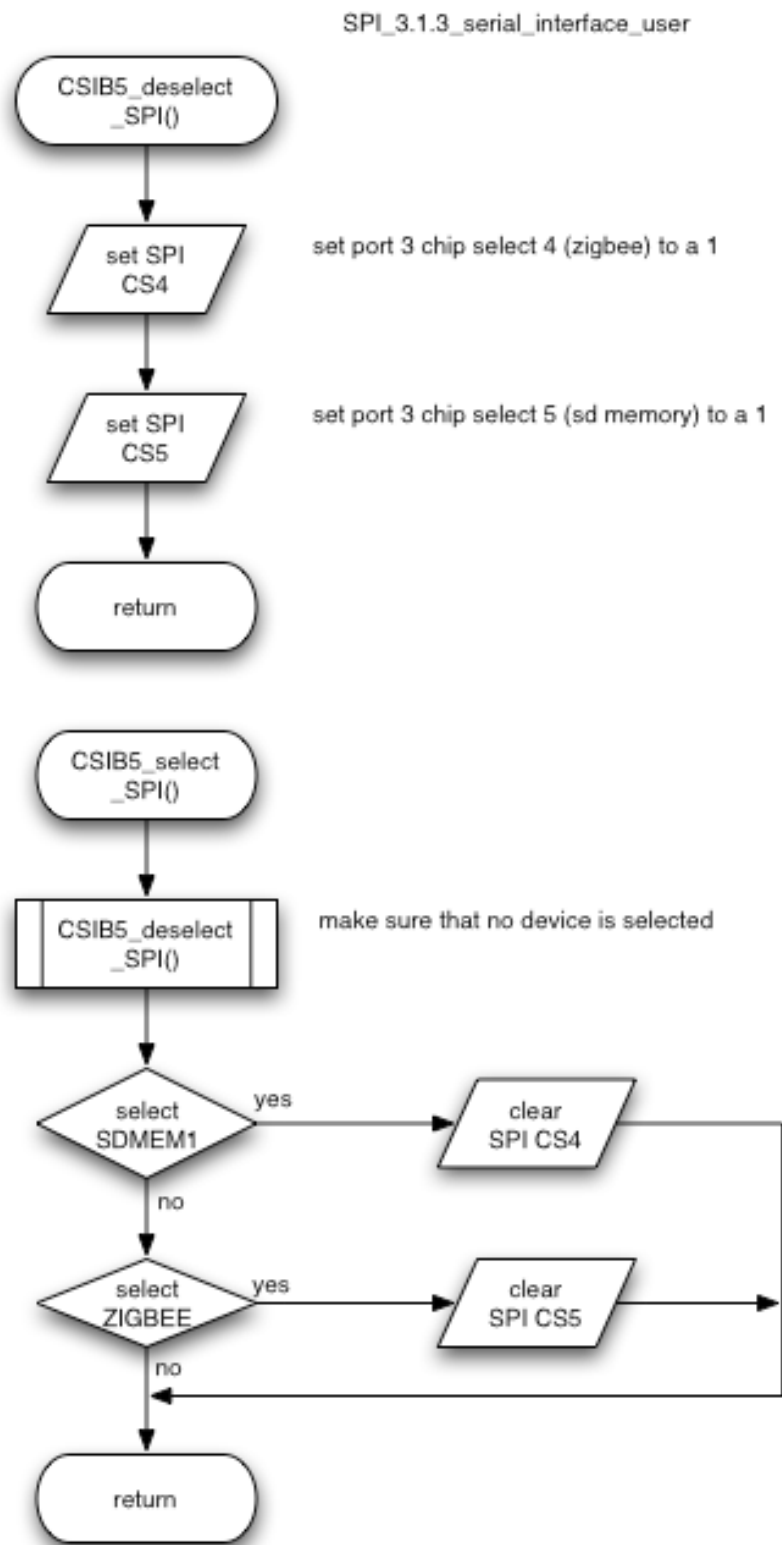
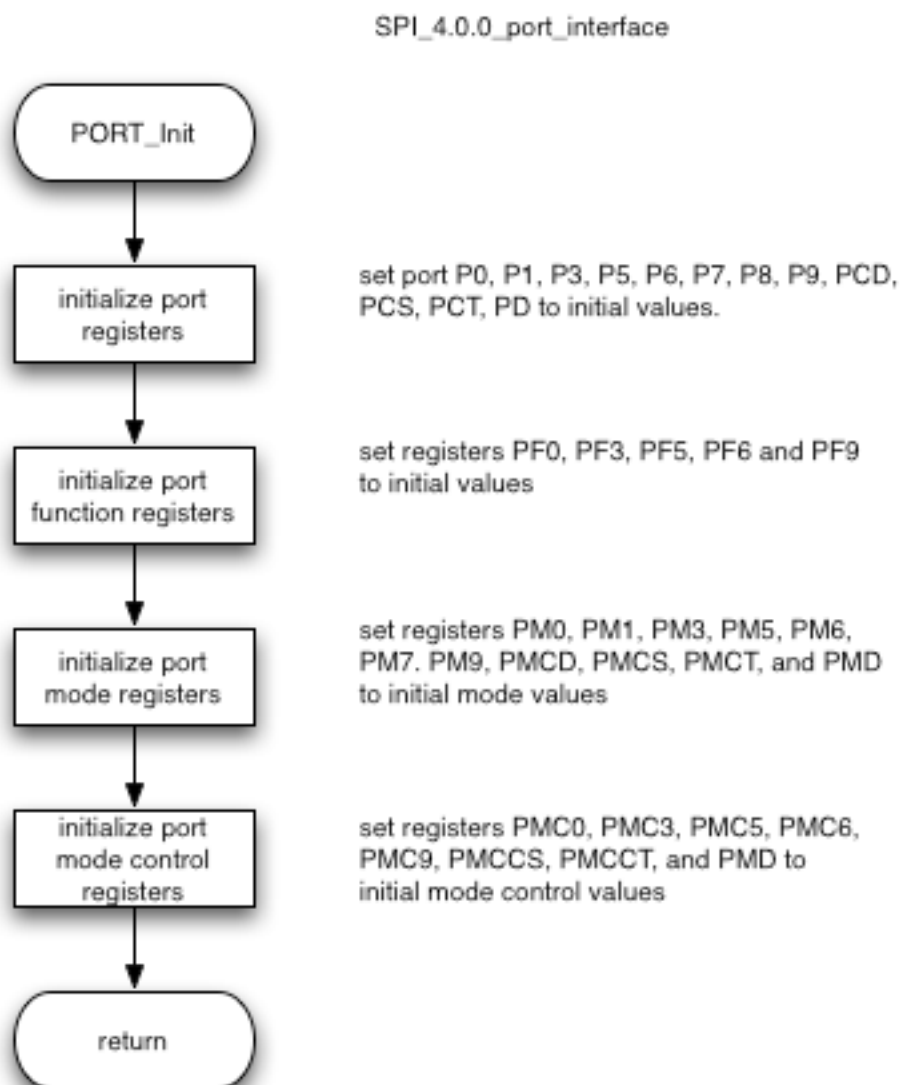


Figure 68. SPI_3.1.3_serial_interface_user



7.6 Port_interface

Figure 69. SPI_4.0.0_port_interface



7.7 LED_interface

Figure 70. SPI_4.1.0_led_interface



Figure 71. SPI_4.1.1_led_interface

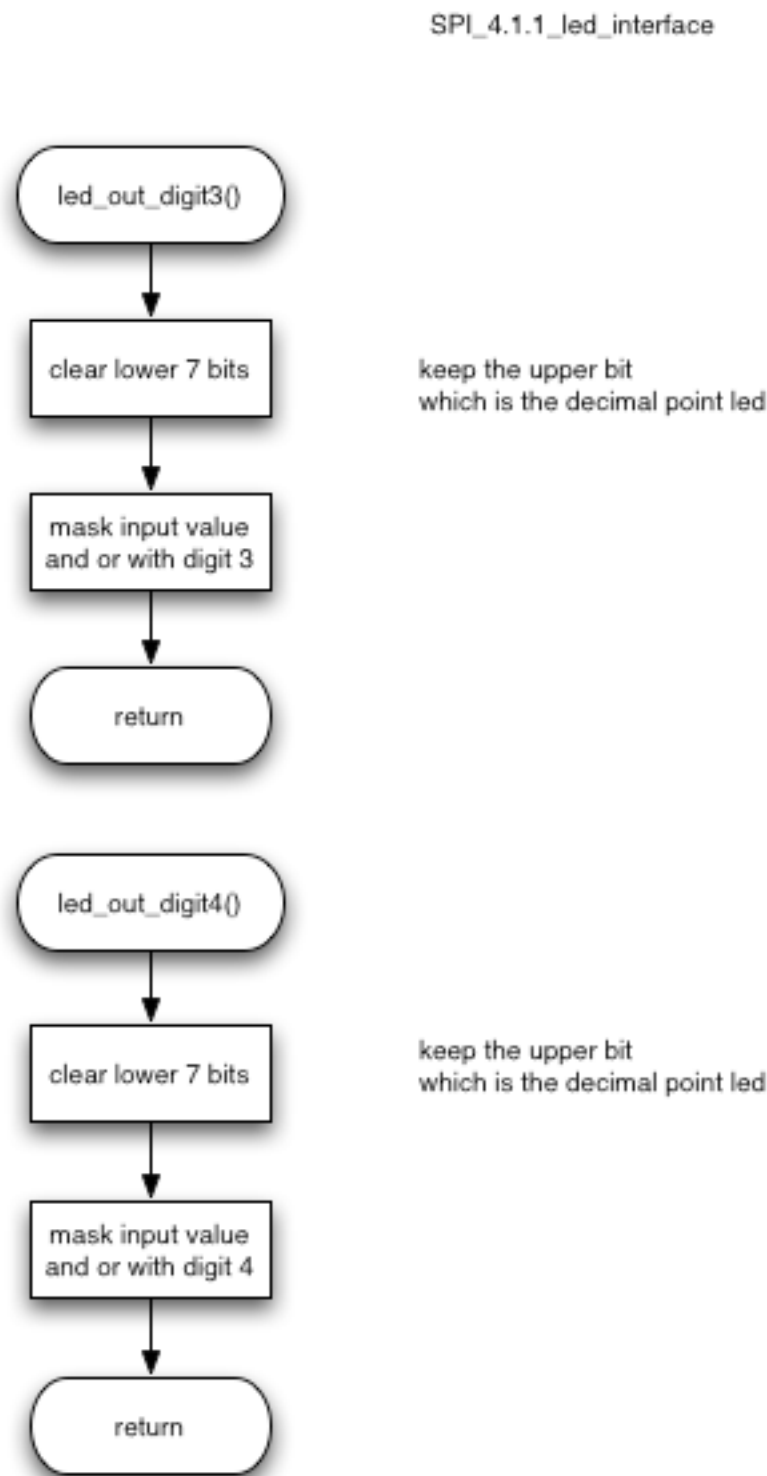


Figure 72. SPI_4.1.2_led_interface

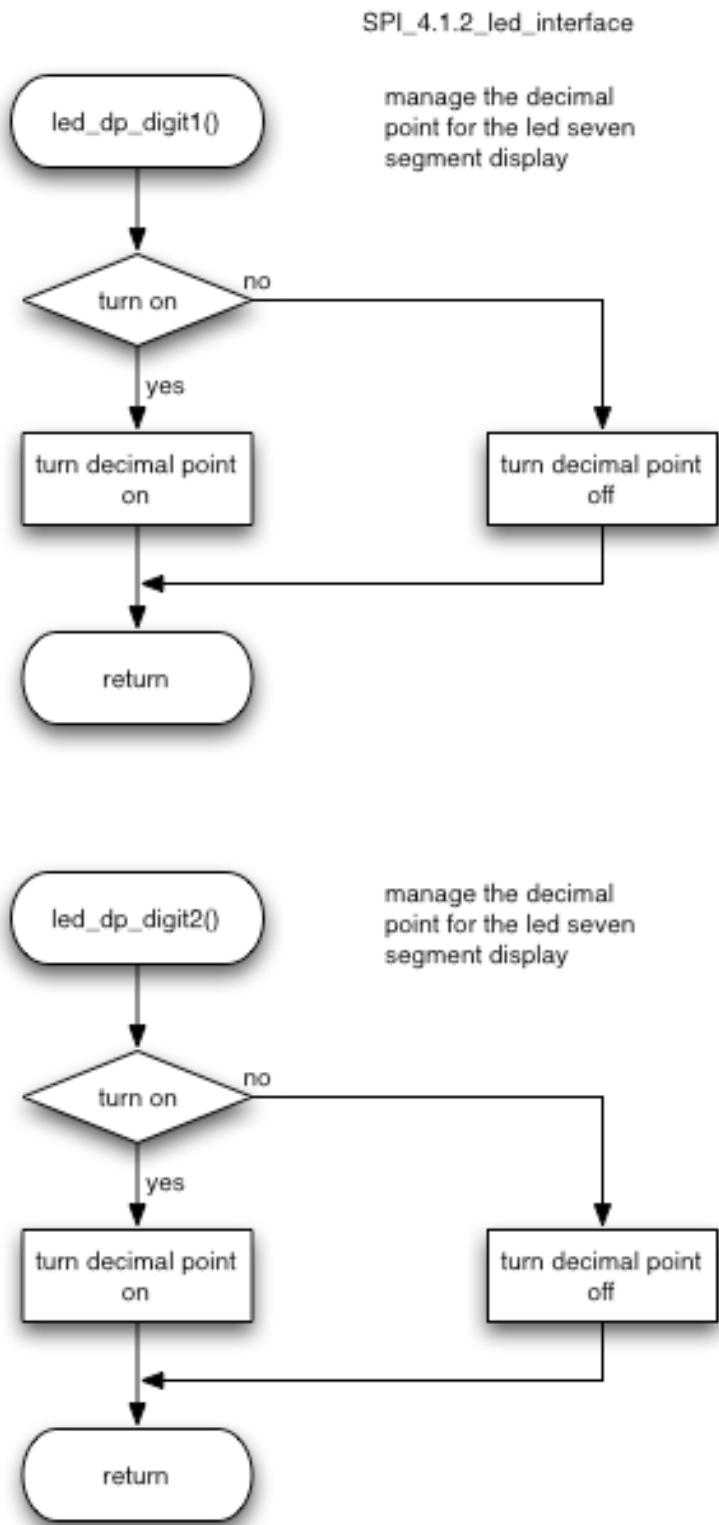


Figure 73. SPI_4.1.3_led_interface

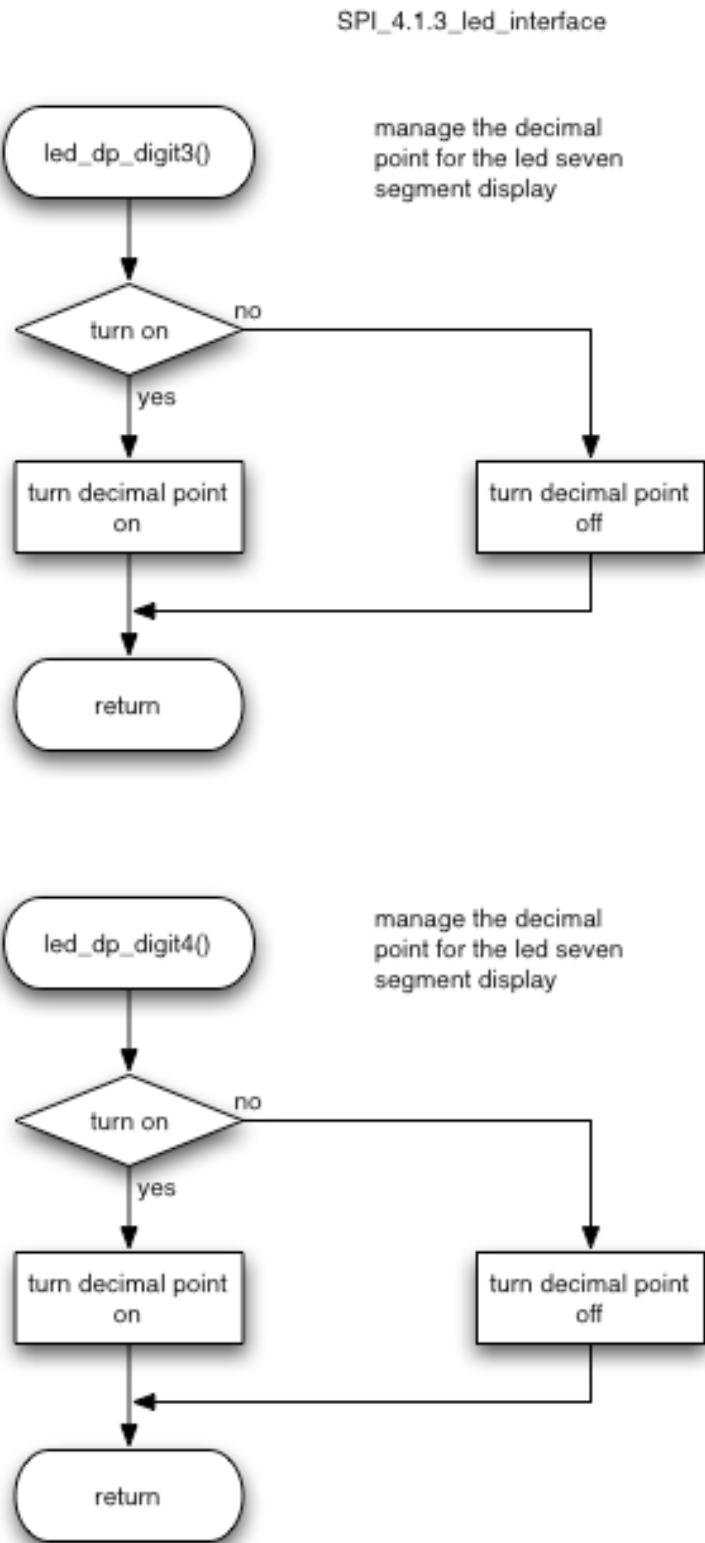


Figure 74. SPI_4.1.4_led_interface

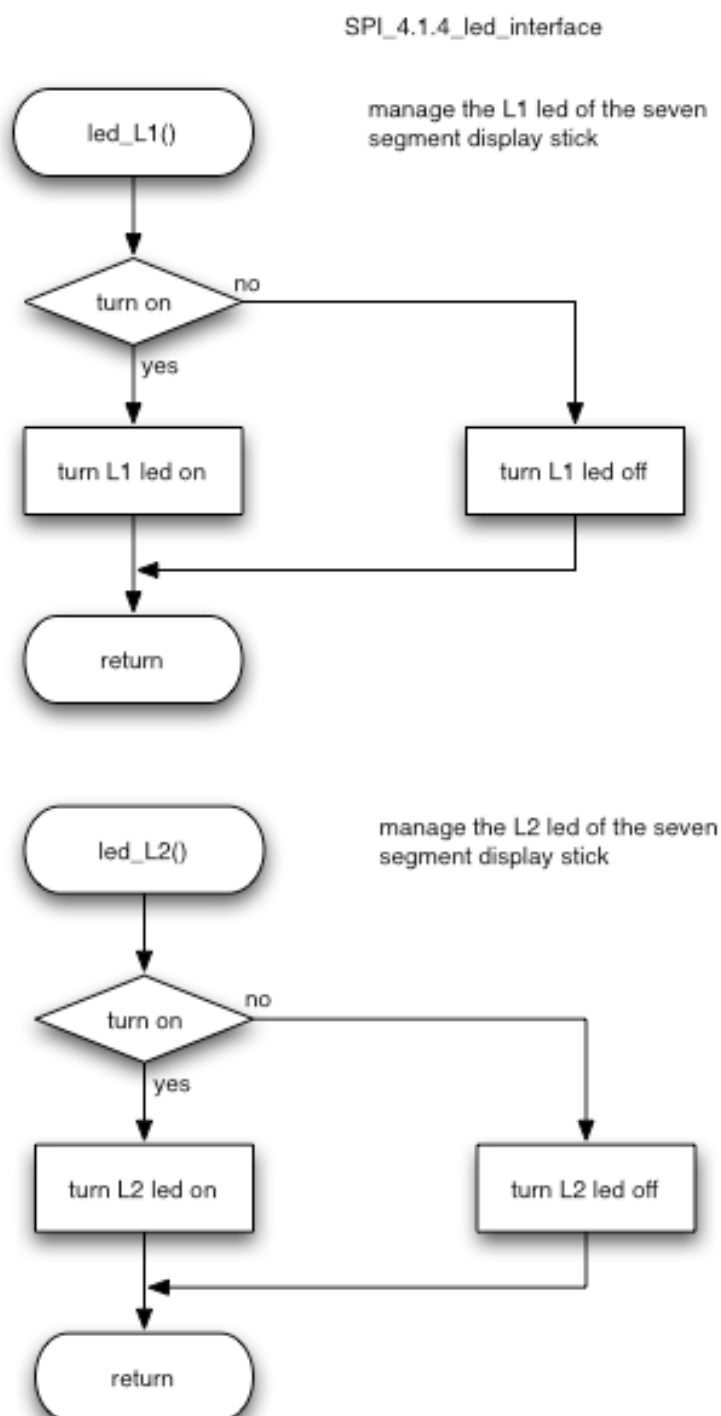


Figure 75. SPI_4.1.5_led_interface

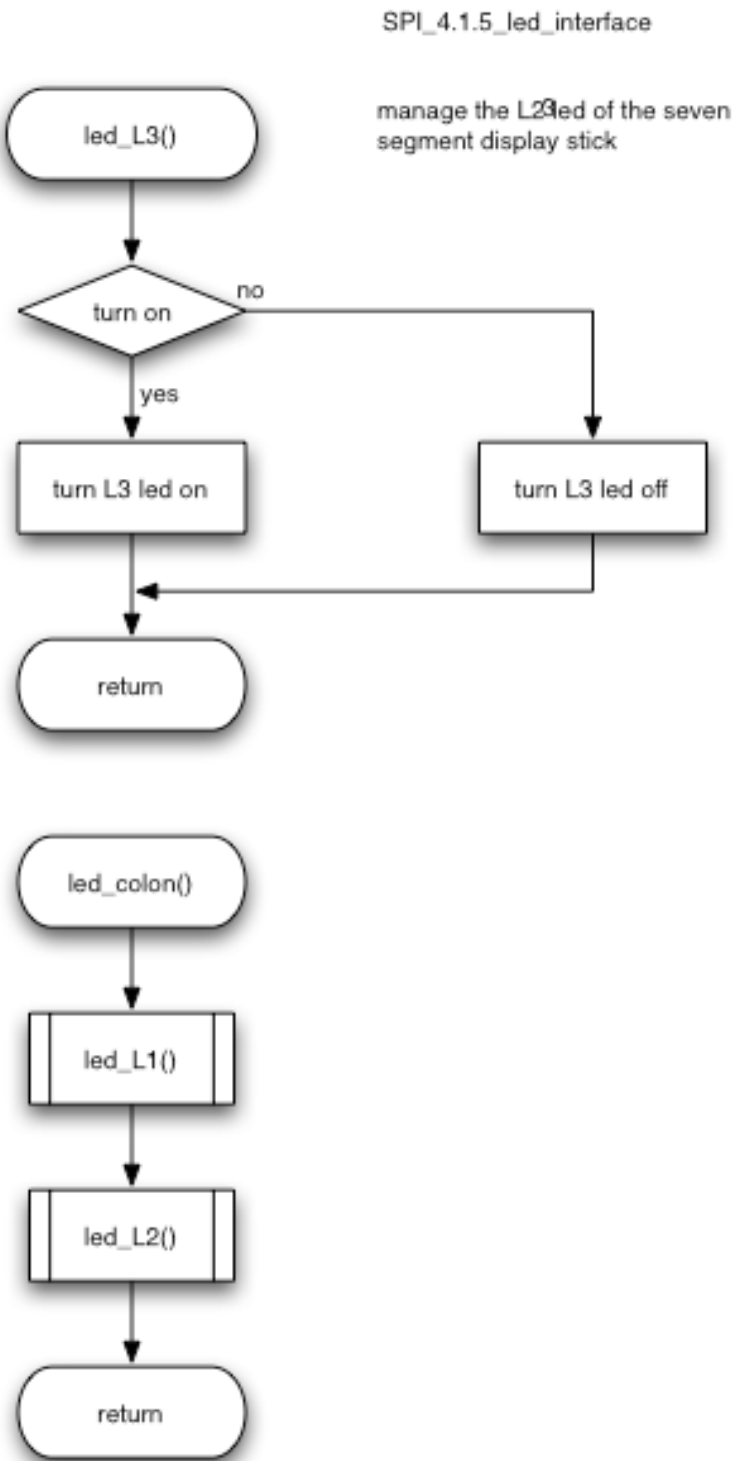


Figure 76. SPI_4.1.6_led_interface

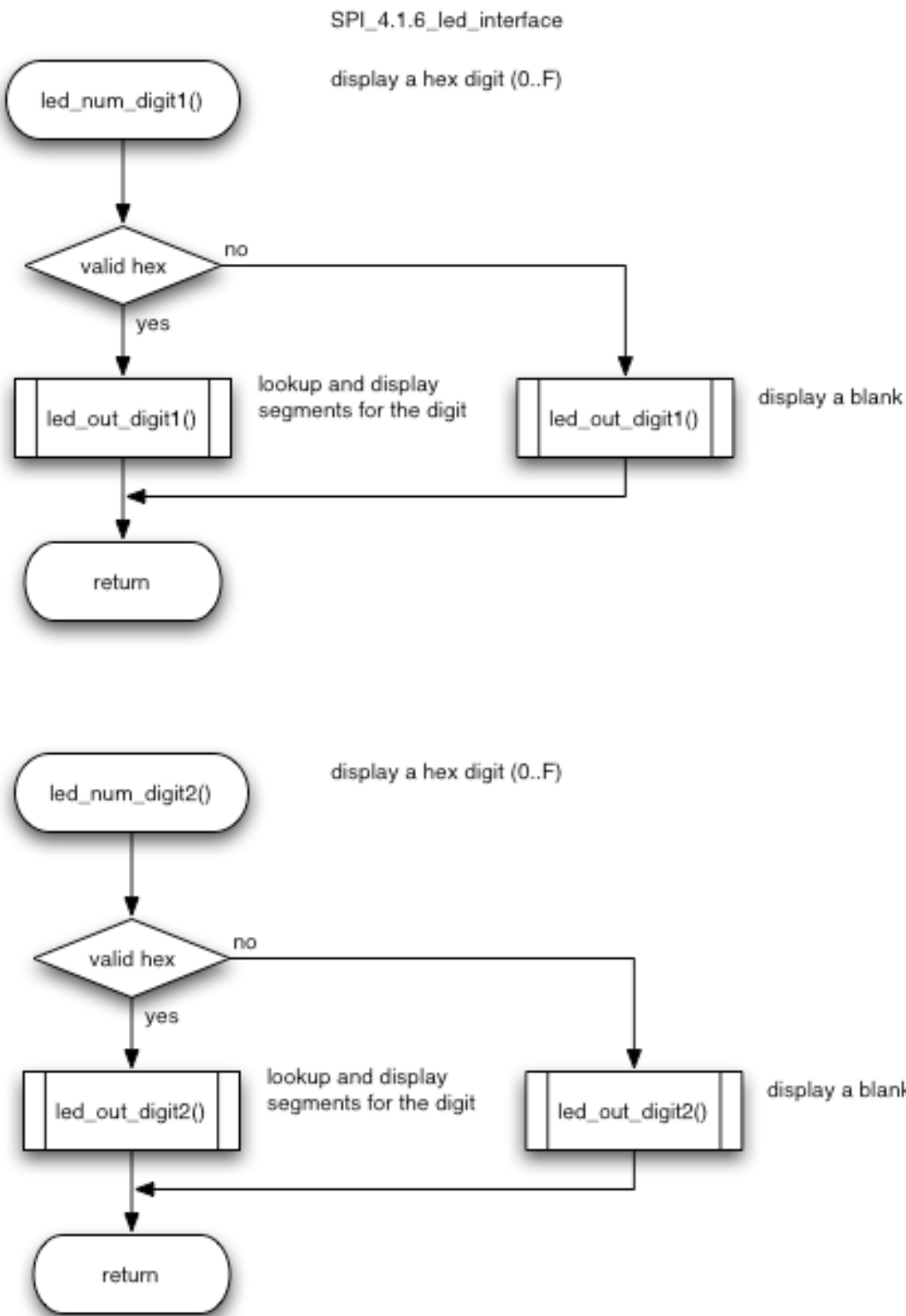


Figure 77. SPI_4.1.7_led_interface

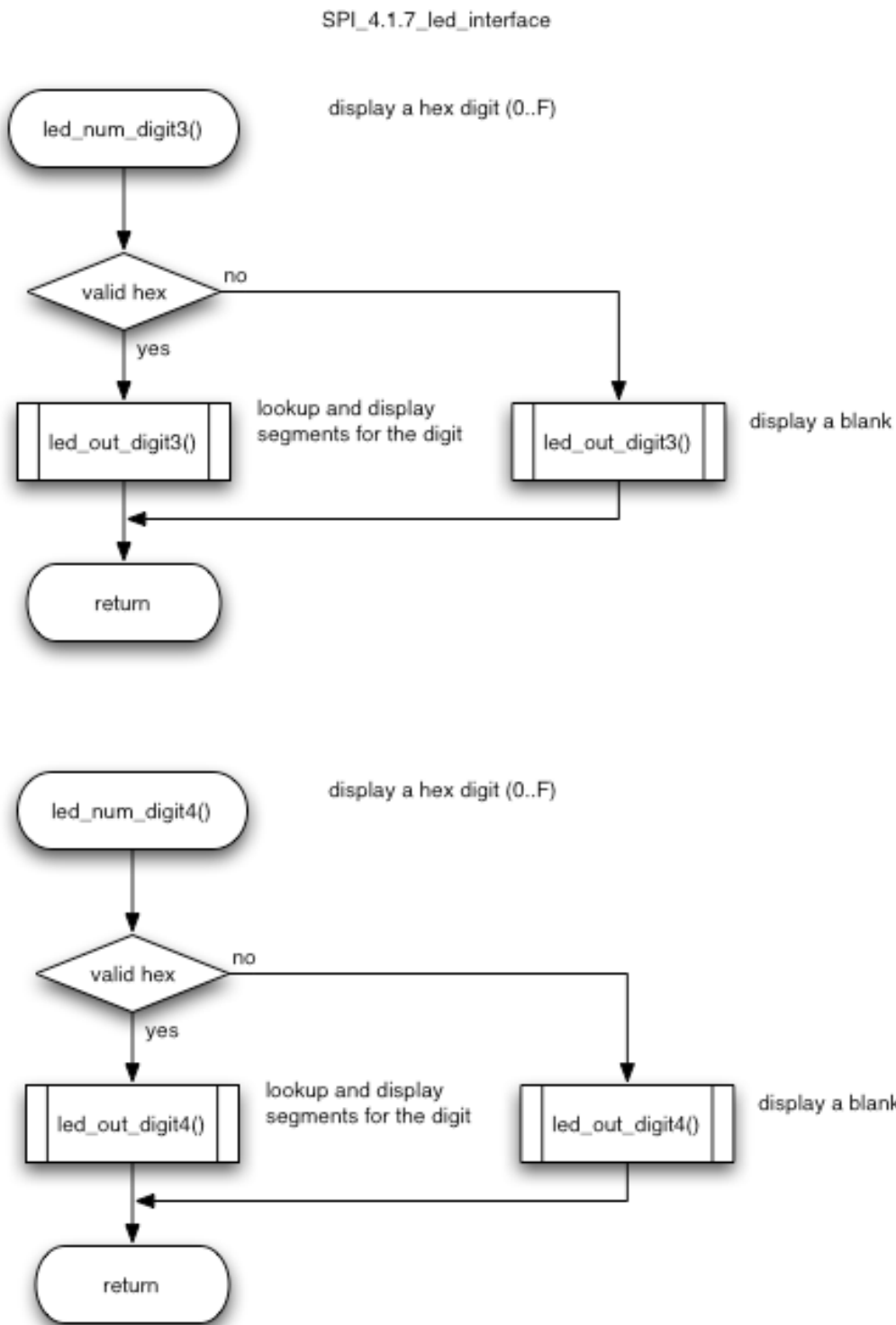


Figure 78. SPI_4.1.8_led_interface

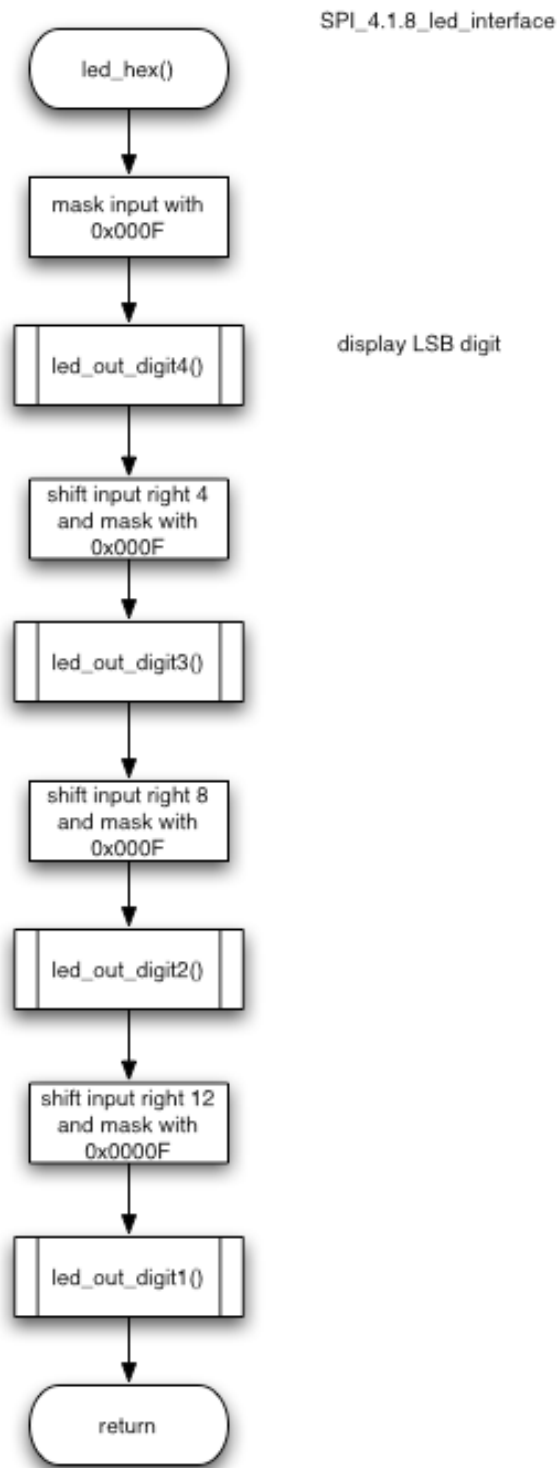


Figure 79. SPI_4.1.9_led_interface

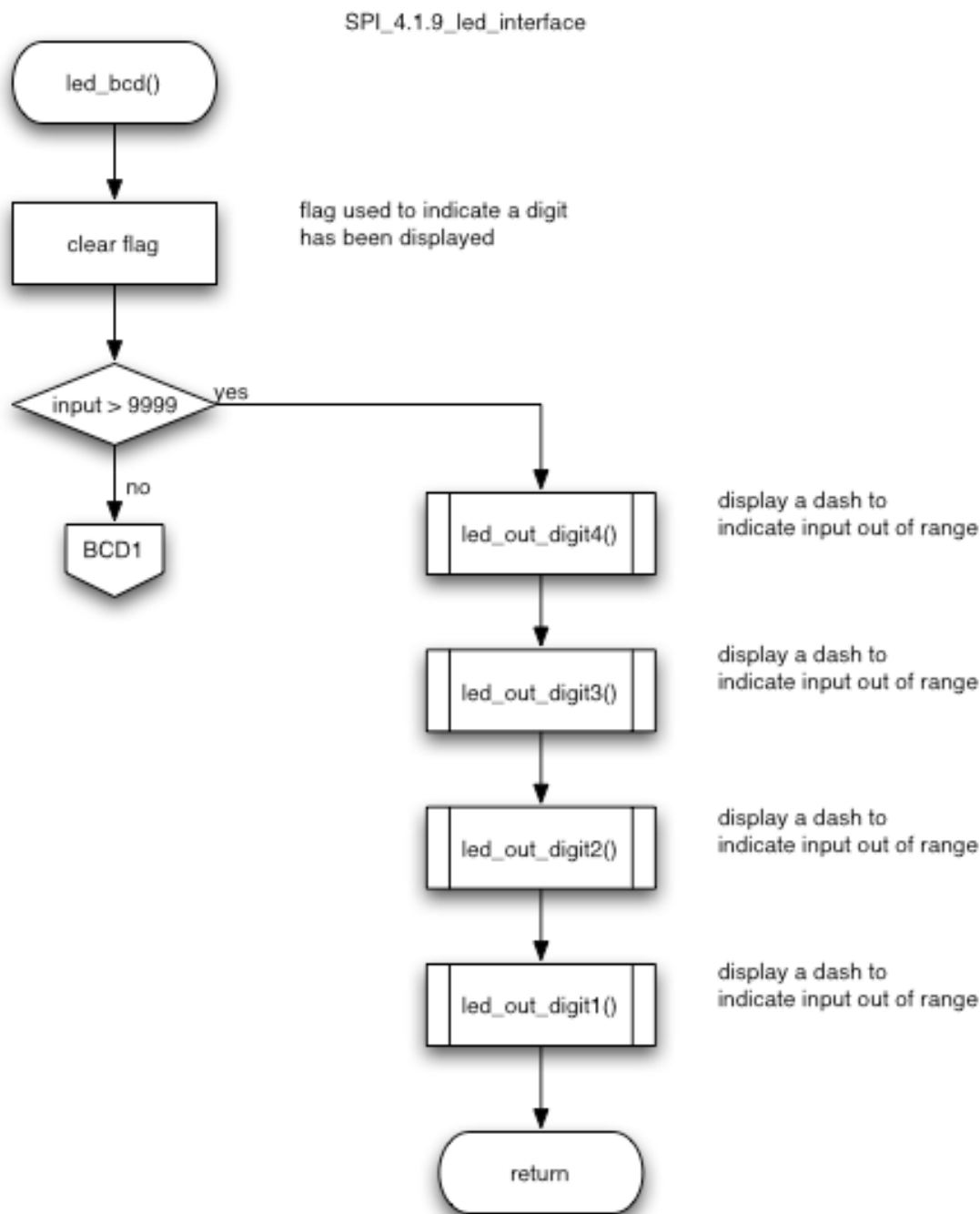


Figure 80. SPI_4.1.10_led_interface

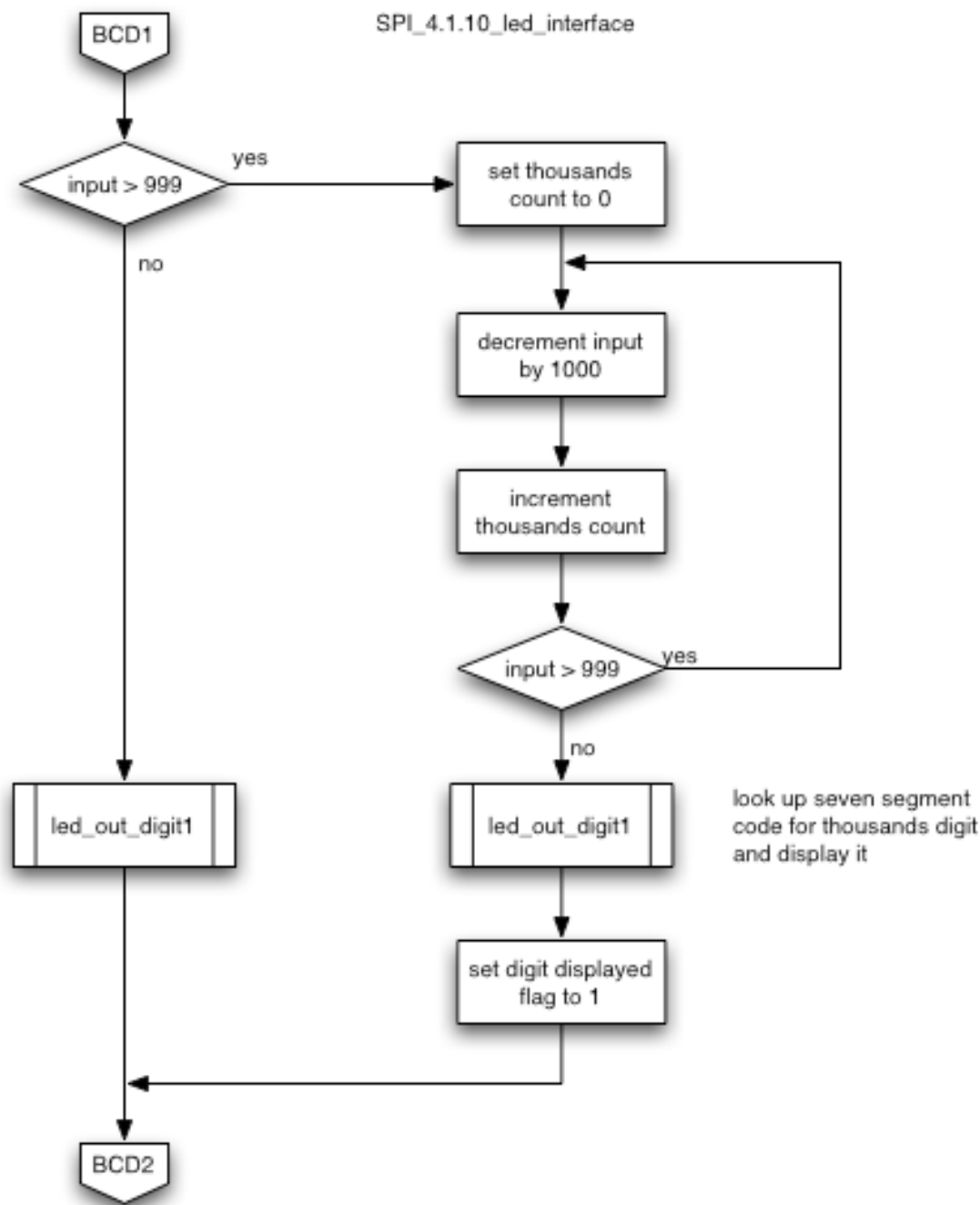


Figure 81. SPI_4.1.11_led_interface

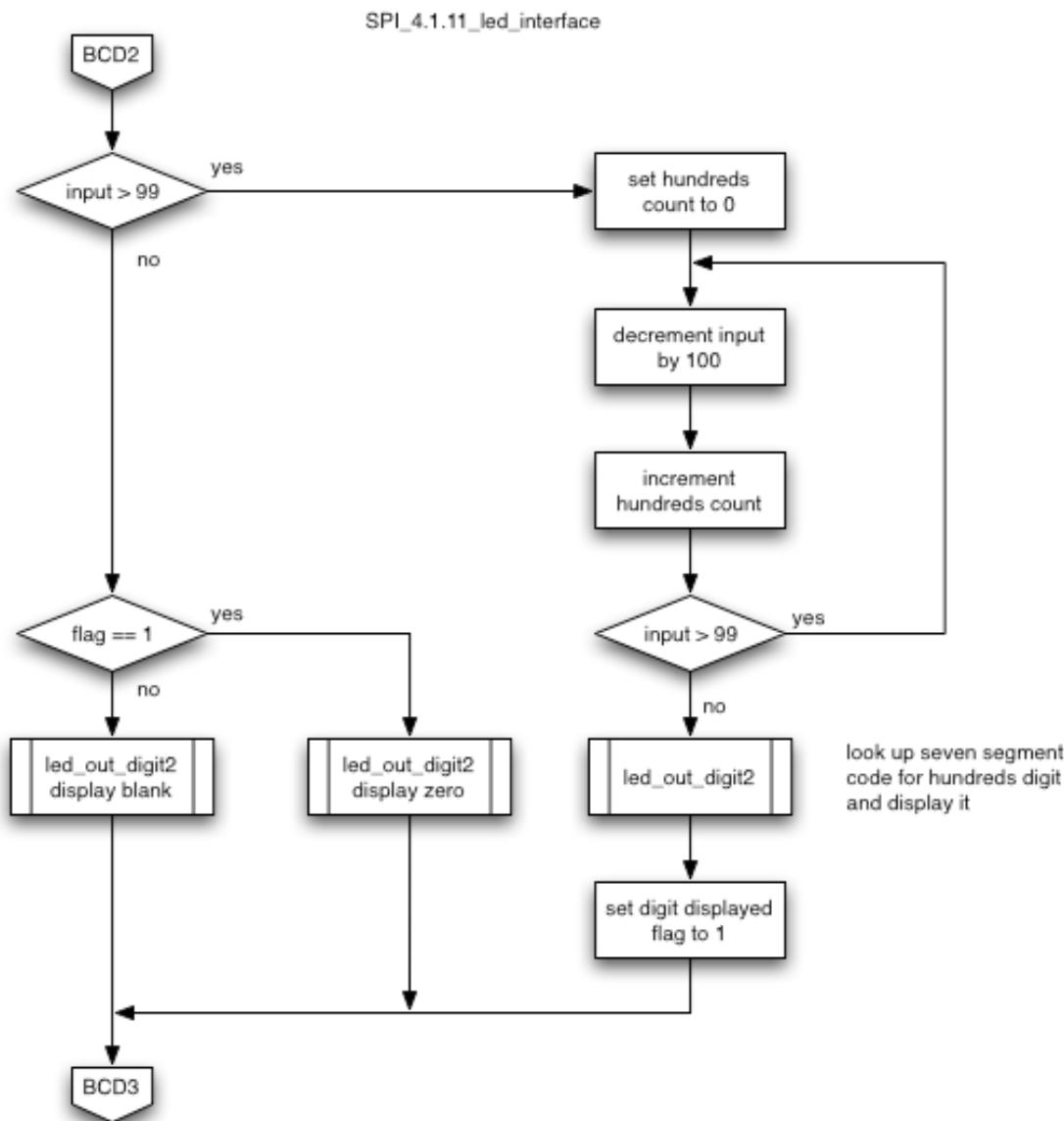


Figure 82. SPI_4.1.12_led_interface

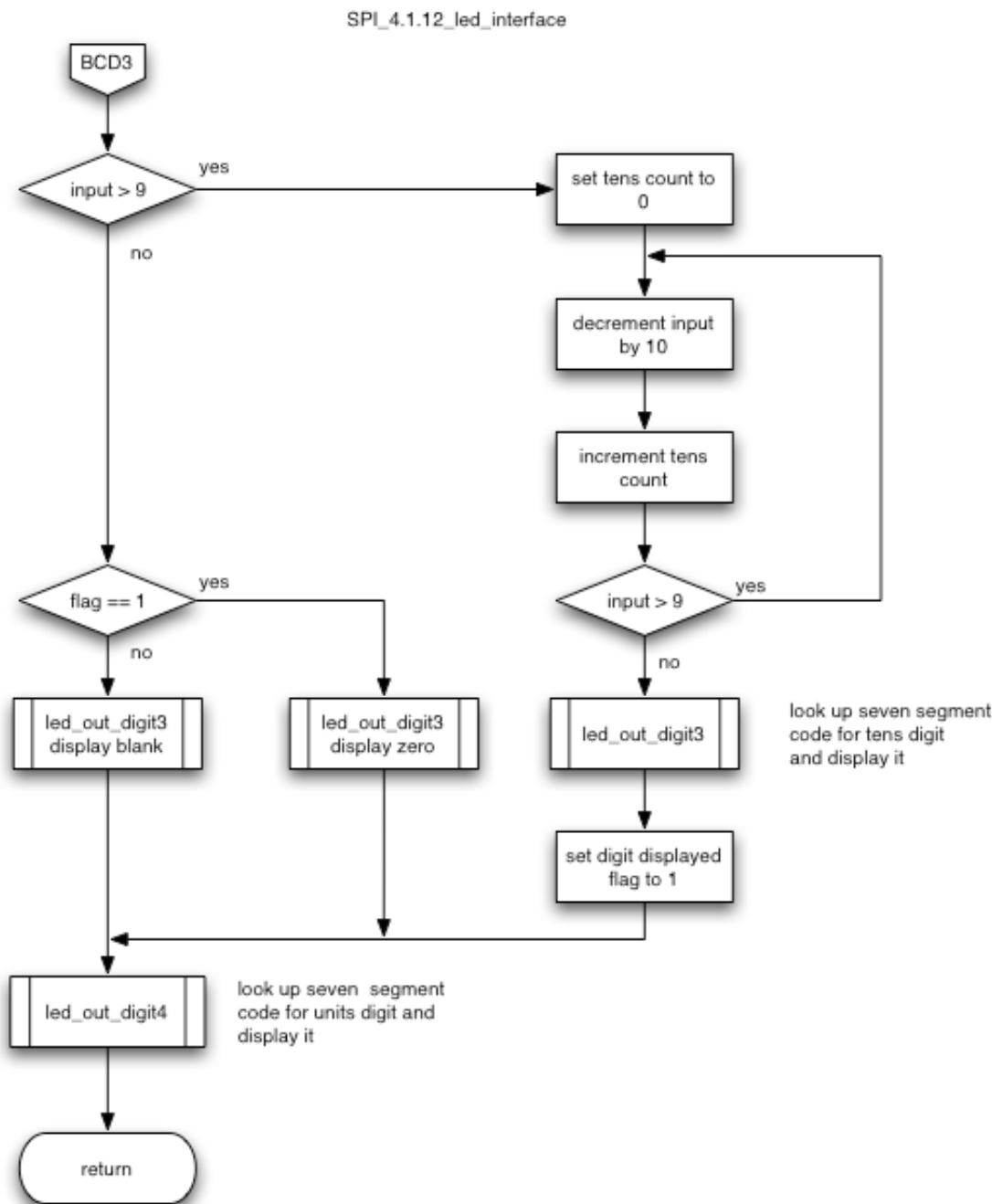
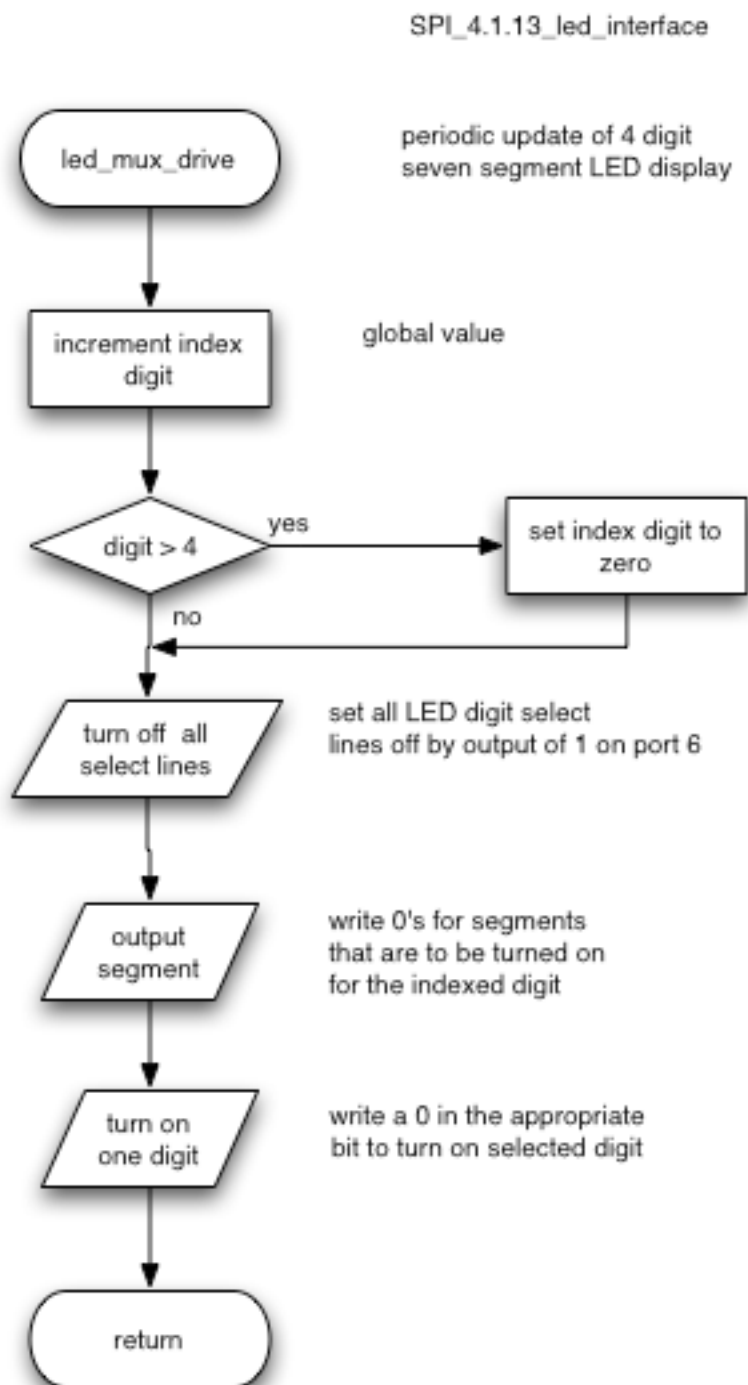


Figure 83. SPI_4.1.13_led_interface



7.8 Switch_interface

Figure 84. SPI_4.2.0_switch_interface

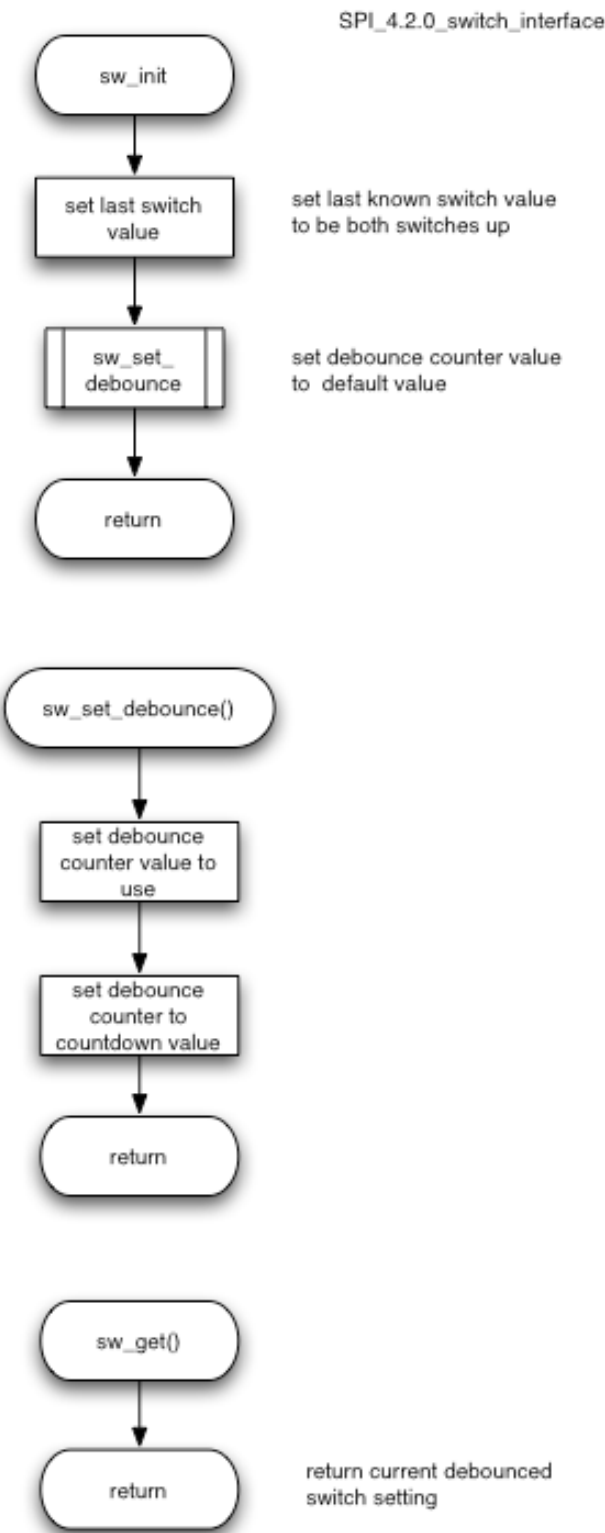
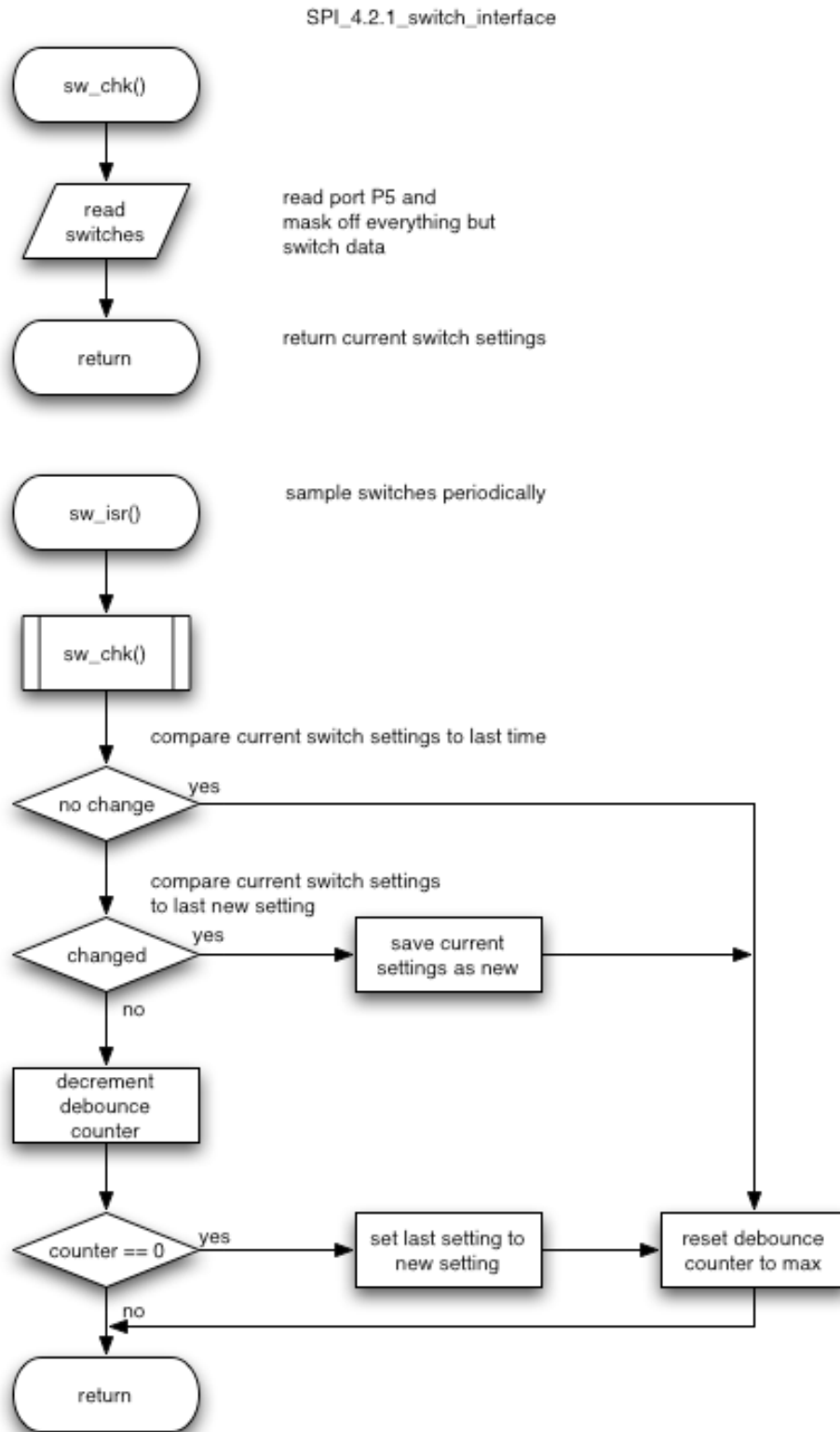


Figure 85. SPI_4.2.1_switch_interface



7.9 Timer_interface

Figure 86. SPI_5.0.0_timer_interface

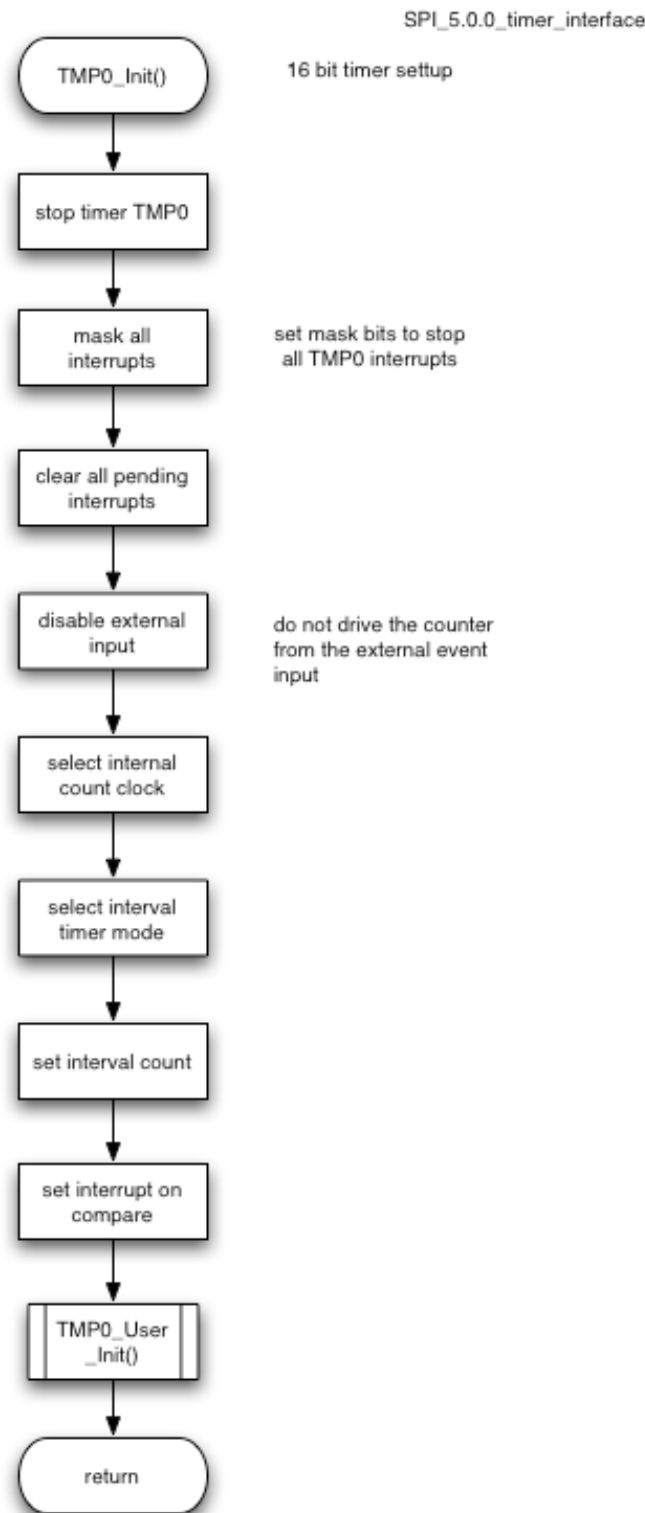


Figure 87. SPI_5.0.1_timer_interface

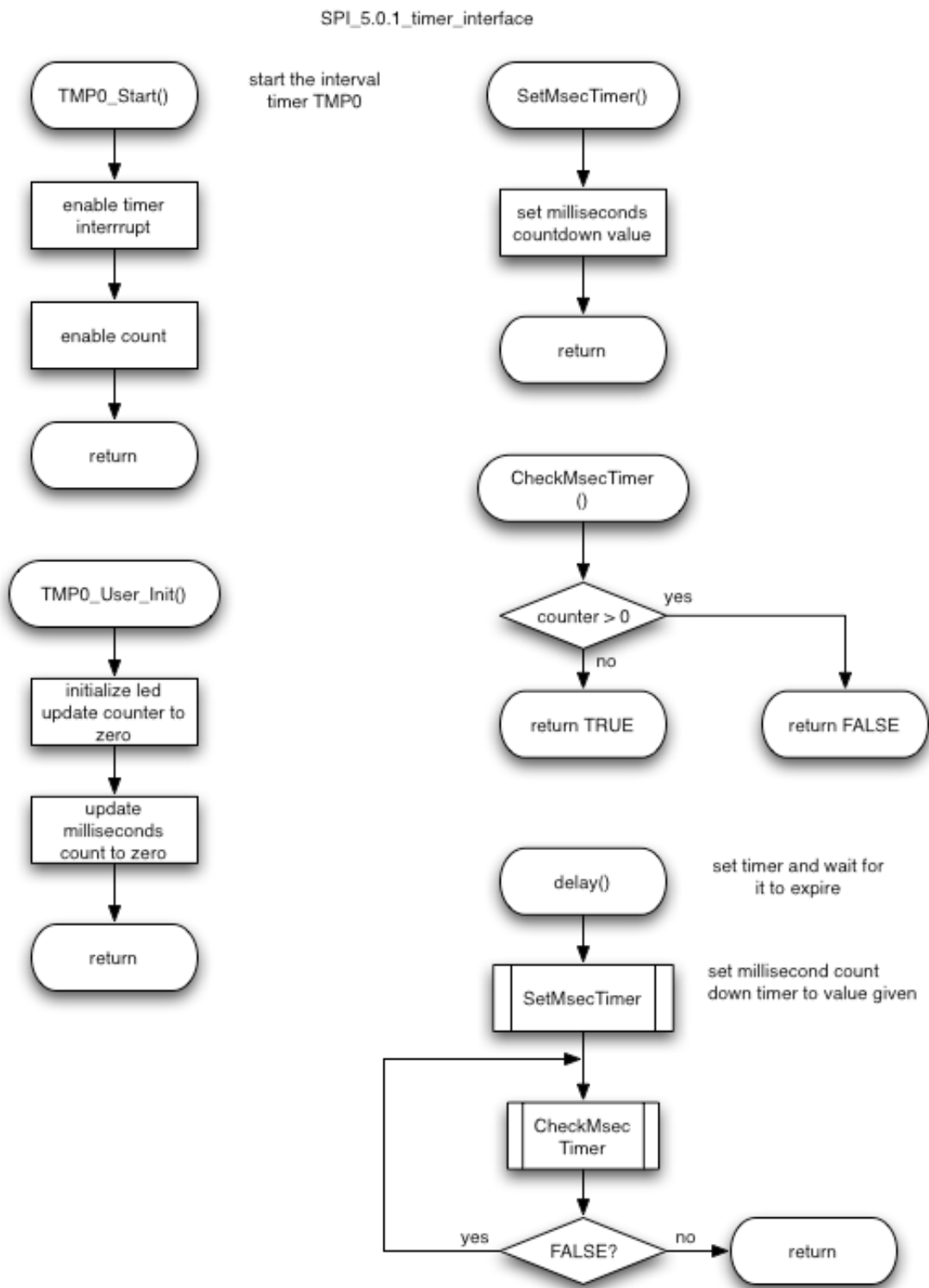
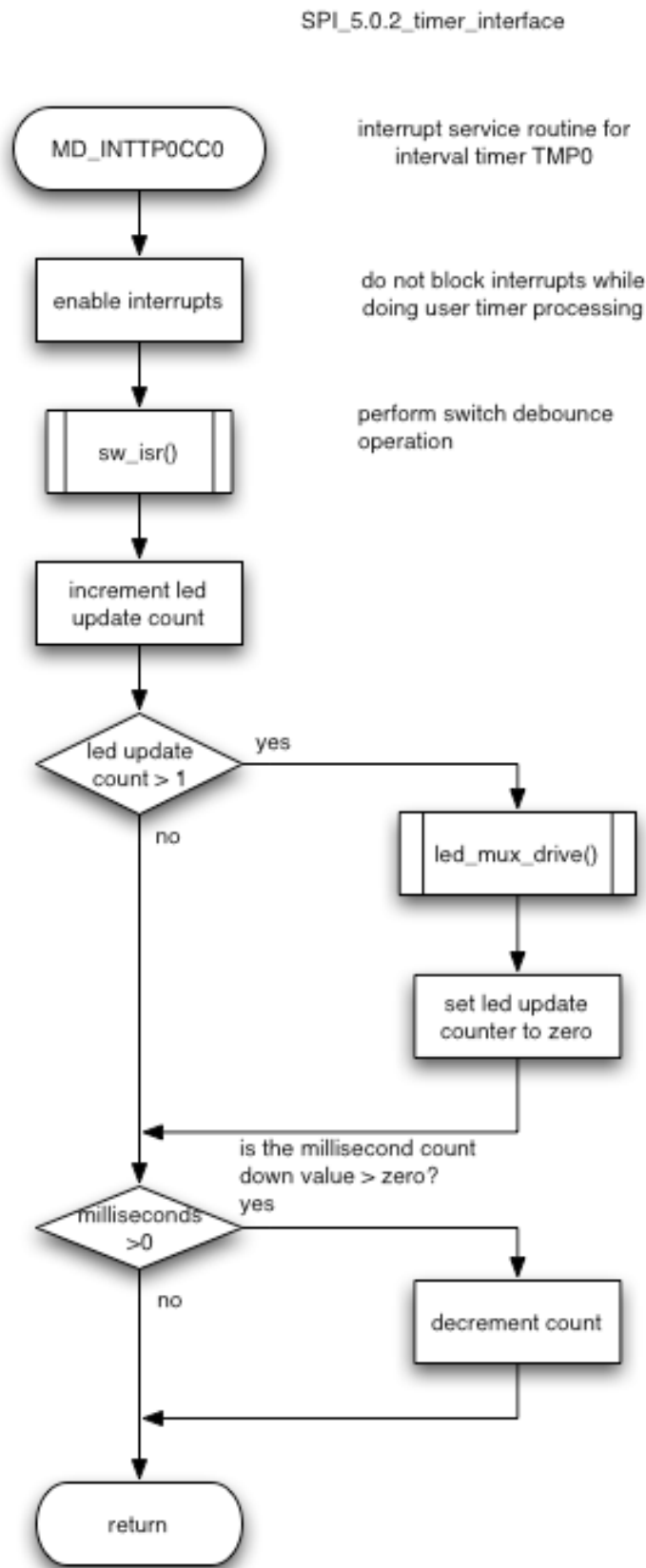


Figure 88. SPI_5.0.2_timer_interface



8. Appendix B — Source Code Listings

This appendix provides the following source code files:

- ◆ crte.s
- ◆ system.s
- ◆ inttab.s
- ◆ systeminit.c
- ◆ main.c
- ◆ sdmemory.c
- ◆ serial.c
- ◆ port.c
- ◆ led_vjj2.c
- ◆ sw_vjj2.c
- ◆ timer.c
- ◆ timer_user.c
- ◆ system.inc
- ◆ macrodriver.h
- ◆ sdmemory.h
- ◆ serial.h
- ◆ port.h
- ◆ led_vjj2.h
- ◆ sw_vjj2.h
- ◆ timer.h


```

#          sp-> -+-----+ __stack + STACKSIZE      __ebss
#                | monitor area | MRAMSEG
#                +-----+
#
#=====

#-----
# special symbols
#-----
.extern __tp_TEXT, 4
.extern __gp_DATA, 4
.extern __ep_DATA, 4
.extern __ssbss, 4
.extern __esbss, 4
.extern __sbss, 4
.extern __ebss, 4

#-----
# C program main function
#-----
.extern _SystemInit
.extern _main

.extern _Clock_Init

#-----
# for argv
#-----
.data
.size __argc, 4
.align 4
__argc:
.word 0
.size __argv, 4
__argv:
.word #.L16
.L16:
.byte 0
.byte 0
.byte 0
.byte 0

#-----
# dummy data declaration for creating sbss section
#-----
.sbss
.lcomm __sbss_dummy, 0, 0

#-----
# system stack
#-----
.set STACKSIZE, 0x800
.bss
.lcomm __stack, STACKSIZE, 4

#-----

```

```

# Monitor Area
#-----

#--Secures 2KB space for monitor ROM section
.section    "MonitorROM", const
.space      0x800, 0xff

#--Secures interrupt vector for debugging at 0x0060
.section    "DBG0"
.space      4, 0xff

-- Secures 16 byte space for mointor RAM section
.section    "MonitorRAM", bss
.lcomm      monitorramsym,16,4    -- defines monitorramsym symbol

#-----
# RESET handler
#-----

.section    "RESET", text
jr          __start

#-----
# start up
#     pointers:  tp - text pointer
#                gp - global pointer
#                sp - stack pointer
#                ep - element pointer
# exit status is set to r10
#-----

.text
.align      4
.globl      __start
.globl      __exit
.globl      __startend
.extern     __PROLOG_TABLE
__start:
mov     #__tp_TEXT, tp          -- set tp register
mov     #__gp_DATA, gp         -- set gp register offset
add     tp, gp                 -- set gp register
mov     #__stack+STACKSIZE, sp -- set sp register
mov     #__ep_DATA, ep         -- set ep register

.option    warning

#
mov     #__PROLOG_TABLE, r12    -- for prologue/epilogue runtime
ldsr    r12, 20                -- set CTBP (CALLT base pointer)

mov     1, r11                 -- on-chip debug mode
st.b    r11, PRCMD[r0]
st.b    r11, OCDM[r0]

nop
nop
nop
nop
nop

```

```

jarl  _Clock_Init, lp          -- call Clock_Init function

mov   #__ssbss, r13           -- clear sbss section
mov   #__esbss, r12
cmp   r12, r13
jnl   .L11
.L12:
st.w  r0, [r13]
add   4, r13
cmp   r12, r13
jl    .L12
.L11:
#
mov   #__sbss, r13            -- clear bss section
mov   #__ebss, r12
cmp   r12, r13
jnl   .L14
.L15:
st.w  r0, [r13]
add   4, r13
cmp   r12, r13
jl    .L15
.L14:
#

ld.w  $__argc, r6             -- set argc
movea $__argv, gp, r7        -- set argv
jarl  _SystemInit, lp         -- call SystemInit function
jarl  _main, lp               -- call main function
__exit:
halt                                     -- end of program
__startend:
#                                     #
#----- end of start up module -----#
#                                     #

```


8.2 system.s**FILE ID: system.s**

```

--/*
--*****
--**
--** This device driver was created by Applilet for the V850ES/JG2 and V850ES/JJ2
--** 32-Bit Single-Chip Microcontrollers
--**
--** Copyright(C) NEC Electronics Corporation 2002-2006
--** All rights reserved by NEC Electronics Corporation
--**
--** This program should be used on your own responsibility.
--** NEC Electronics Corporation assumes no responsibility for any losses incurred
--** by customers or third parties arising from the use of this file.
--**
--** Filename : system.s
--** Abstract : This file implements a device driver for the SYSTEM module
--** APilib: v850esJx2.lib V1.50 [23 Feb. 2006]
--**
-- Device: uPD70F3717
--
-- Compiler: NEC/CA850
--
--*****
--*/
.include "system.inc"
.section "SECURITY_ID", text
.byte CG_SECURITY0      -- Security ID head
.byte CG_SECURITY1
.byte CG_SECURITY2
.byte CG_SECURITY3
.byte CG_SECURITY4
.byte CG_SECURITY5
.byte CG_SECURITY6
.byte CG_SECURITY7
.byte CG_SECURITY8
.byte CG_SECURITY9      -- Security ID tail
    .text
.globl _Clock_Init
.align 4

--/*
--**-----
--**
--** Abstract:
--** Init the Clock Generator and Watchdog timer 2
--**
--** Parameters:
--** None
--**
--** Returns:
--** None
--**-----
--*/
_Clock_Init:

```

```

add    -8, sp
st.w   r11, 0[sp]
st.w   r12, 4[sp]

ld.b   DCHC0[r0], r11          -- stop DMA0
add    -1, sp
st.b   r11, 0[sp]
andi   0xfe, r11, r11
st.b   r11, DCHC0[r0]

ld.b   DCHC1[r0], r11          -- stop DMA1
add    -1, sp
st.b   r11, 0[sp]
andi   0xfe, r11, r11
st.b   r11, DCHC1[r0]

ld.b   DCHC2[r0], r11          -- stop DMA2
add    -1, sp
st.b   r11, 0[sp]
andi   0xfe, r11, r11
st.b   r11, DCHC2[r0]

ld.b   DCHC3[r0], r11          -- stop DMA3
add    -1, sp
st.b   r11, 0[sp]
andi   0xfe, r11, r11
st.b   r11, DCHC3[r0]

-- disable interrupt
stsr   5, r11
ori    0xa0, r11, r11
ldsr   r11, 5

mov    r0, r11
st.b   r11, PRCMD[r0]
st.b   r11, CLM[r0]          --disable clock monitor function

nop
nop
nop
nop
nop
ld.b   PCC[r0], r12
andi   0xf8, r12, r12
or     r12, r11
st.b   r11, PRCMD[r0]
st.b   r11, PCC[r0]

nop
nop
nop
nop
nop
-- Sub clock -> Main clock start
-- stop Main clock
st.b   r0, PRCMD[r0]
clr1   6, PCC[r0]

```

```

-- this wait loop per 250usec
movea 0x1000, r0, r11

__CG_LOOP2:
nop
nop
nop

addi -1, r11, r11
cmp r0, r11
bnz __CG_LOOP2
st.b r0, PRCMD[r0]

clr1 3, PCC[r0]
__CG_LOOP3:
-- Check CLS
tst1 4, PCC[r0]
bnz __CG_LOOP3

-- Sub -> Main end
-- enable RingOSC
clr1 0, RCM[r0]
mov 0x0a, r11 -- fxx = 4*fxx
st.b r11, PRCMD[r0]
st.b r11, CKC[r0]
nop
nop
nop
nop
nop
nop
-- PLL start
set1 0, PLLCTL[r0]
-- PLL work
__CG_LOOP4:
ld.b LOCKR[r0], r11
cmp r0, r11
bnz __CG_LOOP4
set1 1, PLLCTL[r0]
-- enable interrupt
stsr 5, r11
andi 0x5f, r11, r11
ldsr r11, 5

ld.b 0[sp], r11 -- recover DMA3
add 1, sp
st.b r11, DCHC3[r0]

ld.b 0[sp], r11 -- recover DMA2
add 1, sp
st.b r11, DCHC2[r0]

ld.b 0[sp], r11 -- recover DMA1
add 1, sp
st.b r11, DCHC1[r0]

ld.b 0[sp], r11 -- recover DMA0
add 1, sp

```

```
st.b  r11, DCHC0[r0]
-- oscollation stabilization time
-- selection clock
-- 2^16/fx
mov   0x6, r11
st.b  r11, OSTS[r0]
mov   0x1f, r11
st.b  r11, WDTM2[r0]
-- pop
ld.w  0[sp], r11
ld.w  4[sp], r12
add   8, sp

      jmp   [lp]
```

8.3 inttab.s**FILE ID: inttab.s**

```

--/*
--*****
--**
--** This device driver was created by Applilet for the V850ES/JG2 and V850ES/JJ2
--** 32-Bit Single-Chip Microcontrollers
--**
--** Copyright(C) NEC Electronics Corporation 2002-2006
--** All rights reserved by NEC Electronics Corporation.
--**
--** This program should be used on your own responsibility.
--** NEC Electronics Corporation assumes no responsibility for any losses incurred
--** by customers or third parties arising from the use of this file.
--**
--** Filename : inttab.s
--** Abstract : This file implements interrupt vector table
--** APilib: v850esJx2.lib V1.50 [23 Feb. 2006]
--**
--*****
--*/

--INT vector

-----
-- variable initiate
-----

    --.section "RESET", text
    --jr      __start

    .section "NMI", text          --nmi pin input          0x0010
    reti

    .section "INTWDT2", text      --WDT2 OVF nonmaskable 0x0020
    reti

    .section "TRAP00", text      --TRAP instruction        0x0040
    .globl    __trap00
__trap00:
    reti

    .section "TRAP10", text      --TRAP instruction        0x0050
    .globl    __trap01
__trap01:
    reti
--** this section "DBG0" is defined in crte.s
-- .section "ILGOP", text        --illegal op code        0x0060
-- .globl    __ilgop
--__ilgop:
-- .space          4, 0xff
-- #reti

    .section "INTLVI", text      --INTLVI                0x0080
    reti
# .section "INTP0", text        --INTP0 pin
# reti

```

```

.section "INTP1", text          --INTP1 pin
reti

.section "INTP2", text          --INTP2 pin
reti

.section "INTP3", text          --INTP3 pin
reti

.section "INTP4", text          --INTP4 pin
reti

.section "INTP5", text          --INTP5 pin
reti

.section "INTP6", text          --INTP6 pin
reti

.section "INTP7", text          --INTP7 pin
reti

.section "INTTQ00V", text      --TQ00V          0x0110
reti
.section "INTTQ0CC0", text      --TQ0CC0
reti
.section "INTTQ0CC1", text      --TQ0CC1
reti
.section "INTTQ0CC2", text      --TQ0CC2
reti
.section "INTTQ0CC3", text      --TQ0CC3
reti
.section "INTTP00V", text      --TP00V          0x0160
reti
-- .section "INTTP0CC0", text      --TP0CC0
-- reti
.section "INTTP0CC1", text      --TP0CC1
reti
.section "INTTP10V", text      --TP10V
reti
.section "INTTP1CC0", text      --TP1CC0
reti
.section "INTTP1CC1", text      --TP1CC1
reti
.section "INTTP20V", text      --TP20V
reti
.section "INTTP2CC0", text      --TP2CC0
reti
.section "INTTP2CC1", text      --TP2CC1
reti
.section "INTTP30V", text      --TP30V
reti
.section "INTTP3CC0", text      --TP3CC0
reti
.section "INTTP3CC1", text      --TP3CC1
reti
.section "INTTP40V", text      --TP40V
reti
.section "INTTP4CC0", text      --TP4CC0

```

```

reti
.section "INTTP4CC1", text      --TP4CC1
reti
.section "INTTP5OV", text      --TP5OV
reti

.section "INTTP5CC0", text      --TP5CC0
reti

.section "INTTP5CC1", text      --TP5CC1

reti

.section "INTTM0EQ0", text      --TM0EQ0          0x0280
reti

.section "INTCB0R", text      --INTCB0R/INTIIC1    0x0290
.space          4, 0xff
#reti
.section "INTCB0T", text      --INTCB0T          0x02A0
.space          4, 0xff
#reti
.section "INTCB1R", text      --INTCB1R

reti

.section "INTCB1T", text      --INTCB1T

reti

.section "INTCB2R", text      --INTCB2R

reti

.section "INTCB2T", text      --INTCB2T

reti

.section "INTCB3R", text      --INTCB3R

reti

.section "INTCB3T", text      --INTCB3T

reti

-- .section "INTCB4R", text      --INTUA0R/INTCB4R    used by minicube2
-- reti
-- .section "INTCB4T", text      --INTUA0T/INTCB4T
-- reti
.section "INTIIC2", text      --INTUA1R/INTIIC2

reti

.section "INTUA1T", text      --INTUA1T

reti

.section "INTIIC0", text      --INTUA2R/INTIIC0

reti

.section "INTUA2T", text      --INTUA2T

reti

.section "INTAD", text      --INTAD

reti

```

```

.section "INTDMA0", text      --INTDMA0
reti

.section "INTDMA1", text      --INTDMA1
reti

.section "INTDMA2", text      --INTDMA2
reti

.section "INTDMA3", text      --INTDMA3
reti

.section "INTKR", text        --INTKR
reti
.section "INTP8", text        --INTP8
reti

.section "INTTP6OV", text     --INTTP6OV
reti
.section "INTTP6CC0", text    --INTTP6CC0
reti
.section "INTTP6CC1", text    --INTTP6CC1
reti
.section "INTTP7OV", text     --INTTP7OV
reti
.section "INTTP7CC0", text    --INTTP7CC0
reti
.section "INTTP7CC1", text    --INTTP7CC1
reti
.section "INTTP8OV", text     --INTTP8OV
reti
.section "INTTP8CC0", text    --INTTP8CC0
reti
.section "INTTP8CC1", text    --INTTP8CC1
reti
-- .section "INTCB5R", text    --INTCB5R          0x0510
-- reti
.section "INTWTI", text       --INTWTI          0x03E0
reti

-- end of file

```


8.4 systeminit.c**FILE ID: systeminit.c**

```

/*
*****
**
** This device driver was created by Applilet for the V850ES/JG2 and V850ES/JJ2
** 32-Bit Single-Chip Microcontrollers
**
** Copyright(C) NEC Electronics Corporation 2002-2006
** All rights reserved by NEC Electronics Corporation
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses incurred
** by customers or third parties arising from the use of this file.
**
** Filename : systeminit.c
** Abstract : This file implements macro initiate
** APIlib: v850esJx2.lib V1.50 [23 Feb. 2006]
**
** Device: uPD70F3721
**
** Compiler: NEC/CA850
**
*****
*/
/*
** *****
** Include files
** *****
*/
#include "macrodriver.h"
#include "port.h"
#include "watchtimer.h"
#include "serial.h"
/*
** *****
** MacroDefine
** *****
*/
extern unsigned long _S_romp;

/*
**-----
**
** Abstract:
** Init every Macro
**
** Parameters:
** None
**
** Returns:
** None
**-----
*/

```

```
void SystemInit( void )
{
    __DI( );                /* disable interrupt */

    _rcopy(&_S_romp, -1);

    ClrIORBit(DCHC0, 0x1);   /* disable dma0 - dma3 */
    ClrIORBit(DCHC1, 0x1);
    ClrIORBit(DCHC2, 0x1);
    ClrIORBit(DCHC3, 0x1);

    VSWC = 0x01;            /* mainclock (2MHz, 16.6MHz) lwait */

    PORT_Init( );           /* Port initiate */
    UART3_Init( );          /* UART3 initiate */
    CSIB5_Init( );          /* CSIB5 initiate */
    TMP0_Init();            /* Timer initiate */
    __EI( );                /* enable interrupt */
}
```

8.5 main.c

FILE ID: main.c

```

/*
*****
**
** This device driver was created by Applilet for the V850ES/JG2 and V850ES/JJ2
** 32-Bit Single-Chip Microcontrollers
**
** Copyright(C) NEC Electronics Corporation 2002-2006
** All rights reserved by NEC Electronics Corporation
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses incurred
** by customers or third parties arising from the use of this file.
**
** Filename : main.c
** Abstract : This file implements main function
** APIlib : V850ESJx2.lib V1.50 [23 Feb. 2006]
**
** Device: uPD70F3721
**
** Compiler: NEC/CA850
**
*****
*/
/*
** *****
** Include files
** *****
*/
#include "macrodriver.h"
#include "int.h"
#include "port.h"
#include "timer.h"
#include "serial.h"
#include "sdmemory.h"

#include "sw_vjj2.h" /* switch input */
#include "led_vjj2.h" /* LED display output */

/* prototypes */
int get_sector(int *sector);
void dump_byte(UCHAR *data, USHORT length);

/*
** *****
** MacroDefine
** *****
*/
#define BUF_SIZE 516

/*
** -----
**
** Abstract:

```

```

**      main function
**
**  Parameters:
**      None
**
**  Returns:
**      None
**
**-----
*/
void main( void )
{
char msg_nl[] = {"\r\n"};
const char msg1[] = {"\r\n\r\nSD Memory Demonstration program      (using polled I/O)
12/20/06\r\n"};
const char msg2[] = {"      demonstrating reading and writing of SD/MMC memory card via
SPI interface\r\n"};
const char msg3[] = {"\r\n  enter sector number to use (00-99) "};
const char msg4[] = {"      enter text to be written to SD memory on console\r\n"};
const char msg5[] = {"              (512 character per sector limit)\r\n"};
const char msg6[] = {"      press SW1 to write data to SD memory\r\n"};
const char msg7[] = {"      press SW2 to read data back from SD memory and display
it\r\n"};
const char msg_sdmr[] = {"\r\nSDmemory_Read sector %d\r\n"};
const char msg_sdmw[] = {"\r\nSDmemory_Write sector %d\r\n"};
const char msg_a[] = {"main - memory card init status 0x%02x\r\n"};
const char msg_b[] = {"main - memory card specific data request status 0x%02x\r\n"};
const char msg_c[] = {"main - memory card identification data request status
0x%02x\r\n"};
const char msg_d[] = {"main - sector %d read status 0x%02x\r\n"};
const char msg_e[] = {"main - sector %d write status 0x%02x\r\n"};
char msg_buf[120];

MD_STATUS status;
MD_STATUS mem_stat, mem_stat9, mem_stat10;
unsigned char sw_val;
UCHAR data[20];
int err,i;
int line;
int sector;
USHORT size, done, SD_status;
unsigned char buffer1[BUF_SIZE];
unsigned char buffer2[BUF_SIZE];

    WT_Start();      /* watch timer start up */
    sw_init();        /* initialize switch variables */
    led_init();        /* initialize LED display */
    TMP0_Start();     /* start timer for switch debouncing, led mux and millisecond counting
*/
    CSIB5_deselect_SPI(); /* bring all spi device select lines high */

    delay(250);        /* the setup of uart3 can put glitches on the line which looks like
start bit*/

                        /* allow some time for it to settle before output of text starts */

    uart3_tx_msg((char *)msg1);
    uart3_tx_msg((char *)msg2);

```

```

uart3_tx_msg((char *)msg4);
uart3_tx_msg((char *)msg5);
uart3_tx_msg((char *)msg6);
uart3_tx_msg((char *)msg7);

CSIB5_send_done = 0;

mem_stat = SDmemory_Init(); /* initialize SD memory access */
if(mem_stat != MD_OK) {
    sprintf(msg_buf, msg_a ,mem_stat); /* dbg SD memory init status 0x%02x */
    uart3_tx_msg(msg_buf); /* dbg */
    /* consider abort message and hang */
}
else
{
    delay(200);

    mem_stat = SDReadStatus(&SD_status);
    sprintf(msg_buf,"read status after init 0x%02x card status
0x%04x\r\n",mem_stat,SD_status);
    uart3_tx_msg(msg_buf);
    if(mem_stat == MD_MASTER_RCV_END)
        mem_stat = MD_OK;

}
if(mem_stat != MD_OK)
{
    uart3_tx_msg("initialization failed\r\n");
    while(1){;} // hang forever
}

/* after initialization has completed, enter an endless loop to ask for the */
/* sector number to read or write, then wait for data to be entered or for */
/* one of the switches to be pressed */
while (1) {
    UART3_User_Init(); /* reset buffer pointers */
    size = 0;
    line = 0;
    data[1] = 0;
    done = 0;
    buffer1[size++]=START_BLOCK; /* transmit data always starts with START_BLOCK */
    led_bcd(size); /* show size */

    /* output message to enter the sector number */
    uart3_tx_msg((char *)msg3);

    /* read the operator sector number input, if error, just request input again */
    err = get_sector(&sector);
    if(err || (sector > 99)) continue;
    uart3_tx_msg(msg_n1);

    while(done == 0)
    {
        /* check for any receive characters, display the character entered */
        if( Check_UART3_Receive(&data[0]) == 1)
        {
            buffer1[size] = data[0];

```

```

    if(data[0] == 0x08)
    {
        size--;
        /* move cursor back one, print a space and move back one again */
        /* should check that it is not the first character */
        line--;
    }
    else
    {
        size++;
        line++;
    }
    if(data[0] == '\r')
    {
        buffer1[size] = '\n';
        size++;
        uart3_tx_msg(&msg_nl[1]);
        line = 0;
    }
    /* check if too many characters entered, wrap around and restart*/
    /* allow for start token and crc bytes */
    if(size > BUF_SIZE-3)
    {
        size = 0;
        line = 0;
        buffer1[size]=START_BLOCK;
        size++;
        uart3_tx_msg(msg_nl);
    }
    led_bcd(size); /* display the current count of characters */
    uart3_tx_msg((char *)data);
    if(line >= 80)
    {
        line = 0;
        uart3_tx_msg(msg_nl);
    }
} /* end receive characters */

/* check for switch depression */
sw_val = sw_get();

if(sw_val != SW_LU_RU)
{
    if(sw_val == SW_LD_RU) /* 0x02 SW2 up, SW1 down 1 0 */
    {
        /* wait for switches to be released */
        while(sw_get() != SW_LU_RU){};

        sprintf(msg_buf, msg_sdmw, sector);
        uart3_tx_msg(msg_buf);
        /* clear the rest of the buffer */
        for(i=size; i<=512; i++)
            buffer1[i] = 0x21; /* arbitrary fill character */
        //buffer1[512] = 0xaa; dbg
        buffer1[513] = 0xff; /* call crc16 and put result here */
        buffer1[514] = 0xff;
        //buffer1[size] = 0x00; // dbg - terminate string
    }
}

```

```

        /* write size of data and buffer data to sd memory card */
        mem_stat = SDWriteSector(buffer1,sector); /* sectors are always 512
bytes */
        sprintf(msg_buf, msg_e, sector, mem_stat); // dbg SD memory card
specific data request status 0x%02x
        uart3_tx_msg(msg_buf); /* dbg */
        dump_byte(buffer1,515);
        done = 1;
    }
    if(sw_val == SW_LU_RD) /* 0x01 SW2 down, SW1 up      0          1      */
    {
        /* wait for switches to be released */
        while(sw_get() != SW_LU_RU) {};

        sprintf(msg_buf, msg_sdmr, sector);
        uart3_tx_msg(msg_buf);
        for(i=0; i<512; buffer2[i++]='z'); // fill buffer first, overwrite last
read
        /* read selected sector from sd memory card into buffer 2 */
        mem_stat = SDReadSector(buffer2, sector);
        sprintf(msg_buf, msg_d, sector, mem_stat); // dbg SD memory card
specific data request status 0x%02x
        uart3_tx_msg(msg_buf); /* dbg */

        /* display buffer2 */
        dump_byte(buffer2,514);
        done = 1;
    }
    if(sw_val == SW_LD_RD) /* 0x00 SW2 down, SW1 down    0          0      */
    {
        /* wait for switches to be released */
        while(sw_get() != SW_LU_RU) {};
    }
} // while(done)
}
}

/*****
/* Function:      get_sector()
/* Description:  monitor round robin buffer for operator input of
/*              sector number digits
/* Input:        sector - pointer to place decoded sector number at
/* Return:       1 on error, 0 on success
*****/
int get_sector(int *sector)
{
    int rxnum;
    UCHAR data[2];
    UCHAR done = 0;

    rxnum = 0;
    data[1] = 0;
    while(done == 0) /* done set at end of string */
    {
        /* start the interrupt driven receive process to get 1 character */
        /* this clears rcv3_done */
        if( Check_UART3_Receive(&data[0]) == 1)

```

```

    {
        uart3_tx_msg((char *)data);
        if(data[0] == '\r' || data[0] == '\n' || data[0] == '\0')
            done = 1;
        if(done == 0)
        {
            if ((data[0] < '0') || (data[0] > '9'))
                return (1);

            rxnum = rxnum * 10 + (data[0] & 0x0f);
        }
    }
    *sector = rxnum;
    return(0);
}

/*****
/* Function:      dump_byte()
/* Description:  dump memory in byte form, also show ascii equivalent
/*              dump to serial port
/* Input:        *data    - pointer to start dumping data from
/*              length  - number of bytes of data to display
/* Return:       none
*****/
void dump_byte(UCHAR *data, USHORT length)
{
    int i,j,k;
    char buff[8];
    char equiv[20];

    /* initialize the ascii equivalent */
    for(k=0; k<16; equiv[k++]=0x2a){;}
    equiv[16] = '\r';
    equiv[17] = '\n';
    equiv[18] = 0;

    for(i=0,j=0; i<length; i++, j=i%16)
    {
        if(j == 0)
        {
            uart3_tx_msg(equiv);
        }
        sprintf(buff,"%02x ", *data);
        uart3_tx_msg(buff);
        if(*data >= 0x20 && *data < 0x7f)
            equiv[j] = *data;
        else
            equiv[j] = '.';
        data++;
    }
    /* finish filling out the ascii equivalent */
    if(k=length%16)
        for(k=length%16; k<16; equiv[k++]=0x2a){;}

    uart3_tx_msg(equiv);
}

```


8.6 sdmemory.c

FILE_ID: sdmemory.c

```

/* sd memory.c */
#include "macrodriver.h"
#include "sdmemory.h"
#include "serial.h"
#include "timer.h"

MD_STATUS err_val(UCHAR response);
void dump_led_digit(void);
void build_cmd(char index, unsigned int arg);
void send_pad(char count);

unsigned char cmd_buf[10];
unsigned char rxbuf[256];
unsigned short r2_reply;
char buffer[518]; // error messages, temp place to read into
unsigned char pad[16];
union CMD
{
    unsigned int cmd_arg;
    unsigned char cmd_ch[4];
};
void csib5_test(int); // dbg

extern MD_STATUS csib5_snd_flag; /* dbg - serial.c */
extern UINT csib5_snd_count;    /* dbg - serial.c */
int retry; /* dbg - how many times did we try */

/*****
/* Function:      SDmemory_Init()
/* Description:  reset the sd/mmc card and put it into spi mode
/* Input:        none
/* Return:       MD_OK      - initialization performed sucessfully
/*              MD_REQ_TIMEOUT  -*/
/*              MD_INVALID_STATE -*/
/*              MD_NO_START    - CSIB5_SendData status, transmit error */
*****/
MD_STATUS SDmemory_Init(void)
{
    int i,j,done;
    MD_STATUS status, r_status;
    //char init_err[] = {"SD memory init status 0x%02x  %d\r\n"};
    unsigned char data;

    //delay(10);    this should not be necessary

    /* step 1 - reset the SD/MMC card and go into idle state */
    CSIB5_deselect_SPI(); /* deselect all spi devices */

    /* send clock pulses to allow card power up synchronization */
    /* to complete */
    send_pad(10);
    done = 1;

```

```

while(done != 0)
{
    delay(5);
    /* get ready for an R1 response message */
    R1_Initiate();

    CSIB5_select_SPI(SDMEM1); /* select the sd/mmc memory card */

    /* send pad,CMD0 to go into SPI mode */
    build_cmd(0,0);

    delay(10);

    R1_message = SDmemory_R_query(R1,20); /* it can take quite some time for card to
go to idle mode */
    CSIB5_deselect_SPI();
    if(R1_message == 0x01) /* 1 = idle, reset done */
        done = 0;
    else
        if(done++ > 100)
            return(MD_REQ_TIMEOUT);

} /* end while cmd0 */
#ifdef DEBUG
    sprintf(buffer,"\r\nStep 1 done (retry %d) now in idle mode\r\n", retry); // dbg
    uart3_tx_msg(buffer);
#endif

j = i = 400;
while(i != 0)
{
    /* send CMD1 until we get a 0 back, indicating card is done initializing */
    /* step 2 - do card initialization */
    R1_Initiate(); /* get ready for an R1 response message */
    CSIB5_select_SPI(SDMEM1); /* select the sd/mmc memory card */

    build_cmd(1,0); /* activate initialization process, CMD1 */
    i--;
    delay(1);
    R1_message = SDmemory_R_query(R1,20);
    CSIB5_deselect_SPI(); /* deselect all spi devices */

    if(R1_message == 0x00) /* ready, no longer in idle */
    {
        #ifdef DEBUG
            sprintf(buffer,"Step 2 done (retry %d) now in SPI mode\r\n",j-i);
            uart3_tx_msg((char *)buffer);
        #endif
        return(MD_OK);
    }
    delay(5); /* set n millisecond delay */
}
return (MD_INVALID_STATE);
}

/*****/
/* Function:      SDmemory_CMD_R16() */
/* Description: send cmd to request read of CSD or CID register, these*/

```

```

/*          commands return 16 bytes of data and CRC          */
/* Input:    index  - the command number to be sent (9 or 10)  */
/*          data   - pointer to 18 byte result buffer          */
/* Return:    MD_MASTER_RCV_END - command reply received ok    */
/*          MD_REQ_TIMEOUT   - no command reply received      */
/*****
MD_STATUS SDmemory_CMD_R16(char index, UCHAR *data)
{
    MD_STATUS status;
    UCHAR Rl_message;
    UCHAR DT_message;
    int i;

    Rl_Initiate(); /* set up to receive an Rl response */

    CSIB5_select_SPI(SDMEM1); /* select the sd memory card */

    build_cmd(index,0); /* build command 9 or 10, arg = 0 and start sending it */

    for(i=0; i<200; i++)
        buffer[i] = 0; // small delay

    Rl_message = SDmemory_R_query(Rl,100);
    if(Rl_message)
    {
        #ifdef DEBUG
        sprintf(buffer,"R_query %d 0x%02x error\r\n",index,Rl_message);
        uart3_tx_msg(buffer);
        #endif

        CSIB5_deselect_SPI();
        return(err_val(Rl_message));
    }

    for(i=0; i<200; i++)
        buffer[i] = 0xff; // small delay

    /* look for the data token */
    DT_message = SDmemory_DT_query(200);
    if(DT_message == 0x80)
    {
        #ifdef DEBUG
        uart3_tx_msg("DT query timeout\r\n");
        #endif
        CSIB5_deselect_SPI();

        return(MD_REQ_TIMEOUT);
    }

    /* initialize for receipt of 16 bytes + 2 bytes CRC */
    // status = CSIB5_ReceiveData(data,18);

    /* now send 18 dummy bytes to clock in the good data */
    CSIB5_SendData((UCHAR *)buffer,18,data);

    /* send NEC pad bytes */
    send_pad(NEC);

```

```

    CSIB5_deselect_SPI(); /* this also clears scope trigger */

    return (MD_MASTER_RCV_END);
}

/*****
/* Function:      R1_Initiate()
/* Description: prepare for interrupt driven response message after
/*               a command has been sent.
/* Input:         none
/* Output:        R1_recieve = 0
/*               R1_message = 0xff default to invalid reply
/*               rcv_msg_done = 0 clear interrupt flag
/* Return:        none
*****/
void R1_Initiate(void)
{
    CB5STR = 0; // clear status errors
    R1_received = 0;
    R1_message = 0xff; /* an invalid response */
    //CSIB5_rcv_done = 0;
}

/*****
/* Function:      R2_Initiate()
/* Description: prepare for interrupt driven response message after
/*               a command has been sent.
/* Input:         none
/* Output:        R2_recieve = 0
/*               R2_message = 0xffff default to invalid reply
Imaginary Buffer Line
/* Return:        none
*****/
void R2_Initiate(void)
{
    CB5STR = 0; // clear status errors
    R1_received = 0; /* r2 response is an r1 with additional byte */
    R2_received = 0;
    R2_message = 0xffff;
    //CSIB5_rcv_done = 0;
}

/*****
/* Function:      SDmemory_R_query()
/* Description: query for the specified response type, repeat the
/*               query for the number of times requested
/* Input:         response - query type
/*               repeat - number of times to retry the query before
/*               failing
/* Return:        first received character
*****/
UCHAR SDmemory_R_query(char response, short repeat)
{
    short i = repeat;
    int count;
    UCHAR txbuf[2] = {0xff,0xff};

    rxbuf[0] = 0xff; /* default to no reply */

```

```

retry = 0;          /* dbg to see how long it takes to get response back */
while(rxbuf[0] == 0xff)
{
    switch(response) {
        case(R1):
        case(R1b):
        {
            /* get ready to read a one byte R1 response message */
            R1_Initiate();
            count = 1;
            break;
        }
        case(R2):
        {
            /* get ready to read a one byte R1 response message */
            /* followed by 1 byte of status data */
            R2_Initiate();
            count = 2;
            break;
        }
        case(R3):
        {
            /* get ready to read a one byte R1 response message */
            /* followed by 4 bytes of OCR data */
            R1_Initiate();
            count = 1;
        }
    } // end switch

    CSIB5_SendData(txbuf, count, rxbuf);

    //R1_message = rxbuf[count-1];
    //Byte = CB5RXL; /* read recieved data byte again */
    //sprintf(buffer,"Byte = %2x\r\n",Byte);
    //uart3_tx_msg(buffer);

    retry++; // dbg
    i--;
    if(i == 0) return(0x80);
}
R1_received = 1;
return (rxbuf[0]);
} /* SDmemory_R_query */

/*****
/* Function:      SDmemory_DT_query()
/* Description:   query the card until it gets a Data Token (0xfe)
/*               value. Repeat the query up to count times before
/*               failing.
/* Input:         count - max number of times to repeat query
/* Return:        one byte of read info (0xfe if found, 0x80 if not)
*****/
UCHAR SDmemory_DT_query(short count)
{
    short i = count;
    unsigned char Byte = 0xff;

```

```

UCHAR txbuf[1] = {0xff};

retry = 0; // dbg
while(Byte != 0xfe)
{
    /* get ready to read a one byte R1 response message */
    R1_Initiate();
    CSIB5_SendData(txbuf, 1, &Byte);
    retry++; //dbg
    i--;
    if(i == 0) return(0x80);
}
return (Byte);
} /* SDmemory_DT_query */

/*****
/* Function:      SDmemory_DR_query()
/* Description:  query the card until it gets a Data Response Token
/* Input:        count - max number of times to retry query
/* Return:       - returns one byte of read info ( 0x80 if not)
/*              0x05 - data accepted
/*              0x0b - data rejected CRC
/*              0x0d - data rejected error
*****/
UCHAR SDmemory_DR_query(short count)
{
    short i = count;
    short j;
    unsigned char Byte = 0xff;
    UCHAR txbuf[1] = {0xff};

    retry = 0; // dbg
    while(Byte == 0xff)
    {
        for(j=0; j<0xff; j++)
        {
            txbuf[0] = j; // delay
        }
        retry++; // dbg
        /* get ready to read a one byte response message */
        R1_Initiate();
        CSIB5_SendData(txbuf, 1, &Byte);
        i--;
        if(i == 0) return(0x80);
    }

    return (Byte&0x1f);
} /* SDmemory_DR_query */

/*****
/* Function:      SDReadSector()
/* Description:  read all of the requested sector into the buffer
/* Input:        pBuffer - pointer to start of receive buffer to use
/*              Sector - sector number to read
/* Return:       MD_STATUS */
*****/
MD_STATUS SDReadSector(UCHAR *pBuffer, int Sector)
{

```

```

MD_STATUS status;
UCHAR DT_message;
int i, block_address;

csib5_test(1); // dbg - turn on data check
block_address = Sector << 9;
#ifdef DEBUG
sprintf(buffer, "SDReadSector from addr 0x%08x (sector %d) into buffer 0x%08x\r\n",
        block_address, Sector, pBuffer);
uart3_tx_msg(buffer);
#endif
Rl_Initiate(); /* set up to receive an Rl response */

CSIB5_select_SPI(SDMEM1); /* select the sd memory card */

build_cmd(17, block_address);

Rl_message = SDmemory_R_query(Rl, 400);
if(Rl_message)
{
    #ifdef DEBUG
    sprintf(buffer, "query 17 0x%02x error (retry %d) \r\n", Rl_message, retry);
    uart3_tx_msg(buffer);
    #endif
    CSIB5_deselect_SPI();
    return(err_val(Rl_message));
}
for(i=0; i<514; i++)
    buffer[i] = 0xff; // small delay (is this needed & could it be put into query?)

DT_message = SDmemory_DT_query(200); /* locate the data token */
if(DT_message == 0x80)
{
    #ifdef DEBUG
    uart3_tx_msg("DT query timeout\r\n");
    #endif
    CSIB5_deselect_SPI();
    return(MD_REQ_TIMEOUT);
}

/* now send 514 dummy bytes to clock in one sector of data + CRC16 */
status = CSIB5_SendData((UCHAR *)buffer, 514, pBuffer);
/* send NEC pad bytes */
send_pad(NEC);
CSIB5_deselect_SPI();
if(status != MD_OK)
{
    return(status);
}

return (MD_MASTER_RCV_END);
} /* SDReadSector */

/*****
/* Function:      SDWriteSector()
/* Description:   write the sector using the data in the buffer
/* Input:        pBuffer - pointer to start of receive buffer to use
/*              Sector - sector number to read
*****/

```

```

/* Return:      MD_MASTER_SEND_END - successssful write          */
/*              MD_REQ_TIMEOUT - */
/*****
MD_STATUS SDWriteSector(UCHAR *pBuffer, int Sector)
{
    MD_STATUS status;
    UCHAR DR_message;
    int i, block_address;

csib5_test(0);
    block_address = Sector << 9;
    Rl_Initiate(); /* set up to receive an Rl response */

    CSIB5_select_SPI(SDMEM1); /* select the sd memory card */
//pad ??
    build_cmd(24, block_address);

    Rl_message = SDmemory_R_query(Rl,NCR);
    if(Rl_message)
    {
        #ifdef DEBUG
        sprintf(buffer,"Rl query cmd24 0x%02x error\r\n",Rl_message);
        uart3_tx_msg(buffer);
        #endif
        CSIB5_deselect_SPI();
        return(err_val(Rl_message));
    }

    /* send NWR pad bytes */
    send_pad(NWR);

    /* now send 515 data bytes (token, data, crc) */
    status = CSIB5_SendData(pBuffer, 515, (unsigned char *)buffer);
    if(status != MD_OK)
    {
        CSIB5_deselect_SPI();
        return(status);
    }
    delay(5);

    /* check for data response token */
    DR_message = SDmemory_DR_query(20);
#ifdef DEBUG
    sprintf(buffer,"DR_message 0x%02x\r\n",DR_message); // dbg
    uart3_tx_msg(buffer); // dbg
#endif
    if(DR_message == 0x80)
    {
        #ifdef DEBUG
        uart3_tx_msg("DR query timeout\r\n");
        #endif
        CSIB5_deselect_SPI();
        return(MD_REQ_TIMEOUT);
    }
#ifdef DEBUG
    if(DR_message == 0x05) uart3_tx_msg("data accepted\r\n");
    if(DR_message == 0x0b) uart3_tx_msg("data rejected CRC\r\n");

```



```

    if(DR_message == 0x0d) uart3_tx_msg("data rejected error\r\n");
    #endif
    /* send NEC pad bytes */
    send_pad(NEC);

    CSIB5_deselect_SPI(); /* this also clears scope trigger */

    return (MD_MASTER_SEND_END);
} /* SDWriteSector */

/*****
/* Function:      SDReadStatus()
/* Description:   read the status register
/* Input:         pointer to place return status value
/* Return:        MD_MASTER_RCV_END - status request &reply successful
*****/
MD_STATUS SDReadStatus(USHORT *pStatus)
{
    MD_STATUS status;
    UCHAR DT_message;
    unsigned short temp;

    R2_Initiate(); /* set up to receive an R2 response */

    CSIB5_select_SPI(SDMEM1); /* select the sd memory card */

    build_cmd(13, 0);

    R1_message = SDmemory_R_query(R2,20);
    if(R1_message)
    {
        #ifdef DEBUG
        sprintf(buffer,"ReadStatus R2_query 13 0x%02x 0x%02x 0x%02x %s error\r\n",
            R1_message,rxbuf[0], rxbuf[1], err_text(R1_message));
        uart3_tx_msg(buffer);
        #endif
        CSIB5_deselect_SPI();
        return(err_val(R1_message));
    }
    temp = (rxbuf[0] <<8) | rxbuf[1];
    *pStatus = temp;

    /* send NEC pad bytes */
    send_pad(NEC);

    CSIB5_deselect_SPI(); /* this also clears scope trigger */

    return (MD_MASTER_RCV_END);
}

/*****
/* Function:      build_cmd()
/* Description:   send NCS padding characters, then build the command
/*               in the command buffer, supply the crc7 checksum,
/*               and the send the command message
/* Input:         index - the command number to be sent
/*               arg   - the command argument
/* Return:        none
*****/

```

```

/*****/
void build_cmd(char index, unsigned int arg)
{
    int i;
    union CMD c;

    /* send NCS pad bytes here */
    send_pad(NCS);

    /* build the command in cmd_buf */
    cmd_buf[0] = 0x40 | (index & 0x3f);
    c.cmd_arg = arg;
    cmd_buf[1] = c.cmd_ch[0];
    cmd_buf[2] = c.cmd_ch[1];
    cmd_buf[3] = c.cmd_ch[2];
    cmd_buf[4] = c.cmd_ch[3];
    cmd_buf[5] = do_crc7(cmd_buf, 5);
    cmd_buf[6] = 0xff;

    CSIB5_SendData(cmd_buf, 7, rxbuf);
}

/*****/
/* Function:      send_pad()                                     */
/* Description:   send count characters of value 0xff out SPI port */
/* Input:         count - number of padding characters (0xff) to be */
/*                sent (max 15 or size of pad array)                */
/* Return:        none                                           */
/*****/
void send_pad(char count)
{
    int i;

    for(i=0; i<16; i++)
        pad[i]=0xff;

    CSIB5_SendData(pad, count, rxbuf);
}

#if 0
/*****/
/* Function:      SDmemory_CMD16()                               */
/* Description:   set the block length for all following transactions */
/*                Supported only if Partial block RD/WR operations are */
/*                allowed in CSD                                       */
/* Input:         block_len - new default block length >= 512         */
/*                expressed as 2**block_len                             */
/* Return:        MD_ARGERROR - input argument > 512                 */
/*                MD_OK        - proper R1 response received          */
/*****/
MD_STATUS SDmemory_CMD16(unsigned int block_len)
{
    if(block_len < 512)
        return(MD_ARGERROR);
    /* convert to power of 2 */
    R1_Initiate(); /* set up to receive an R1 response */
    CSIB5_select_SPI(SDMEM1); /* select the sd memory card */
    build_cmd(16, 0x00000009);
    R1_message = SDmemory_R_query(R1, 20);
}

```

```

CSIB5_deselect_SPI();
if(Rl_message)
{
    sprintf(buffer,"ReadStatus Rl_query cmd16 0x%02x %s error\r\n",
            Rl_message, err_text(Rl_message));
    uart3_tx_msg(buffer);
    return(err_val(Rl_message));
}
return(MD_OK); /* Rl response */
}

/*****
/* Function:      dump_csd()
/* Description:   display selected info from CSD register
/* Input:         data - pointer to CSD register data
/* Return:        none
*****/
void dump_csd(unsigned char *data)
{
    int    i,j;
    unsigned int read_bl_len;
    char read_bl_partial, write_bl_partial;
    unsigned int wC_SIZE;
    unsigned int wC_SIZE_MULT;
    unsigned int wDummy;
    int dTotalSectors = 0;
    char *time_unit[] = {"1ns","10ns","100ns","1us","10us","100us","1ms","10ms"};
    char *time_value[] = {"reserve","1.0","1.2","1.3","1.5","2.0","2.5","3.0","3.5","4.0","4.5","5.0","5.5","6.0","7.0","8.0"};

    uart3_tx_msg("\r\nCSD Register info\r\n");
    i = data[1] & 0x03;
    j = (data[1] >> 2) & 0x0f;
    sprintf(buffer,"TAAC (0x%02x) time value %s units of %s",data[1],time_value[j],time_unit[i]);
    uart3_tx_msg(buffer);
    sprintf(buffer,"NSAC (0x%02x) clock cycles*100\r\n",data[2]);
    uart3_tx_msg(buffer);

    /* Get the READ_BL_LEN */
    read_bl_len = (1 << (data[5] & 0x0F));
    read_bl_partial = data[6] >> 7;
    write_bl_partial = (data[13] >> 5) & 1;
    /* Get the C_SIZE */
    wC_SIZE = (data[6] & 0x03);
    wC_SIZE = wC_SIZE << 10;

    wDummy = data[7];
    wDummy = wDummy << 2;
    wC_SIZE |= wDummy;

    wDummy = (data[8] & 0xC0);
    wDummy = wDummy >> 6;
    wC_SIZE |= wDummy;

    /* Get the wC_SIZE_MULT */

```

```

wC_SIZE_MULT = (data[9] & 0x03);
wC_SIZE_MULT |= wC_SIZE_MULT << 1;
wDummy       = (data[10] & 0x80);
wDummy       = wDummy >> 7;
wC_SIZE_MULT |= wDummy;
wC_SIZE_MULT = (1 << (wC_SIZE_MULT+2));

dTotalSectors = wC_SIZE+1;
dTotalSectors *= wC_SIZE_MULT;

sprintf(buffer,"TotalSectors  %d   device size %d   sector size %d
bytes\r\n",dTotalSectors,wC_SIZE,read_bl_len);
uart3_tx_msg(buffer);
if(read_bl_partial)
    uart3_tx_msg("read block partial allowed\r\n");
else
    uart3_tx_msg("read block partial not allowed\r\n");
if(write_bl_partial)
    uart3_tx_msg("write block partial allowed\r\n");
else
    uart3_tx_msg("write block partial not allowed\r\n");
}

/*****
/* Function:      dump_cid()
/* Description:   display selected info from the cid register
/* Input:         data - pointer to CID register data
/* Return:        none
*****/
void dump_cid(unsigned char *data)
{
    unsigned short stemp;
    unsigned int itemp;

    /* manufacturer and OEM/Application id ID */
    stemp = data[1]<<8;
    stemp |= data[2];
    sprintf(buffer,"Manufacturer ID 0x%02x      OEM/Application ID
0x%04x\r\n",data[0],stemp);
    uart3_tx_msg(buffer);

    /* product name */
    sprintf(buffer,"Product Name |%c%c%c%c%c|      Product Revision   %d%d\r\n",
        data[3],data[4],data[5],data[6],data[7],data[8],data[9]>>4,data[9]&0x0f);
    uart3_tx_msg(buffer);

    itemp = data[10];
    itemp = (itemp<<8) | data[11];
    itemp = (itemp<<8) | data[12];
    itemp = (itemp<<8) | data[13];
    sprintf(buffer,"Serial Number 0x%08x  Manufacturing Date Code %d/%d\r\n",
        itemp, data[14]>>4, (data[14]&0x0f)+1997);
    uart3_tx_msg(buffer);
}
#endif

/*****
/* Function:      do_crc7()
*****/

```

```

/* Description: since CRC7 is only used for the first message, return */
/*              the fixed value.                                     */
/* Input:       data - pointer to data                             */
/*              size - length of data                             */
/* Return:      checksum7                                          */
/*****
unsigned char do_crc7(unsigned char *data, unsigned short size)
{
    return(0x95);
}

/*****
/* Function:     err_val()                                         */
/* Description:  convert reply to MD_STATUS error value           */
/* Input:       response - R1 message response                     */
/* Return:      MD_STATUS - equivalent code for R1 error           */
/*****
MD_STATUS err_val(UCHAR response)
{
    MD_STATUS val[] = { MD_INVALID_STATE,
                        MD_ERASE_ERR, MD_ILLEGAL_CMD, MD_CKSUM_ERR,
                        MD_ERASE_SEQ, MD_ADDRESS_ERR, MD_ARGERROR};

    UCHAR mask;
    int i;

    mask = 1;
    for(i=0; i<8; i++)
    {
        if(response & mask)
            return(val[i]);
        mask = mask << 1;
    }
    return(0);
}

/*****
/* Function:     err_text()                                         */
/* Description:  convert R1 reply to text pointer value           */
/* Input:       response - R1 message response                     */
/* Return:      pointer to error message                           */
/*****
char * err_text(UCHAR response)
{
    char *val[] = { "idle state",
                    "erase reset", "illegal command", "com crc",
                    "erase sequence", "address", "parameter"};

    UCHAR mask;
    int i;

    mask = 1;
    for(i=0; i<8; i++)
    {
        if(response & mask)
            return(val[i]);
        mask = mask << 1;
    }
    return("unknown");
}

```

8.7 serial.c**FILE ID: serial.c**

```

/*
*****
**
** This device driver was created by Applilet for the V850ES/JG2, V850ES/JJ2
** 32-Bit Single-Chip Microcontrollers
**
** Copyright(C) NEC Electronics Corporation 2002-2006
** All rights reserved by NEC Electronics Corporation
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename :      serial.c
** Abstract :      This file implements a device driver for the SERIAL module
** APIlib : V850ESJx2.lib V1.50 [23 Feb. 2006]
**
** Device :uPD70F3721
**
** Compiler :      NEC/CA850
**
*****/

#include "macrodriver.h"
#include "serial.h"
#include "led_vjj2.h" //dbg only

#pragma interrupt INTUA3R MD_INTUA3R
#pragma interrupt INTUA3T MD_INTUA3T

USHORT  uart3_snd_count;

volatile UCHAR *uart3_snd_pbuf;
USHORT  uart3_rcv_count;
UCHAR   *uart3_rcv_pbuf;
volatile UCHAR send3_done;

USHORT  csib5_snd_size;
UINT    csib5_snd_count;
UCHAR   *csib5_snd_pbuf;
MD_STATUS csib5_snd_flag;

MD_STATUS csib5_rcv_flag;
USHORT  csib5_rcv_size;
UCHAR   *csib5_rcv_pbuf;
UINT    csib5_rcv_count;

int test; //dbg

/*-----
** Abstract:
** This function initializes UARTA3.
**

```

```

** Parameters:      None
**
** Returns:None
**-----*/
void UART3_Init( void )
{
    ClrIORBit(UA3CTL0, 0x80);          /* stop uarta3 before making any changes */
    ClrIORBit(UA3CTL0, 0x40);          /* disable transmit */
    ClrIORBit(UA3CTL0, 0x20);          /* disable receive */

    SetIORBit(UA3TIC, 0x40);           /* disable tx interrupt service */
    SetIORBit(UA3RIC, 0x40);           /* disable rx interrupt service */
    ClrIORBit(UA3TIC, 0x80);           /* clear interrupt request issued */
    ClrIORBit(UA3RIC, 0x80);           /* clear interrupt request issued */

    SetIORBit(PMC8, 0x03);             /* setting port mode for uart */
    SetIORBit(UA3CTL0, 0x10);          /* UA3DIR=1, LSB-first */
    ClrIORBit(UA3CTL0, 0xc);           /* UA3PS1=UA3PS0=0, no parity */
    SetIORBit(UA3CTL0, 0x2);           /* UA3CL=1, 8 bits data frame */
    ClrIORBit(UA3CTL0, 0x1);           /* UA3SL=0, 1 stop bit */
    SetIORBit(UA3OPT0, 0x14);           /* UA3SLS2=UA3SLS0=1, 13 bits SBF data length */
    ClrIORBit(UA3OPT0, 0x8);           /* UA3SLS1=0, 13 bits SBF data length */
    ClrIORBit(UA3OPT0, 0x2);           /* UA3TDL=0, transfer data level: normal */
    ClrIORBit(UA3OPT0, 0x1);           /* UA3RDL=0, receive data level: normal */

    /* baud rate 9600 */
    UA3CTL1 = UART3_BAUDRATE_M0;      /* 0x03 baudrate setting fclk=fx/8 fx=20Mhz */
    UA3CTL2 = UART3_BAUDRATE_K0;      /* 0x82 fclk/130 */

    SetIORBit(UA3RIC, 0x7);            /* receive end interrupt priority is lowest */
    SetIORBit(UA3TIC, 0x7);            /* transmit end interrupt priority is lowest */
    ClrIORBit(UA3RIC, 0x40);           /* enable reception interrupt servicing */
    ClrIORBit(UA3TIC, 0x40);           /* enable transmission interrupt servicing */

    SetIORBit(UA3CTL0, 0x80);          /* enable UARTA3 */
    SetIORBit(UA3CTL0, 0x60);          /* enable receive and transmit operation */

    UART3_User_Init();                /* user initialization */

    return;
}

/*-----*/
** Abstract:
** This function is responsible for start of UART3 data transfer.
**
** Parameters:
** UCHAR *txbuf : Address of transfer buffer.
** USHORT txnum : The number of data to transmit(frame number).
**
** Returns:None
**-----*/
void UART3_SendData(UCHAR *txbuf, USHORT txnum)
{
    SetIORBit(UA3CTL0, 0x40);          /* TX start */

    uart3_snd_pbuf = txbuf;

```

```

    uart3_snd_count = txnum;
    send3_done = 0;

    UA3TX = *uart3_snd_pbuf++;
    uart3_snd_count--;

    return ;
}

/*-----
** Abstract:
** This function is responsible for start of UARTA3 data receiving.
**
** Parameters:
** rxbuf : Address of receive buffer.
** rxnum : The size of receive buffer.
**
** Returns:None
**-----*/
void UART3_ReceiveData(UCHAR *rxbuf, USHORT rxnum)
{
    SetIORBit(UA3CTL0, 0x20);          /* RX start */

    uart3_rcv_pbuf = rxbuf;
    uart3_rcv_count = rxnum;

    return ;
}

/*-----
** Abstract:
** This function is the UART3 transmit interrupt handler for INTST3.
**
** Parameters:    None
**
** Returns:None
**-----*/
__interrupt void MD_INTUA3T( void )
{
    if( uart3_snd_count ){
        UA3TX = *uart3_snd_pbuf++; /* send the next character, increment the pointer */
        uart3_snd_count--;          /* decrement number of characters left to send */
    }
    else{
        /* send finish, user own coding */
        send3_done = 1;
    }
}

/*-----
** Abstract:
** This function is the UART3 receive interrupt handler for INTSR3.
**
** Parameters:    None
**
** Returns:None
**-----*/

```



```

__multi_interrupt void MD_INTUA3R( void )
{
    __EI();

    if( UA3STR & 0x07 ){          /* status check */
        return;
    }

    UART3_Receive(UA3RX);
}

/*-----
** Abstract:
** This function initializes CSIB5. It is called by systeminit.
**
** Parameters:      None
**
** Returns:None
**-----*/
void CSIB5_Init( void )
{
    CB5CTL0 = 0;                  /* stop CSIB5 before making changes */

    SetIORBit(CB5TIC, 0x40);      /* stop transmit interrupt */
    SetIORBit(CB5RIC, 0x40);      /* stop receive interrupt */
    ClrIORBit(CB5TIC, 0x80);      /* clear interrupt req issued */
    ClrIORBit(CB5RIC, 0x80);      /* clear interrupt req issued */

    SetIORBit( PMC6, 0x0040);      /* PMC66 = SIB5 input */
    SetIORBit( PMC6, 0x0080);      /* PMC67 = SOB5 output */
    SetIORBit( PMC6, 0x0100);      /* PMC68 = SCKB5 I/O */

    ClrIORBit(CB5CTL0, 0x10);      /* MSB first */
    ClrIORBit(CB5CTL0, 0x02);      /* single transfer mode */

    //CB5CTL1 = 0x03;              /* type 1, fxx/16 = 1.25 MHz*/
    //CB5CTL1 = 0x04;              /* type 1, fxx/32 = 625KHz */
    CB5CTL1 = 0x05;              /* type 1, fxx/64 = 312.5KHz*/
    CB5CTL2 = 0x00;              /* data length - 8bit */
    //SetIORBit(CB5RIC, 0x05);      /* reception interrupt priority setting level 5 */
    //ClrIORBit(CB5RIC, 0x40);      /* enable interrupt servicing */
    SetIORBit(CB5CTL0, 0x40);      /* enable send interrupt */
    SetIORBit(CB5CTL0, 0x20);      /* enable receive */
    SetIORBit(CB5CTL0, 0x81);      /* enable operation, communication start trigger valid */

    return;
}

/*-----
** Abstract:
** This function is responsible for initiating transfer of data out CSIB5.
** Since every byte sent is a byte received, it also receives data.
**
** Parameters:
** UCHAR* txbuf : Address of transmit buffer.
** USHORT txnum : The number of data bytes to transmit(frame number).
** UCHAR* rxbuf : Address of receive buffer.

```

```

**
** Returns:
** MD_OK
** MD_NO_START
**-----*/
MD_STATUS CSIB5_SendData(UCHAR* txbuf, USHORT txnum, UCHAR* rxbuf)
{
char cb5_status;
int i;
char check_char; // dbg

/* init parameters */
csib5_snd_size = txnum;
csib5_snd_pbuf = txbuf;
csib5_snd_count = 0;
csib5_rcv_pbuf = rxbuf;

CB5STR = 0; // clear overflow

while(csib5_snd_count < csib5_snd_size)
{
    CB5TXL = *csib5_snd_pbuf; // send a byte of data
    csib5_snd_pbuf++;
    csib5_snd_count++;
    cb5_status = CB5STR;

    if(cb5_status & 0x80) // check if transmit has started
    {
        while((CB5STR & 0x80) == 0x80){;} // wait for tx to stop sending
        CB5STR = 0;
        *csib5_rcv_pbuf = CB5RX; // read receive register and save
    }
    if(test) { // dbg
        if(csib5_snd_count == 10) check_char = *csib5_rcv_pbuf; //dbg
        if((csib5_snd_count>300 && csib5_snd_count<508) && (*csib5_rcv_pbuf != check_char)) // dbg
            /*+++++*/
            ClrIORBit(P3,SPI_CS4); /****** dbg- set scope trigger on dif *****/
    }

    csib5_rcv_pbuf++;
}
else
{
    return(MD_NO_START);
}
}
return(MD_OK);
}
// select when to check for bad data
void csib5_test(int value) // dbg
{
    test = value;
}

```

8.8 port.c**FILE ID: port.c**

```

/*
*****
**
** This device driver was created by Applilet for the V850ES/JG2, V850ES/JJ2
** 32-Bit Single-Chip Microcontrollers
**
** Copyright(C) NEC Electronics Corporation 2002-2006
** All rights reserved by NEC Electronics Corporation
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : port.c
** Abstract : This file implements a device driver for the port module
** APilib : V850ESJx2.lib V1.50 [23 Feb. 2006]
**
** Device : uPD70F3721
**
** Compiler : NEC/CA850
**
*****
*/

/*
**=====
** Include files
**=====
*/
#include "macrodriver.h"
#include "port.h"

/*
**=====
** Constants
**=====
*/

/*
**-----
** Abstract:
** Initialize the I/O module
**
** Parameters:
** None
**
** Returns:
** None
**-----
*/
void PORT_Init( void )
{
    /* initialize the port registers */
    P0 = PORT_P0;      // SD card IRQ on P04

```

```

P1 = PORT_P1;          // zigbee
P3 = PORT_P3;          // USB uart0, I2C, zigbee, chip selects
/* P4 = PORT_P4;        used by minicube2 */
P5 = PORT_P5;          // = 0 user switch input
P6 = PORT_P6;          // = 0x1f LED digit select, SPI to SD memory card, zigbee
P7L = PORT_P7L;
P7H = PORT_P7H;
P8 = PORT_P8;          /* = 0x00 used as uart3 */
P9 = PORT_P9;
PCD = PORT_PCD;
// PCM = PORT_PCM;
PCS = PORT_PCS;
PCT = PORT_PCT;
PDH = PORT_PDH;
PDL = PORT_PDL;

/* initialize the function registers */
PF0 = PORT_PF0;
PF3 = PORT_PF3;
// PF4 = PORT_PF4;
PF5 = PORT_PF5;
PF6 = PORT_PF6;
// PF8 = PORT_PF8; /* used as uart3 */
PF9 = PORT_PF9;

/* initialize the mode registers */
PM0 = PORT_PM0;
PM1 = PORT_PM1;
PM3 = PORT_PM3;
// PM4 = PORT_PM4;
PM5 = PORT_PM5;
PM6 = PORT_PM6;
PM7L = PORT_PM7L;
PM7H = PORT_PM7H;
// PM8 = PORT_PM8;
PM9 = PORT_PM9;
PMCD = PORT_PMCD;
// PMCM = PORT_PMCM; // = 0xff all set to input
PMCS = PORT_PMCS;
PMCT = PORT_PMCT;
PMDH = PORT_PMDH;
PMDL = PORT_PMDL;

/* initialize the mode control registers */
PMC0 &= ~PORT_PMC0;
PMC3 &= ~PORT_PMC3;
// PMC4 &= ~PORT_PMC4;
PMC5 &= ~PORT_PMC5;
PMC6 &= ~PORT_PMC6;
// PMC8 = 0x03;
PMC9 &= ~PORT_PMC9;
// PMCCM = PORT_PMCCM; // = 0x01 minicube uses /WAIT input
PMCCS &= ~PORT_PMCCS;
PMCCCT &= ~PORT_PMCCCT;
PMCDH &= ~PORT_PMCDH;
PMCDL &= ~PORT_PMCDL;
return;
}

```

8.9 led_vjj2.c

FILE ID led_vjj2.c

```

/* led_vjj2.c - routines for LED display */
/* for AF-V850ES-JJ2 CPU evaluation board */
/* Version: 1.1 11-10-2006 */

/* using LITEON LTC-4627JR 4 digit 7 segment LED display */
/* this is a multiplexed display */
/* P60 = Digit 1 (left) source control */
/* P61 = Digit 2 source control */
/* P62 = Digit 3 source control */
/* P63 = Digit 4 (right) source control */
/* P64 = colon (L1 & L2) and L3 control */
/*
/* A0 = segment A & L1 common sink
/* A1 = segment B & L2 common sink
/* A2 = segment C & L# common sink
/* A3 = segment D common sink
/* A4 = segment E common sink
/* A5 = segment F common sink
/* A6 = segment G common sink
/* A7 = decimal point common sink

/* need pragma declaration to access SFR's in C */
#pragma ioreg

#include "led_vjj2.h"

/* table of bit patterns for seven-segment digits */
static unsigned char dig_tab[] = {
    LED_PAT_0, /* 0 */
    LED_PAT_1, /* 1 */
    LED_PAT_2, /* 2 */
    LED_PAT_3, /* 3 */
    LED_PAT_4, /* 4 */
    LED_PAT_5, /* 5 */
    LED_PAT_6, /* 6 */
    LED_PAT_7, /* 7 */
    LED_PAT_8, /* 8 */
    LED_PAT_9, /* 9 */
    LED_PAT_A, /* A */
    LED_PAT_B, /* B */
    LED_PAT_C, /* C */
    LED_PAT_D, /* D */
    LED_PAT_E, /* E */
    LED_PAT_F /* F */
};

/* raw seven segment LED data to be displayed by multiplex */
/* display routine */
volatile unsigned char led_digit[5];
static unsigned char port_select[5] = {0x1e, 0x1d, 0x1b, 0x17, 0x0f};
volatile int digit;

char buf[20]; // debug
void dump_led_digit(void) // debug

```

```

{
int i;
    uart3_tx_msg("\r\nled_digit ");
    for(i=0; i<5; i++)
    {
        sprintf(buf,"0x%02x ",led_digit[i]);
        uart3_tx_msg(buf);
    }
    uart3_tx_msg("\r\n");
}

/*****
/* Function:      led_init()
/* Description:   set up ports for display of LED digits, initialize
/*                display settings to all off
/* Input:         none
/* Output:        Port settings, led_digit array initialized
/* Return:        none
*****/
void led_init(void)
{
    /* turn all LED segments, decimal points etc. off */
    led_digit[0] = led_digit[1] = led_digit[2] = led_digit[3] = led_digit[4] = LED_OFF;

    /* ports initialized in Port_Init() by Applilet */
}

/*****
/* Function:      led_out_digit1()
/* Description:   set display value for LED digit 1 (MSB)
/*                save the decimal point setting
/* Input:         val - raw bit setting for digit1 seven segment display*/
/* Output:        led_digit array changed
/* Return:        none
*****/
void led_out_digit1(unsigned char val)
{
    led_digit[0] &= 0x80; /* keep the decimal point */
    led_digit[0] |= val & 0x7f;
}

/*****
/* Function:      led_out_digit2()
/* Description:   set display value for LED digit 2
/*                save the decimal point setting
/* Input:         val - raw bit setting for digit2 seven segment display*/
/* Output:        led_digit array changed
/* Return:        none
*****/
void led_out_digit2(unsigned char val)
{
    led_digit[1] &= 0x80; /* keep the decimal point */
    led_digit[1] |= val & 0x7f;
}

/*****
/* Function:      led_out_digit3()
/* Description:   set display value for LED digit 3
*****/

```

```

/*          save the decimal point setting          */
/* Input:    val - raw bit setting for digit3 seven segment display*/
/* Output:    led_digit array changed                */
/* Return:    none                                   */
/*****/
void led_out_digit3(unsigned char val)
{
    led_digit[2] &= 0x80; /* keep the decimal point */
    led_digit[2] |= val & 0x7f;
}

/*****/
/* Function:    led_out_digit4()                    */
/* Description: set display value for LED digit 4 (LSB) */
/*          save the decimal point setting          */
/* Input:    val - raw bit setting for digit4 seven segment display*/
/* Output:    led_digit array changed                */
/* Return:    none                                   */
/*****/
void led_out_digit4(unsigned char val)
{
    led_digit[3] &= 0x80; /* keep the decimal point */
    led_digit[3] |= val & 0x7f;
}

/*****/
/* Function:    led_dp_digit1()                    */
/* Description: turn on or off digit1 DP LED        */
/*          save the current digit setting          */
/* Input:    on = LED_ON, turn LED decimal point on */
/*          = LED_OFF, turn LED decimal point off   */
/* Output:    modifies led_digit[0]                 */
/* Return:    none                                   */
/*****/
void led_dp_digit1(unsigned char on)
{
    if (on == LED_ON)
        led_digit[0] = led_digit[0] | 0x80; /* set bit 7 high to turn off */
    else
        led_digit[0] = led_digit[0] & 0x7f; /* set bit 7 low to turn on */
}

/*****/
/* Function:    led_dp_digit2()                    */
/* Description: turn on or off digit2 DP LED        */
/*          save the current digit setting          */
/* Input:    on = LED_ON, turn LED decimal point on */
/*          = LED_OFF, turn LED decimal point off   */
/* Output:    modifies led_digit[1]                 */
/* Return:    none                                   */
/*****/
void led_dp_digit2(unsigned char on)
{
    if (on == LED_ON)
        led_digit[1] = led_digit[1] | 0x80; /* set bit 7 high to turn off */
    else
        led_digit[1] = led_digit[1] & 0x7f; /* set bit 7 low to turn on */
}

```

```

}

/*****
/* Function:      led_dp_digit3()
/* Description:   turn on or off digit3 DP LED
/*               save the current digit setting
/* Input:         on = LED_ON, turn LED decimal point on
/*               = LED_OFF, turn LED decimal point off
/* Output:        modifies led_digit[2]
/* Return:        none
*****/
void led_dp_digit3(unsigned char on)
{
    if (on == LED_ON)
        led_digit[2] = led_digit[2] | 0x80; /* set bit 7 high to turn off */
    else
        led_digit[2] = led_digit[2] & 0x7f; /* set bit 7 low to turn on */
}

/*****
/* Function:      led_dp_digit4()
/* Description:   turn on or off digit4 DP LED
/*               save the current digit setting
/* Input:         on = LED_ON, turn LED decimal point on
/*               = LED_OFF, turn LED decimal point off
/* Output:        modifies led_digit[3]
/* Return:        none
*****/
void led_dp_digit4(unsigned char on)
{
    if (on == LED_ON)
        led_digit[3] = led_digit[3] | 0x80; /* set bit 7 high to turn off */
    else
        led_digit[3] = led_digit[3] & 0x7f; /* set bit 7 low to turn on */
}

/*****
/* Function:      led_L1()
/* Description:   turn on or off L1 LED, this is the top led of the
/*               center colon.
/*               save the current digit setting
/* Input:         on = LED_ON, turn L1 LED on
/*               = LED_OFF, turn L1 LED off
/* Output:        modifies led_digit[4]
/* Return:        none
*****/
void led_L1(unsigned char on) /* turn on or off L1 LED */
{
    if(on == LED_ON)
        led_digit[4] &= 0xFE; /* turn it on */
    else
        led_digit[4] |= 1; /* turn it off */
}

/*****
/* Function:      led_L2()
/* Description:   turn on or off L2 LED, this is the bottom led of the
/*               center colon.
*****/

```



```

/*          save the current digit setting          */
/* Input:    on = LED_ON, turn L2 LED on            */
/*          = LED_OFF, turn L2 LED off              */
/* Output:    modifies led_digit[4]                 */
/* Return:    none                                  */
/*****
void led_L2(unsigned char on)          /* turn on or off L2 LED */
{
    if(on == LED_ON)
        led_digit[4] &= 0xFD; /* turn it on */
    else
        led_digit[4] |= 2;
}

/*****
/* Function:    led_L3()                                */
/* Description: turn on or off L3 LED, this is the top led between */
/*          digits 3 and 4                                */
/*          save the current digit setting                */
/* Input:    on = LED_ON, turn L3 LED on                    */
/*          = LED_OFF, turn L3 LED off                      */
/* Output:    modifies led_digit[4]                        */
/* Return:    none                                          */
/*****
void led_L3(unsigned char on)          /* turn on or off L3 LED */
{
    if(on == LED_ON)
        led_digit[4] &= 0xFB; /* turn it on */
    else
        led_digit[4] |= 4;
}

/*****
/* Function:    led_colon()                                */
/* Description: turn on or off both LEDs of the colon        */
/*          save the current digit setting                */
/* Input:    on = LED_ON, turn L1 and L2 LED on                    */
/*          = LED_OFF, turn L1 and L2 LED off                      */
/* Output:    modifies led_digit[4]                        */
/* Return:    none                                          */
/*****
void led_colon(unsigned char on)          /* turn on or off L1 L2 colon */
{
    led_L1(on);
    led_L2(on);
}

/*****
/* Function:    led_num_digit1()                                */
/* Description: display number in digit1 (MSB) LED display    */
/* Input:    num - number in range 0x00 - 0x0f displayed on led */
/* Output:    led_digit[0] modified                            */
/* Return:    none                                          */
/*****
void led_num_digit1(unsigned char num)
{
    if (num > 0x0F) {

```

```

        led_out_digit1(LED_PAT_BLANK);
        return;
    }
    led_out_digit1(dig_tab[num]);
}

/*****
/* Function:      led_num_digit2()
/* Description: display number in digit2 LED display
/* Input:         num - number in range 0x00 - 0x0f displayed on led
/* Output:        led_digit[1] modified
/* Return:        none
*****/
void led_num_digit2(unsigned char num)
{
    if (num > 0x0F) {
        led_out_digit2(LED_PAT_BLANK);
        return;
    }
    led_out_digit2(dig_tab[num]);
}

/*****
/* Function:      led_num_digit3()
/* Description: display number in digit3 LED display
/* Input:         num - number in range 0x00 - 0x0f displayed on led
/* Output:        led_digit[2] modified
/* Return:        none
*****/
void led_num_digit3(unsigned char num)
{
    if (num > 0x0F) {
        led_out_digit3(LED_PAT_BLANK);
        return;
    }
    led_out_digit3(dig_tab[num]);
}

/*****
/* Function:      led_num_digit4()
/* Description: display number in digit4 (LSB) LED display

/* Input:         num - number in range 0x00 - 0x0f displayed on led
/* Output:        led_digit[3] modified
/* Return:        none
*****/
void led_num_digit4(unsigned char num)
{
    if (num > 0x0F) {
        led_out_digit4(LED_PAT_BLANK);
        return;
    }
    led_out_digit4(dig_tab[num]);
}

/*****
/* Function:      led_hex()
/* Description: display number as four hex digits
*****/

```

```

/* Input:      num - number to display                                */
/* Output:     led_digit[0..3] modified                               */
/* Return:     none                                                  */
/*****
void led_hex(unsigned short num)
{
    led_out_digit4(dig_tab[num & 0x000F]);
    led_out_digit3(dig_tab[(num >> 4) & 0x000F]);
    led_out_digit2(dig_tab[(num >> 8) & 0x000F]);
    led_out_digit1(dig_tab[(num >>12) & 0x000F]);
}

/*****
/* Function:    led_dig_bcd()                                         */
/* Description: display four digits of BCD coded number              */
/* Input:       bcdnum - number to display in BCD                    */
/*              0 - 9         displayed as one decimal digit, left 3 blank */
/*              10 - 99        displayed as two decimal digits         */
/*              100 - 999      displayed as three decimal digits        */
/*              1000 - 9999    displayed as four decimal digits         */
/*              > 9999         displayed as "----"                    */
/* Output:      led_digit[0..3] modified                               */
/* Return:      none                                                  */
/*****
void led_bcd(unsigned short bcdnum)
{
    unsigned char flag, tens, hundreds, thousands;

    flag = 0;

    if (bcdnum > 9999) {
        led_out_digit4(LED_PAT_DASH); /* display digits as dashes */
        led_out_digit3(LED_PAT_DASH);
        led_out_digit2(LED_PAT_DASH);
        led_out_digit1(LED_PAT_DASH);
        return;
    }

    if (bcdnum > 999) {
        thousands = 0;
        do{
            bcdnum -= 1000;
            thousands++;
        } while (bcdnum > 999);
        led_out_digit1(dig_tab[thousands]);
        flag = 1;
    }
    else
        led_out_digit1(LED_PAT_BLANK);

    if (bcdnum > 99) {
        hundreds = 0;
        do{
            bcdnum -= 100;
            hundreds++;
        } while (bcdnum > 99);
        led_out_digit2(dig_tab[hundreds]);
    }
}

```

```

        flag = 1;
    }
    else
        if(flag)
            led_out_digit2(dig_tab[0]);
        else
            led_out_digit2(LED_PAT_BLANK);

/* 10 <= bcdnum <= 99 */
if (bcdnum > 9) {
    tens = 0;
    do {
        /* calculate ten's place and remainder */
        bcdnum -= 10; /* by multiple subtractions of 10 */
        tens++;      /* while counting up the tens digit */
    } while (bcdnum > 9);
    led_out_digit3(dig_tab[tens]);
} else
    if(flag)
        led_out_digit3(dig_tab[0]);
    else
        led_out_digit3(LED_PAT_BLANK);

led_out_digit4(dig_tab[bcdnum]); /* just display LSB LED digit */
}

/*****
/* Function:      led_mux_drive()
/* Description:  update next digit of LED display
/* Input:        none
/* Output:       led_digit[0..3] modified
/* Return:       none
*****/
void led_mux_drive(void)
{
    digit++;
    if(digit > 4)
        digit = 0;
    // turn select off for all led's in port 6
    P6 = DIGIT_OFF; // set all select bits

    // select new segment data for new digit
    P9L = led_digit[digit];

    // turn select on for current new digit (0 - on)
    P6 = port_select[digit];
}

```

8.10 sw_vjj2.c**FILE ID: sw_vjj2.c**

```

/* sw_vjj2.c - routines for switch input */
/* for AF-EV850 - JJ2 CPU evaluation */
/* Version: 1.0 11-11-2006 */

/* P51 = input for left switch (SW2) */
/* P50 = input for right switch (SW3) */

/* need pragma declaration to access SFR's in C */
#pragma ioreg

#include "sw_vjj2.h"

/* local variables for switch handling */
static unsigned char sw_last; /* last debounced switch value */
static unsigned char sw_new; /* new value being debounced */
static unsigned char sw_deb_value; /* value of debounce counter */
static unsigned char sw_deb_count; /* debounce counter */

/*****/
/* Function: sw_init() */
/* Description: set up ports for switch input */
/* Input: none */
/* Return: none */
/*****/
void sw_init(void)
{
    /* initialization done in Port_Init() by Applilet */
    /* set static variables */
    sw_last = SW_LU_RU; /* default is right up, left up (no switch pressed) */
    sw_set_debounce(SW_DEF_DEB_COUNT); /* set default debounce counter value */
}

/* void sw_set_debounce(unsigned char count) */
/* set the debounce counter value */
void sw_set_debounce(unsigned char count)
{
    sw_deb_value = count; /* set new debounce counter value */
    sw_deb_count = count; /* set counter to max */
}

/* unsigned char sw_get(void) */
/* return debounced switch input */

unsigned char sw_get(void)
{
    return sw_last;
}

/* unsigned char sw_chk(void) */
/* return input from switches, undebounded */
unsigned char sw_chk(void)
{
    return (P5 & 0x03);
}

```

```
}

/* void sw_isr( void ) */
/* this routine called by periodic timer interrupt to poll and debounce switches */
/* after a new value has been seen steadily for sw_deb_value times, sw_last is updated */
void sw_isr( void )
{
    unsigned char val;

    val = sw_chk();          /* get current value */
    /* if value is the same as before, no change; reset debounce and return */
    if (val == sw_last) {
        sw_deb_count = sw_deb_value; /* reset debounce counter to max */
        return;
    }

    /* val != sw_last, there is a new input */
    /* if it's not the same as the previous new one, */
    /* set the NEW new one, reset the debounce counter and return */
    if (val != sw_new) {
        sw_new = val;
        sw_deb_count = sw_deb_value;
        return;
    }

    /* val != sw_last, val == sw_new */
    /* count down the debounce counter */
    sw_deb_count--;

    /* if we have counted down to zero, we have seen the same sw_new */
    /* for debounce count times, it is now the debounced switch value */
    if (sw_deb_count == 0) {
        sw_last = val;
        sw_deb_count = sw_deb_value;
        return;
    }

    /* if still debouncing, just return */
    return;
}
```

8.11 timer.c**FILE ID: timer.c**

```

/*
*****
**
** This device driver was created by Applilet for the V850ES/JG2, V850ES/JJ2
** 32-Bit Single-Chip Microcontrollers
**
** Copyright(C) NEC Electronics Corporation 2002-2006
** All rights reserved by NEC Electronics Corporation
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : timer.c
** Abstract : This file implements a device driver for the timer module
** APILib : V850ESJx2.lib V1.50 [23 Feb. 2006]
**
** Device : uPD70F3721
**
** Compiler : NEC/CA850
**
*****
*/

/*
*****
** Include files
*****

*/
#include "macrodriver.h"
#include "timer.h"
/*
*****
** Constants
*****

*/

/*
**-----
**
** Abstract:
** This function initializes TMP0.
**
** Parameters:
** None
**
** Returns:
** None
**
**-----
*/
void TMP0_Init( void )
{

```

```

/* Stop counting */
ClrIORBit(TP0CTL0, 0x80);
/* Mask interrupt */
SetIORBit(TP0CCIC0, 0x40);
SetIORBit(TP0CCIC1, 0x40);
SetIORBit(TP0OVIC, 0x40);
/* Clear interrupt request flag */
ClrIORBit(TP0CCIC0, 0x80);
ClrIORBit(TP0CCIC1, 0x80);
ClrIORBit(TP0OVIC, 0x80);

ClrIORBit(TP0CTL1, 0x20);      /* disable external event count input */
TP0CTL0 |= TM_TMP0_CLOCK;     /* internal count clock */
/* Interval timer mode */
ClrIORBit(TP0CTL1, 0x07);
TP0CCR0 = TM_TMP0_INTERVALVALUE; /* set interval value to compare against */
TP0CCR1 = 0xffff;
/* Interrupt INTTP0CC0 */
SetIORBit(TP0CCIC0, 0x07);
TMP0_User_Init( );
}

/*
**-----
**
** Abstract:
** This function starts TMP0 counter.
**
** Parameters:
** None
**
** Returns:
** None
**
**-----
*/
void TMP0_Start( void )
{
    ClrIORBit(TP0CCIC0, 0x40);      /* enable interrupt INTTP0CC0 */
    SetIORBit(TP0CTL0, 0x80);      /* start counting */
    return;
}

/*
**-----
**
** Abstract:
** This function stops the TMP0 counter and clear the count register.
**
** Parameters:
** None
**
** Returns:
** None
**
**-----
*/

```



```

#if 0
void TMP0_Stop( void )
{
    ClrIORBit(TP0CTL0,0x80);      /* stop counting */
    /* Mask interrupt */
    SetIORBit(TP0CCIC0,0x40);
    /* Clear interrupt request flag */
    ClrIORBit(TP0CCIC0,0x80);
    return;
}

/*
**-----
**
** Abstract:
** This function changes TMP0 condition.
**
** Parameters:
** USHORT*:    array_reg
** USHORT:     array_num
**
** Returns:
** MD_OK
** MD_ARGERROR
**-----
*/
MD_STATUS TMP0_ChangeTimerCondition( USHORT* array_reg, USHORT array_num )
{
    if((array_num < 1) || (array_num > 2)){
        return MD_ARGERROR;
    }
    if( array_num >= 1 ){
        TP0CCR0 = *array_reg;
    }
    if( array_num >= 2){
        TP0CCR1 = *(array_reg + 1);
    }
    return MD_OK;
}
#endif

```

8.12 timer_user.c**FILE ID: timer_user.c**

```

/*
*****
**
** This device driver was created by Applilet for the V850ES/JG2, V850ES/JJ2
** 32-Bit Single-Chip Microcontrollers
**
** Copyright(C) NEC Electronics Corporation 2002-2006
** All rights reserved by NEC Electronics Corporation
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename :      timer_user.c
** Abstract :      This file implements a device driver for the timer module
** APIlib :       V850ESJx2.lib V1.50 [23 Feb. 2006]
**
** Device :       uPD70F3721
**
** Compiler :     NEC/CA850
**
*****
*/
/*****
** Include files
*****/
#include "macrodriver.h"
#include "timer.h"

/* add include file for led, switches */
#include "led_vjj2.h"
#include "sw_vjj2.h"

#pragma interrupt INTTP0CC0 MD_INTTP0CC0

/*****
** variables
*****/
/* counter for millisecond timer */
volatile unsigned int milliseconds;
/* counter for led update */
unsigned int led_update;

/*-----
**
** Abstract:
**   TMP0 initializing.
**
** Parameters:
**   None
**
** Returns:
**   None
**

```

```

**-----*/
void TMP0_User_Init( void )
{
    led_update = 0;
    milliseconds = 0;
}

/*-----
**
** Abstract:
** This function is TMP0 INTTP0CC0 interrupt service routine.
**
** Parameters:
** None
**
** Returns:
** None
**
**-----*/
__multi_interrupt void MD_INTTP0CC0( void )
{
    __EI();

    /* debounce switch status when timer interrupt occurs */
    sw_isr();

    /* multiplex LED update - move this code to led_vjj2 */
    led_update++;
    if(led_update > 1)
    {
        led_mux_drive();
        led_update = 0;
    }
    /* count down millisecond timer if it is non zero */
    if (milliseconds > 0)
        milliseconds--;
}

/*****
/* Function:      SetMsecTimer()                               */
/* Description:   set the millisecond count down timer         */
/* Input:         time - number of clock ticks to be counted down */
/* Return:        none                                         */
*****/
void SetMsecTimer(int time)
{
    milliseconds = time;
}

/*****
/* Function:      CheckMsecTimer()                             */
/* Description:   check the millisecond timer count down value */
/* Input:         none                                         */
/* Return:        MD_FALSE - time has not expired             */
/*               MD_TRUE  - timer has counted down to zero    */
*****/
BOOL CheckMsecTimer(void)
{

```

```
    if (milliseconds > 0)
        return MD_FALSE; // return false if not done counting down
    return MD_TRUE;
}

/*****
/* Function:    delay()
/* Description: set count down timer to value given, then wait for it*/
/*              to be counted down to zero before returning
/* Input:       count - count down value to start from
/* Return:      none
*****/
void delay(int count)
{
    SetMsecTimer(count);
    while(CheckMsecTimer() == MD_FALSE){;} //hang until count is zero
}
```

8.13 system.inc**FILE ID: system.inc**

```

--/*
--*****
--**
--** This device driver was created by Applilet for the V850ES/JG2 and V850ES/JJ2
--** 32-Bit Single-Chip Microcontrollers
--**
--** Copyright(C) NEC Electronics Corporation 2002-2006
--** All rights reserved by NEC Electronics Corporation
--**
--** This program should be used on your own responsibility.
--** NEC Electronics Corporation assumes no responsibility for any losses incurred
--** by customers or third parties arising from the use of this file.
--**
--** Filename : system.inc
--** Abstract : This file includes the definitions of the SYSTEM module
--** APIlib: v850esJx2.lib V1.50 [23 Feb. 2006]
--**
-- Device: uPD70F3721
--
--
-- Compiler: NEC/CA850
--
--*****
--*/
.set      CG_Mainosc, 0x5
.set      CG_SECURITY0,      0xff
.set      CG_SECURITY1,      0xff
.set      CG_SECURITY2,      0xff
.set      CG_SECURITY3,      0xff
.set      CG_SECURITY4,      0xff
.set      CG_SECURITY5,      0xff
.set      CG_SECURITY6,      0xff
.set      CG_SECURITY7,      0xff
.set      CG_SECURITY8,      0xff
.set      CG_SECURITY9,      0xff

```

8.14 macrodriver.h

FILE ID: macrodriver.h

```

/*
*****
**
** This device driver was created by Applilet for the V850ES/JG2 and V850ES/JJ2
** 32-Bit Single-Chip Microcontrollers
**
** Copyright(C) NEC Electronics Corporation 2002-2006
** All rights reserved by NEC Electronics Corporation
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses incurred
** by customers or third parties arising from the use of this file.
**
** Filename : macrodriver.h
** Abstract : This is the general header file
** APIlib: v850esJx2.lib V1.50 [23 Feb. 2006]
**
** Device: uPD70F3717
**
** Compiler: NEC/CA850
**
*****
*/

#ifndef _MDSTATUS_
#define _MDSTATUS_
#pragma ioreg /*enable use the register directly in ca850 compiler*/

/* data type defintion */
typedef unsigned int    UINT;
typedef unsigned short  USHORT;
typedef unsigned char   UCHAR;
typedef unsigned char   BOOL;

#define DEBUG            1

#define MD_ON            1
#define MD_OFF           0

#define MD_TRUE          1
#define MD_FALSE         0

#define MD_STATUS        unsigned short
#define MD_STATUSBASE    0x0
/*status list definition*/
#define MD_OK            MD_STATUSBASE+0x0 /* register setting OK*/
#define MD_RESET         MD_STATUSBASE+0x1 /* reset input*/
#define MD_SENDCOMPLETE  MD_STATUSBASE+0x2 /* send data complete*/
#define MD_ADDRESSMATCH  MD_STATUSBASE+0x3 /* IIC slave address match*/
#define MD_OVF           MD_STATUSBASE+0x4 /* timer count overflow*/
#define MD_DMA_END       MD_STATUSBASE+0x5 /* DMA transfer end*/
#define MD_DMA_CONTINUE  MD_STATUSBASE+0x6 /* DMA transfer continue*/
#define MD_SPT           MD_STATUSBASE+0x7 /* IIC stop*/

```

```

#define MD_NACK MD_STATUSBASE+0x08 /* IIC no ACK*/
#define MD_SLAVE_SEND_END MD_STATUSBASE+0x09 /* IIC slave send end*/
#define MD_SLAVE_RCV_END MD_STATUSBASE+0x01 /* IIC slave receive end*/
#define MD_MASTER_SEND_END MD_STATUSBASE+0x11 /* IIC master send end*/
#define MD_MASTER_RCV_END MD_STATUSBASE+0x12 /* IIC/SPI master receive end*/
#define MD_ERASE_END MD_STATUSBASE+0x13 /* erase block complete */

/*error list definition*/
#define MD_ERRORBASE 0x80
#define MD_ERROR MD_ERRORBASE+0x00 /*error*/
#define MD_RESOURCEERROR MD_ERRORBASE+0x01 /*no resource available*/
#define MD_PARITYERROR MD_ERRORBASE+0x02 /*UARTn parity error n=0,1,2*/
#define MD_OVERRUNERROR MD_ERRORBASE+0x03 /*UARTn overrun error n=0,1,2*/
#define MD_FRAMEERROR MD_ERRORBASE+0x04 /*UARTn frame error n=0,1,2*/
#define MD_TIMINGERROR MD_ERRORBASE+0x06 /*Error timing operation error*/
#define MD_SETPROHIBITED MD_ERRORBASE+0x07 /*setting prohibited*/
#define MD_DATAEXISTS MD_ERRORBASE+0x09 /*Data to be transferred next exists in
TXBn register*/
#define MD_NO_DEVICE MD_ERRORBASE+0x11
#define MD_REQ_TIMEOUT MD_ERRORBASE+0x12 /* request timed out */
#define MD_INVALID_STATE MD_ERRORBASE+0x13
#define MD_NO_START MD_ERRORBASE+0x14 /* csib5 communication stopped */
#define MD_ERASE_ERR MD_ERRORBASE+0x15
#define MD_ILLEGAL_CMD MD_ERRORBASE+0x16
#define MD_CKSUM_ERR MD_ERRORBASE+0x17
#define MD_ERASE_SEQ MD_ERRORBASE+0x18
#define MD_ADDRESS_ERR MD_ERRORBASE+0x19
#define MD_ARGERROR MD_ERRORBASE+0x1a /*Error agrument/parameter input error*/

/* macro function definiton */
/* main clock and subclock as clock source*/
enum ClockMode { MainClock, SubClock };
/*timer input channel*/
enum TMChannel { TMChannel0, TMChannel1, TMChannel2, TMChannel3 };
enum INTLevel{ Highest, Level1, Level2, Level3, Level4, Level5, Level6, Lowest };
enum TrigEdge { None, RisingEdge, FallingEdge, BothEdge };
/* clear IO register bit and set IO register bit */
#define ClrIORBit(Reg,ClrBitMap) Reg&=~ClrBitMap
#define SetIORBit(Reg,SetBitMap) Reg|=SetBitMap

#define SYSTEMCLOCK 20000000
#define SUBCLOCK 32768
#define MAINCLOCK 5000000
#define FRCLOCK 200000
#define FxInuse 1

#endif

```

8.15 sdmemory.h**FILE ID: sdmemory.h**

```

/* sd memory .h */
/* header for M-V850ES-KJ1 CPU board for SPI to SD memory communication */

#ifndef _SD_MEMORY_H
#define _SD_MEMORY_H
#include "macrodriver.h"

typedef struct CID_TYPE {
    unsigned char mid;    /* manufacturer ID */
    unsigned short oid;   /* OEM/Application ID */
    char pnm[5];          /* product name */
    unsigned char prv;    /* product revision */
    unsigned int psn;      /* product serial number */
    unsigned short mdt;    /* 12 bit manufacturing date */
    unsigned char crc;    /* CRC7 checksum*/
};

#define START_BLOCK      0xFE
#define STOP_TRAN        0xFD

#define ACCEPT           2
#define CRC_ERR          5
#define WR_ERR           6

#define OUT_OF_RANGE     0x08
#define CARD_ECC_FAIL    0x04
#define CC_ERROR         0x02
#define ERROR            0x01

#define NO_DESELECT      0
#define DESELECT         1

#define NCS              2    /* number of pad bytes before command */
#define NCR              64
#define NAC              2
#define NEC              3
#define NCX              1
#define NWR              3

#define R1               1
#define R1b              2
#define R2               3
#define R3               4

#define SECTOR_SIZE      512

typedef unsigned char STATUS_REG[65];
typedef unsigned char OCR_REG[5];

MD_STATUS SDmemory_Write(UCHAR *buffer, USHORT size);
MD_STATUS SDmemory_Read(UCHAR *buffer, USHORT size);
MD_STATUS SDMemory_Send_CMD(UCHAR cmd, UCHAR *reply);

```



```
MD_STATUS SDmemory_Read_CID(UCHAR *buffer);
MD_STATUS SDmemory_Read_CSD(UCHAR *buffer);
MD_STATUS SDmemory_Read_SCR(UCHAR *buffer);
MD_STATUS SDmemory_Read_OCR(UCHAR *buffer);

MD_STATUS SDmemory_CMD0(void);
MD_STATUS SDmemory_CMD_R16(char index, UCHAR *buffer); // cmd 9 and 10
MD_STATUS SDmemory_CMD13(unsigned short *reg);
MD_STATUS SDmemory_CMD16(unsigned int block_len);
MD_STATUS SDmemory_CMD24(unsigned int data_addr, unsigned int block_len);

void R1_Initiate(void);
void R1b_Initiate(void);
void R2_Initiate(void);
void R3_Initiate(void);

MD_STATUS R1_Response(UCHAR flag);
MD_STATUS R1b_Response(void);
MD_STATUS R2_Response(unsigned short *reg);
MD_STATUS R3_Response(OCR_REG *reg);

unsigned char SDmemory_R_query(char response, short max_retry);
unsigned char SDmemory_DR_query(short max_retry);
unsigned char SDmemory_DT_query(short max_retry);
unsigned char do_crc7(unsigned char *data, unsigned short size);
unsigned short do_crc16(unsigned char *data, unsigned short size);

MD_STATUS SDmemory_Init(void);
MD_STATUS SDReadSector(unsigned char *data, int sector);
MD_STATUS SDWriteSector(unsigned char *data, int sector);
void dump_csd(unsigned char *data);
#endif /* _SD_MEMORY_H */
```

8.16 serial.h

FILE ID: serial.h

```

/*
*****
**
** This device driver was created by Applilet for the V850ES/JG2, V850ES/JJ2
** 32-Bit Single-Chip Microcontrollers
**
** Copyright(C) NEC Electronics Corporation 2002-2006
** All rights reserved by NEC Electronics Corporation
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : serial.h
** Abstract : This file
implements a device driver for the SERIAL module
** APILib :
V850ESJx2.1

ib V1.50 [23 Feb. 2006]
**
** Device : uPD70F3721
**
** Compiler : NEC/CA850
**
*****
*/
#ifndef _MDSERIAL_
#define _MDSERIAL_

#define IIC_RECEIVEBUFSIZE 32

#define
UART3_BAUDR
ATE_M0 0x03
#define
UART3_BAUDR
ATE_K0 0x82

#define SDMEM1 1
#define SDMEM1T 3 // using zigbee chip select for a scope trigger
#define SPI_CS4 0x0010
#define ZIGBEE 2
#define SPI_CS5 0x0020

void UART3_Init( void );
void UART3_SendData( UCHAR*, USHORT);
void UART3_ReceiveData( UCHAR* , USHORT );
void UART3_User_Init( void );
void CALL_UART3_Receive( UCHAR );
char Check_UART3_Receive( UCHAR *);
void uart3_tx_msg(char *);

/* CSIB5 API functions */
void CSIB5_Init( void );

```

```
MD_STATUS CSIB5_SendData( UCHAR* , USHORT, UCHAR* );
MD_STATUS CSIB5_ReceiveData( UCHAR* , USHORT );
void CSIB5_User_Init( void );
void CALL_CSIB5_Send( void );
void CALL_CSIB5_Receive( void );
void CALL_CSIB5_Error( void );
void CSIB5_select_SPI(int device);
void CSIB5_deselect_SPI(void);
```

```
enum TransferMode { Send, Receive };
extern volatile int R1_received;
extern volatile int R2_received;
extern volatile int R3_received;
extern volatile char R1_message;
extern volatile short R2_message;
extern volatile int R3_message;
extern volatile int CSIB5_rcv_done;
extern volatile int CSIB5_send_done;
extern MD_STATUS csib5_rcv_flag;
```

```
#endif
```

```
/*_MDSERIAL
```

```
_*/
```

8.17 port.h**FILE ID: port.h**

```

/*
*****
**
** This device driver was created by Applilet for the V850ES/JG2, V850ES/JJ2
** 32-Bit Single-Chip Microcontrollers
**
** Copyright(C) NEC Electronics Corporation 2002-2006
** All rights reserved by NEC Electronics Corporation
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : port.h
** Abstract : This file implements a device driver for the port module
** APIlib : V850ESJx2.lib V1.50 [23 Feb. 2006]
**
** Device : uPD70F3721
**
** Compiler : NEC/CA850
**
*****
*/

#ifndef _MDPORT_
#define _MDPORT_
/*
** *****
** MacroDefine
** *****
**/
#define PORT_PMC0 0x0
#define PORT_PM0 0xff
#define PORT_PF0 0x0
#define PORT_P0 0x0
#define PORT_PM1 0xff
#define PORT_P1 0x0

#define PORT_PMC3 0x0038 // all are I/O ports
#define PORT_PM3 0xffcf // P34 and P35 are output, P33 input
#define PORT_P3 0x0030 // initial state
#define PORT_PF3 0x0000 // Normal CMOS output

#define PORT_PMC4 0x0
#define PORT_PM4 0xff
#define PORT_P4 0x0
#define PORT_PF4 0x0
#define PORT_PMC5 0x3
#define PORT_PM5 0xff
#define PORT_P5 0x0
#define PORT_PF5 0x0

#define PORT_PMC6 0x001f // select SPI controller on pins 6-8

```

```
                                // select I/O on all others (1 = output)
#define PORT_PM6      0xffe0    // 0 = output mode
#define PORT_P6       0x001f    // 1 deselects led digit
#define PORT_PF6      0x0000    // should have pull ups on these lines

#define PORT_PM7L     0xff
#define PORT_P7L      0x0
#define PORT_PM7H     0xff
#define PORT_P7H      0x0

#define PORT_PMC8      0x00      // UART3
#define PORT_PM8       0xff
#define PORT_PF8       0x00
#define PORT_P8        0x00
#define PORT_PMC9      0xff
#define PORT_PM9       0xff00
#define PORT_P9        0xff
#define PORT_PF9       0x0
#define PORT_PMCD      0xff
#define PORT_PCD       0x0
#define PORT_PCMCM     0xff
#define PORT_PCM       0x00
#define PORT_PMCCM     0x00
#define PORT_PMCS      0xff
#define PORT_PCS       0x0
#define PORT_PMCCS     0x0
#define PORT_PMCT      0xff
#define PORT_PCT       0x0
#define PORT_PMCCCT    0x0
#define PORT_PMDH      0xff
#define PORT_PDH       0x0
#define PORT_PMCDH     0x0
#define PORT_PMDL      0xffff
#define PORT_PDL       0x0
#define PORT_PMCDL     0x0

void PORT_Init(void);
#endif
```

8.18 led_vjj2.h**FILE ID: led_vjj2.h**

```

/* led_vjj2.h */
/* header for AF-V850 - JJ2 CPU evaluation board LED digit display */
/* Version 1.0      11-10-2006 */

#ifndef _LED_VKJ1_H
#define _LED_VKJ1_H

/*****
/* Define definitions */
/*****

/* LED Patterns for decimal and hex digits, characters */
/* for individual bits, ---A--- */
/* 0=on 1=off | */
/* bit 0 = segment A F B */
/* bit 1 = segment B | */
/* bit 2 = segment C ---G--- */
/* bit 3 = segment D | */
/* bit 4 = segment E E C */
/* bit 5 = segment F | */
/* bit 6 = segment G ---D--- DP */
/* bit 7 = decimal point */

#define LED_PAT_0 0xC0
#define LED_PAT_1 0xF9
#define LED_PAT_2 0xA4
#define LED_PAT_3 0xB0
#define LED_PAT_4 0x99
#define LED_PAT_5 0x92
#define LED_PAT_6 0x82
#define LED_PAT_7 0xF8
#define LED_PAT_8 0x80
#define LED_PAT_9 0x98
#define LED_PAT_A 0x88
#define LED_PAT_B 0x83
#define LED_PAT_C 0xC6
#define LED_PAT_D 0xA1
#define LED_PAT_E 0x86
#define LED_PAT_F 0x8E
#define LED_PAT_BLANK 0xFF
#define LED_PAT_DP 0x7F
#define LED_PAT_DASH 0xBF
#define LED_PAT_ULINE 0xF7
#define LED_PAT_OLINE 0xFE
#define LED_PAT_EQUAL 0xB7

#define LED_ON 0x00
#define LED_OFF 0xFF
#define DIGIT_OFF 0x1F

/*****
/* Export functions */

```

```

/*****/

void led_init(void); /* initialize ports for LED output */
void led_out_digit1(unsigned char val); /* output value to digit1 LED */
void led_out_digit2(unsigned char val); /* output value to digit2 LED */
void led_out_digit3(unsigned char val); /* output value to digit3 LED */
void led_out_digit4(unsigned char val); /* output value to digit4 LED */

void led_num_digit1(unsigned char num); /* display number in digit1 LED */
void led_num_digit2(unsigned char num); /* display number in digit2 LED */
void led_num_digit3(unsigned char num); /* display decimal number in digit3 LED */
void led_num_digit4(unsigned char num); /* display decimal number in digit4 LED */
void led_hex(unsigned short num); /* display 4 digit number as hex */
void led_bcd(unsigned short bcdnum); /* display 4 digit number as BCD */

void led_dp_digit1(unsigned char on); /* turn on or off digit1 DP */
void led_dp_digit2(unsigned char on); /* turn on or off digit2 DP */
void led_dp_digit3(unsigned char on); /* turn on or off digit3 DP */
void led_dp_digit4(unsigned char on); /* turn on or off digit4 DP */

void led_colon(unsigned char on); /* turn on or off L1 L2 colon */
void led_L1(unsigned char on); /* turn on or off L1 LED */
void led_L2(unsigned char on); /* turn on or off L2 LED */
void led_L3(unsigned char on); /* turn on or off L3 LED */

void led_mux_drive(void); /* interrupt mux driver */

#endif /* _LED_JJ2_H */

```

8.19 sw_vjj2.h**FILE ID: sw_vjj2.h**

```

/* sw_vjj2.h */
/* header for AF-V850 -JJ2 CPU evaluation board switch reading */
/* Version: 1.0 11-11-2006 */

#ifndef _SW_VJJ2_H
#define _SW_VJJ2_H

/*****
/* Define definitions */
*****/

/* symbolic definitions for switch inputs */
/* SW2 = bottom switch = P51 */
/* SW1 = top switch = P50 */
/*
/*
#define SW_LU_RU 0x03 /* SW2 up, SW1 up 1 1 */
#define SW_LD_RU 0x02 /* SW2 up, SW1 down 1 0 */
#define SW_LU_RD 0x01 /* SW2 down, SW1 up 0 1 */
#define SW_LD_RD 0x00 /* SW2 down, SW1 down 0 0 */

#define SW_DEF_DEB_COUNT 16 /* default debounce counter */

/*****
/* Export functions */
*****/
extern void sw_init(void); /* init ports and variables for switch input */
extern unsigned char sw_chk(void); /* get undebounced switch input */
extern unsigned char sw_get(void); /* get debounced switch input */
extern void sw_set_debounce(unsigned char count); /* set debounce count */
extern void sw_isr(void); /* debounce routine, called by timer ISR */

#endif /* _SW_VJJ2_H */

```


8.20 timer.h**FILE ID: timer.h**

```

/*
*****
**
** This device driver was created by Applilet for the V850ES/JG2, V850ES/JJ2
** 32-Bit Single-Chip Microcontrollers
**
** Copyright(C) NEC Electronics Corporation 2002-2006
** All rights reserved by NEC Electronics Corporation
**
** This program should be used on your own responsibility.
** NEC Electronics Corporation assumes no responsibility for any losses
** incurred by customers or third parties arising from the use of this file.
**
** Filename : timer.h
** Abstract : This file implements a device driver for the timer module
** APIlib : V850ESJx2.lib V1.50 [23 Feb. 2006]
**
** Device : uPD70F3721
**
** Compiler : NEC/CA850
**
*****
*/

#ifndef _MDTIMER_
#define _MDTIMER_

/*
** *****
** MacroDefine
** *****
*/
#define TM_TMP0_CLOCK 0x02 // fxx/4
#define TM_TMP0_INTERVALVALUE 0x2423
#define TM_TMP0_INTERVALVALUE2 0x7a11
#define TM_TMP0_ONESHOTOUTPUTCYCLE 0xf423
#define TM_TMP0_ONESHOTOUTPUTDELAY 0x7a11
#define TM_TMP0_EXTTRIGGERCYCLE 0xf423
#define TM_TMP0_EXTTRIGGERDELAY 0x7a11
#define TM_TMP0_PWMCYCLE 0xf423
#define TM_TMP0_PWMWIDTH 0x7a11
#define TM_TMP0_CCR0COMPARE 0xf423
#define TM_TMP0_CCR1COMPARE 0x7a11
#define TM_TMP1_CLOCK 0x0
#define TM_TMP1_INTERVALVALUE 0x00
#define TM_TMP1_INTERVALVALUE2 0x00
#define TM_TMP1_ONESHOTOUTPUTCYCLE 0x00
#define TM_TMP1_ONESHOTOUTPUTDELAY 0x00
#define TM_TMP1_EXTTRIGGERCYCLE 0x00
#define TM_TMP1_EXTTRIGGERDELAY 0x00
#define TM_TMP1_PWMCYCLE 0x00
#define TM_TMP1_PWMWIDTH 0x00
#define TM_TMP1_CCR0COMPARE 0x00

```

```

#define TM_TMP1_CCR1COMPARE      0x00
#define TM_TMP2_CLOCK            0x0
#define TM_TMP2_INTERVALVALUE    0x00
#define TM_TMP2_INTERVALVALUE2   0x00
#define TM_TMP2_ONESHOTOUTPUTCYCLE 0x00
#define TM_TMP2_ONESHOTOUTPUTDELAY 0x00
#define TM_TMP2_EXTTRIGGERCYCLE  0x00
#define TM_TMP2_EXTTRIGGERDELAY  0x00
#define TM_TMP2_PWMCYCLE         0x00
#define TM_TMP2_PWMWIDTH         0x00
#define TM_TMP2_CCR0COMPARE      0x00
#define TM_TMP2_CCR1COMPARE      0x00
#define TM_TMP3_CLOCK            0x0
#define TM_TMP3_INTERVALVALUE    0x00
#define TM_TMP3_INTERVALVALUE2   0x00
#define TM_TMP3_ONESHOTOUTPUTCYCLE 0x00
#define TM_TMP3_ONESHOTOUTPUTDELAY 0x00
#define TM_TMP3_EXTTRIGGERCYCLE  0x00
#define TM_TMP3_EXTTRIGGERDELAY  0x00
#define TM_TMP3_PWMCYCLE         0x00
#define TM_TMP3_PWMWIDTH         0x00
#define TM_TMP3_CCR0COMPARE      0x00
#define TM_TMP3_CCR1COMPARE      0x00
#define TM_TMP4_CLOCK            0x0
#define TM_TMP4_INTERVALVALUE    0x00
#define TM_TMP4_INTERVALVALUE2   0x00
#define TM_TMP4_ONESHOTOUTPUTCYCLE 0x00
#define TM_TMP4_ONESHOTOUTPUTDELAY 0x00
#define TM_TMP4_EXTTRIGGERCYCLE  0x00
#define TM_TMP4_EXTTRIGGERDELAY  0x00
#define TM_TMP4_PWMCYCLE         0x00
#define TM_TMP4_PWMWIDTH         0x00
#define TM_TMP4_CCR0COMPARE      0x00
#define TM_TMP4_CCR1COMPARE      0x00
#define TM_TMP5_CLOCK            0x0
#define TM_TMP5_INTERVALVALUE    0x00
#define TM_TMP5_INTERVALVALUE2   0x00
#define TM_TMP5_ONESHOTOUTPUTCYCLE 0x00
#define TM_TMP5_ONESHOTOUTPUTDELAY 0x00
#define TM_TMP5_EXTTRIGGERCYCLE  0x00
#define TM_TMP5_EXTTRIGGERDELAY  0x00
#define TM_TMP5_PWMCYCLE         0x00
#define TM_TMP5_PWMWIDTH         0x00
#define TM_TMP5_CCR0COMPARE      0x00
#define TM_TMP5_CCR1COMPARE      0x00
#define TM_TMP6_CLOCK            0x0
#define TM_TMP6_INTERVALVALUE    0x00
#define TM_TMP6_INTERVALVALUE2   0x00
#define TM_TMP6_ONESHOTOUTPUTCYCLE 0x00
#define TM_TMP6_ONESHOTOUTPUTDELAY 0x00
#define TM_TMP6_EXTTRIGGERCYCLE  0x00
#define TM_TMP6_EXTTRIGGERDELAY  0x00
#define TM_TMP6_PWMCYCLE         0x00
#define TM_TMP6_PWMWIDTH         0x00
#define TM_TMP6_CCR0COMPARE      0x00
#define TM_TMP6_CCR1COMPARE      0x00
#define TM_TMP7_CLOCK            0x0
#define TM_TMP7_INTERVALVALUE    0x00

```

```

#define TM_TMP7_INTERVALVALUE2 0x00
#define TM_TMP7_ONESHOTOUTPUTCYCLE 0x00
#define TM_TMP7_ONESHOTOUTPUTDELAY 0x00
#define TM_TMP7_EXTTRIGGERCYCLE 0x00
#define TM_TMP7_EXTTRIGGERDELAY 0x00
#define TM_TMP7_PWMCYCLE 0x00
#define TM_TMP7_PWMWIDTH 0x00
#define TM_TMP7_CCR0COMPARE 0x00
#define TM_TMP7_CCR1COMPARE 0x00
#define TM_TMP8_CLOCK 0x0
#define TM_TMP8_INTERVALVALUE 0x00
#define TM_TMP8_INTERVALVALUE2 0x00
#define TM_TMP8_ONESHOTOUTPUTCYCLE 0x00
#define TM_TMP8_ONESHOTOUTPUTDELAY 0x00
#define TM_TMP8_EXTTRIGGERCYCLE 0x00
#define TM_TMP8_EXTTRIGGERDELAY 0x00
#define TM_TMP8_PWMCYCLE 0x00
#define TM_TMP8_PWMWIDTH 0x00
#define TM_TMP8_CCR0COMPARE 0x00
#define TM_TMP8_CCR1COMPARE 0x00
#define TM_TMQ0_CLOCK 0x0
#define TM_TMQ0_INTERVALVALUE 0x00
#define TM_TMQ0_INTERVALVALUE2 0x00
#define TM_TMQ0_INTERVALVALUE3 0x00
#define TM_TMQ0_INTERVALVALUE4 0x00
#define TM_TMQ0_ONESHOTOUTPUTCYCLE 0x00
#define TM_TMQ0_ONESHOTOUTPUTDELAY 0x00
#define TM_TMQ0_ONESHOTOUTPUTDELAY2 0x00
#define TM_TMQ0_ONESHOTOUTPUTDELAY3 0x00
#define TM_TMQ0_EXTTRIGGERCYCLE 0x00
#define TM_TMQ0_EXTTRIGGERDELAY 0x00
#define TM_TMQ0_EXTTRIGGERDELAY2 0x00
#define TM_TMQ0_EXTTRIGGERDELAY3 0x00
#define TM_TMQ0_PWMCYCLE 0x00
#define TM_TMQ0_PWMWIDTH 0x00
#define TM_TMQ0_PWMWIDTH2 0x00
#define TM_TMQ0_PWMWIDTH3 0x00
#define TM_TMQ0_CCR0COMPARE 0x00
#define TM_TMQ0_CCR1COMPARE 0x00
#define TM_TMQ0_CCR2COMPARE 0x00
#define TM_TMQ0_CCR3COMPARE 0x00
#define TM_TMM_CLOCK 0x0
#define TM_TMM_INTERVALVALUE 0x7cf

void TMP0_Init( void );
void TMP0_Start( void );
void TMP0_Stop( void );
MD_STATUS TMP0_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
__multi_interrupt void MD_INTTP0CC0( void );
void TMP0_User_Init( void );
void delay(int count);

#endif

```

9. Appendix C — Development Tools

The following software and hardware tools were used in the development of this application note.

9.1 Software Tools

- ◆ Applilet — code generation tool, applilet_v850es_jx2_v150.exe
- ◆ The compiler, assembler and linker are part of a package called CA850 Compiler.
- ◆ Project Manager PM+ — the integrated development environment (IDE)

9.2 Hardware Tools

- ◆ Microsoft Windows 2000 or Windows XP
- ◆ Demo Board AF-EV850 Basic Rev 1.0
- ◆ MiniCube2 with USB interface

10. Appendix D — Applicable Documents

1. User's Manual V850ES 32-Bit Microprocessor Core Architecture
Document No. U15943EJ3V0UM00 (3rd Edition)
2. User's Manual V850ES/JJ2 32-Bit Single-Chip Microcontrollers.
Document No U17714EJ2V0UD00 (2nd Edition)
3. User's Manual CA850 Ver 3.00 C Compiler Package
C Language Target Device V850 Series
Document No. U17291EJ2V0UM00 (2nd Edition Nov 2004)
4. User's Manual CA850 Ver 3.00 C Compiler Package
Operation Target Device V850 Series
Document No. U17293EJ2V0UM00 (2nd Edition Nov. 2004)
5. User's Manual CA850 Ver 3.00 C Compiler Package
Link Directives Target Device V850 Series
Document No. U17294EJ2V0UM00 (2nd Edition Nov. 2004)
6. User's Manual ID850 QB Ver 3.20 Integrated Debugger
Operation Target Device V850 Series
Document No. U17964EJ1V0UM00 (1st Edition)
7. SD Specifications, PART 1 PHYSICAL LAYER Simplified Specification
Version 1.10 April 3, 2006
8. SanDisk MultiMediaCard and Reduced-Size MultiMediaCard
Product Manual Version 1.3 Document No. 80-36-00320 April 2005
9. MultiMediaCard Specification Ver 0.9 June 2004
Samsung Electronics., LTD
10. LITEON LTC-4627JR Specification
11. AF-EV850 Basic Rev 1.0
NEC Electronics America, Inc.
AV-EV850 Basic Schematic
12. Preliminary User's Manual QB-MINI2
On-Chip Debug Emulator and Programming Function
Document ZUD-CD-06-0018-2-E June 23, 2006
23. QB-MINI2 Operating Precautions
Document No. ZUD-CD-06-0046-4 Aug 24, 2006
14. Preliminary User's Manual QB-Programmer
Programming GUI Operation
Document No. ZUD-CD-06-0006-1 E June 12, 2006

11. Appendix E — Modifications for MiniCube2

The MiniCube2 is an on-chip debug emulator with flash programming function, which is used for debugging and programming a program to be embedded in on-chip flash memory microcontrollers. It uses a USB connection to the development PC.

The MiniCube uses a piece of monitor code that is loaded with the development code. In order to accommodate this code and some changes to control lines, the following changes are needed.

11.1 System Initialization Modifications

Changes are required in the following files:

```
crte.s
    increase stack size from 0x200 to 0x800
    set up ROM area for monitor to use
    set up RAM area for monitor to use
    set up vector DBG0 for monitor
#-----
#    Monitor Area
#-----

#--Secures 2KB space for monitor ROM section
    .section    "MonitorROM", const
    .space      0x800, 0xff

#--Secures interrupt vector for debugging at 0x0060
    .section    "DBG0"
    .space      4, 0xff

-- Secures 16 byte space for monitor RAM section
    .section    "MonitorRAM", bss
    .lcomm      monitorramsym,16,4    -- defines monitorramsym symbol

inttab.s
    INTP0
    INTCB4R, INTCB4T removed
    INTP0CC0,    allow monitor to modify vector
    INTCB0R      allow monitor to modify vector
    INTCB0T      allow monitor to modify vector
```

port.c

Port 4 and PortCM are used by MiniCube, remove initialization that Applilet has added.

11.2 Link Directive Changes in 850.dir

Adjust memory layout to accommodate the MiniCube2.

```
/* monitor needs 0x800 bytes at end of ROM, rom ends at 0x0003FFFF
MROMSEG      : !LOAD ?R V0x0003f800{
               MonitorROM          = $PROGBITS           ?A MonitorROM;
};

/* end of JJ2 3721 RAM ends at 0x03ffefff, monitor needs 16 bytes
MRAMSEG      : !LOAD ?RW V0x03ffefe0{
               MonitorRAM          = $NOBITS ?AW MonitorRAM;
};
```

12. Appendix F— Port Association List

The following list shows which device is connected to which port.

UART - RS232 (CSIB3) cpu		UART - USB (CSIB4) cpu	
P81 (Tx)	TXDA3 P31	(Txd)	RXDA0
P80 (Rx)	RXDA3 P30	(Rxd)	TXDA0

LED Seven Segment display		SW2	P51	in
	1 turns on	SW1	P50	in
P60	Digit 1 common anode			out
P61	Digit 2			out
P64	colon, top dot			out
P62	Digit 3			out
P63	Digit 4			out
	0 turns on			
P90	segment A L1 (upper dot of colon)			out
P91	segment B L2 (lower dot of colon)			out
P92	segment C L3			out
P93	segment D			out
P94	segment E			out
P95	segment F			out
P96	segment G			out
P97	decimal point			out

SD card CN8	(CSIB5) cpu	U6 EEPROM - I2C0 (aka UARTA2)
P04 IRQ	INTP1 in	P39 SCL00
P67 DO	SOB5 out	P38 SDA00
P68 SCLK	SCKB5 out	
P66 DI	SIB5 in	
P35 /CS	out	

ZigBee CN3			MiniCube2 CN6		
chipcon CC2420EM		cpu			cpu
P10	FIFO	in	P41	SI	SOB0
P03	FIFOP	in	P40	SO	SIB0
P11	CCA	in	P42	SCK	/SCKB0
P33	SFD	in	AD5	FLMD1	
P34	CSn	out	FLMD0	FLMD0	
P68	SCLK	SCKB5			
P67	SI	SOB5			
P66	SO	SIB5			