NEC

Application Note

DMA Transfers to SD Card and External SRAM

Document no. U19133EU1V0AN00

The information in this document is current as of February 2008. 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 's 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
February 2008	—	—	First release

Contents

1. Introduction

This application note provides simple examples of the use of the direct memory access (DMA) controller with other peripherals included in NEC Electronics microcontrollers. The examples illustrate the use of the DMA controller in several typical situations. The AF-EV850 Basic evaluation board, which uses the V850ES JJ2 processor (70F3721), serves as the demonstration platform.

Other features demonstrated in this application note are:

- External-memory access
- Interval-timer operation
- A/D converter use
- Interrupt-driven UART, using round-robin input buffering
- MiniCube2 debug

This document includes:

- Descriptions of peripheral features
- Program descriptions and specifications
- Settings for the Applilet
- Demonstration platform description
- Hardware block diagrams
- Software flow charts
- Software modules

The Applilet is a software tool for generating simple driver code for the processor peripherals. This tool provides a convenient means of generating the initial code for the on-chip peripherals for quick evaluation. Although the code generated usually requires modification for customization to specific requirements, the Applilet provides a quick start. The code generated by the Applilet is derived from the settings described in the section “Applilet Selections.”

Reference the microcontroller’s user manual and other related documents for additional details.

2. Overview of DMA Transfer Operations

NEC Electronics V850ES Series microcontrollers offer multi-channel, high-speed DMA controllers as on-chip peripherals. The microcontrollers support both single-byte and block DMA transfers and perform the DMA operations listed in the table below.

Table 1. DMA Transfers Demonstrated

From	To
Internal Memory	Peripheral I/O (SD Memory Card)
Internal Memory	External Memory (SRAM module)
External Memory (SRAM module)	External Memory (SRAM module)
Peripheral I/O (SD Memory Card)	External Memory (SRAM module)
Peripheral I/O (A/D Converter)	Internal Memory

This application note demonstrates how the DMA controller in a V850ES Series microcontroller transfers data to and from an SD memory card (a removable mass-storage device that uses the same public protocol as MMC). The demonstration uses data obtained by taking readings with the MCU’s analog-to-digital converter at timed intervals and automatically storing the readings in memory. The controller also performs the routine task of quickly moving blocks of data from one area of memory to another.

3. Features

3.1 NEC Electronics DMA Controller Features

The DMA controller implemented as an on-chip peripheral for most NEC Electronics microcontrollers allows the transfer of large data blocks without CPU intervention. After the DMA control registers are programmed, the DMA transfer automatically relocates data from the source to the destination.

The DMA controller controls data transfers between memory and peripheral I/O, between memories, or between peripheral I/O and peripheral I/O, depending on the DMA register settings. A typical DMA controller implemented in NEC Electronics microcontrollers features:

- Multiple independent DMA channels
- 8- or 16-bit data transfers
- 64K maximum transfer count
- Support for 2-cycle data transfers—read cycle followed by a write cycle
- Support for single-transfer mode—releases the bus at each byte/half word transfer
- Transfer request sources
  - Interrupt by on-chip peripheral
  - External input pin
  - Request by software
- Transfer types
  - Internal RAM to peripheral I/O
  - Peripheral I/O to peripheral I/O
  - Internal RAM to external memory
  - External memory to external memory

3.1.1 Registers Controlling DMA Operations

The register settings shown in the table below control DMA operation.

Table 2. Registers Controlling DMA Operation

DMA Control Registers	Register Name	Descriptions
DMA Source Address Register	DSA0 - DSA3	Typically 26 Bits, Defines DMA Source Address
DMA Destination Address Register	DDA0 - DDA3	Typically 26 Bits, Defines DMA Destination Address
DMA Byte Count Register	DBC0 - DBC3	A 16-Bit Register, Sets Transfer Byte Count
DMA Addressing Control Register	DADC0 - DADC3	Sets Transfer Data Size, Source/Dest. Count Direction
DMA Channel Control Register	DCHC0 - DCHC3	DMA Transfer Enable/Disable, Transfer Complete Flag
DMA Trigger Factor Register	DTFR0 - DTFR3	DMA Transfer Interrupt Request by On-chip Peripherals

3.1.2 DMA Transfer Mode

Most V850ES Series microcontrollers support single-transfer mode. In the single-transfer mode, the bus is released at each byte/half word transfer. For a subsequent DMA transfer request, the transfer is performed again. This operation continues until a terminal count occurs.

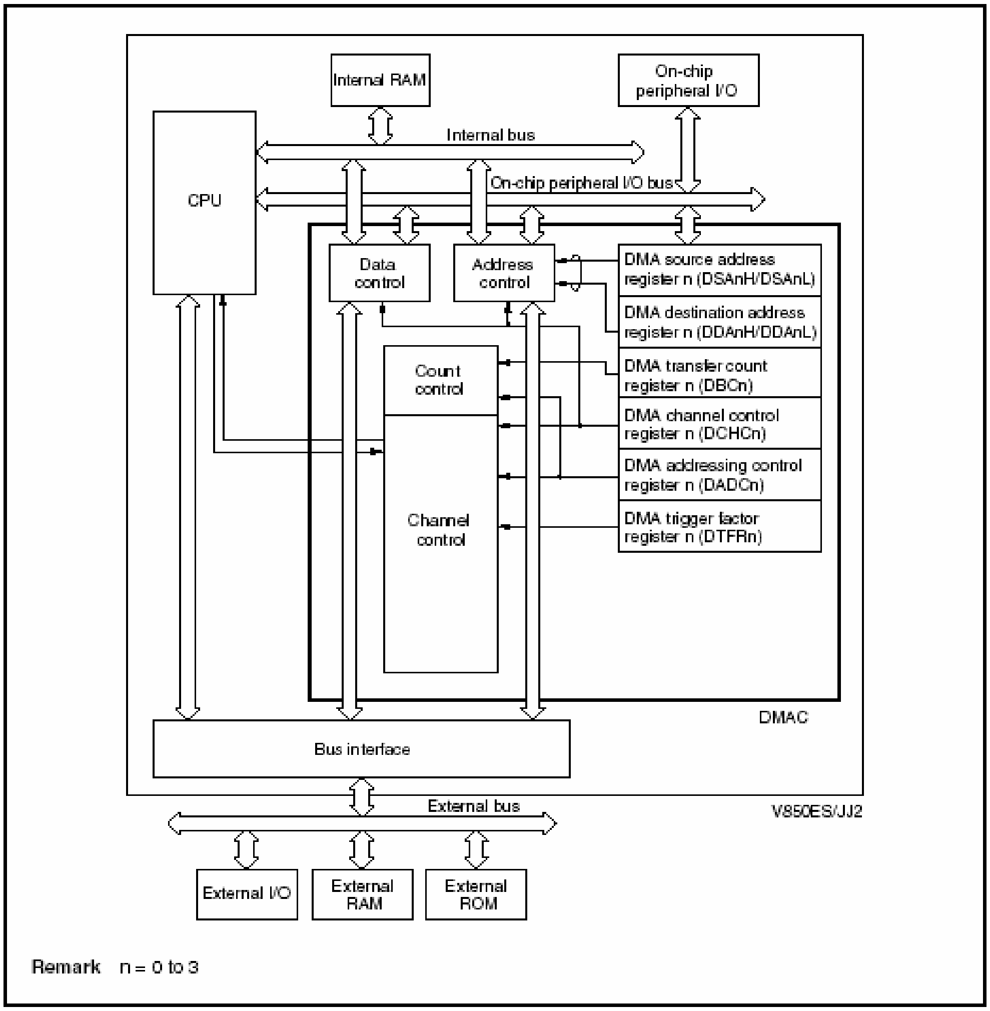
3.1.3 DMA Transfer Type

Most V850ES Series microcontrollers support two-cycle transfers, where each data transfer includes a read and a write cycle.

During the read cycle, the controller outputs the transfer source address and reads from the source. During the write cycle, the controller outputs the transfer destination address and writes to the destination. An idle cycle of one clock is always inserted between the read and write cycles.

Figure 1 shows a block diagram of the DMA controller in the V850ES/JJ2 microcontroller.

Figure 1. DMA Controller Block Diagram


Internal RAM
On-chip peripheral I/O
Internal bus
CPU
On-chip peripheral I/O bus
Data control
Address control
DMA source address register n (DSAnH/DSAnL)
DMA destination address register n (DDAnH/DDAnL)
DMA transfer count register n (DBCn)
DMA channel control register n (DCHCn)
DMA addressing control register n (DADCn)
DMA trigger factor register n (DTFRn)
Count control
Channel control
DMAC
Bus interface
External bus
V850ES/JJ2
External I/O
External RAM
External ROM
Remark n = 0 to 3

3.2 SD Memory Card System Features

An SD memory card serves as the peripheral I/O device for this DMA transfer demonstration, which uses the Synchronous Peripheral Interface (SPI) mode for transfers. The physical specification and operating information for the SPI mode may be found in the reference documents from the SD Memory Card Association:

- SD Memory Card Physical Layer Specification, Version-1.01
- SD Specification, Part-E1 SDIO Simplified Specification, Version-1.10
- Technical Committee, SD Card Association

The SD memory card provides high capacity and performance with built-in security features. SD memory card communication is based on a 9-pin interface that provides clock, command, four data and three power lines. The maximum operating frequency can be as high as 25 MHz.

The SD memory card provides applications with low-cost storage, implemented with removable cards that support high security levels and a compact, easy-to-implement interface. SD memory cards can be read/write (such as flash, or one-time programmable memory) or read-only (ROM).

3.2.1 SD Memory Card Hardware Interface

An SD memory card system has two alternative communication methods: SD and SPI. The SD communication mode uses wider bus width, thereby achieving higher-speed data transfers. The SPI standard defines the physical link with the host microcontroller and is commonly used in microcontroller-based designs. The SD memory-card SPI interface uses the signals shown in the table below.

Table 3. SD Memory-Card SPI Interface Control Signals

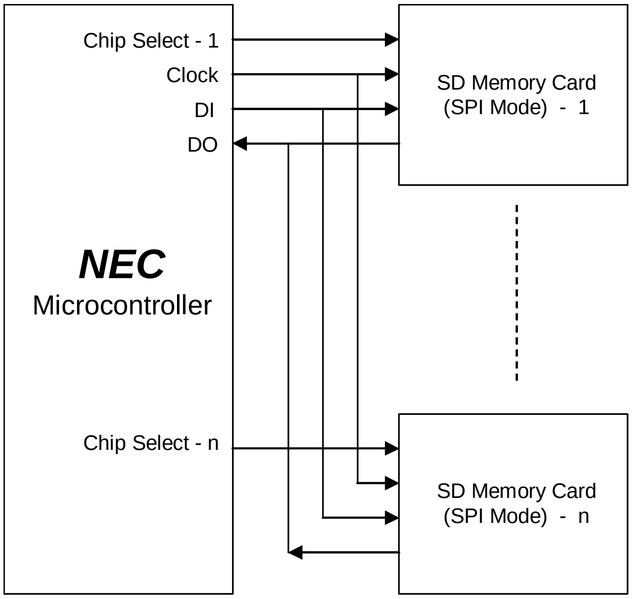
Signal	Name	Description
Chip Select	CS	Host Microcontroller to DS Memory Card Chip Select signal
Clock	CLK	Host Microcontroller to SD Memory Card Clock Signal
Data Input	DI	Host Microcontroller to SD Memory Card Data Input signal
Data Output	DO	SD Memory Card to Host Microcontroller Data Output signal

The SD Memory Card system uses nine pins to interface with the microcontroller. The pin assignments for the SPI interface are shown in the following table.

Table 4. SPI Interface Pin Assignments

Pin Number	Signal Name	I/O	Description
1	CS	Input	Chip Select Signal, Input with Push-Pull Driver
2	DI	Input	Data Input, Input with Push-Pull Driver
3	VSS	Power	GND
4	VDD	Power	VDD
5	CLK	Input	Clock Input, Input with Push-Pull Driver
6	VSS2	Power	GND
7	DO	Output	Data Output, Output with Push-Pull Driver
8	RSV	Reserved	
9	RSV	Reserved	

Figure 2. SD Interface with V850ES Series Microcontroller via CSI Interface



3.3 SRAM Module Features

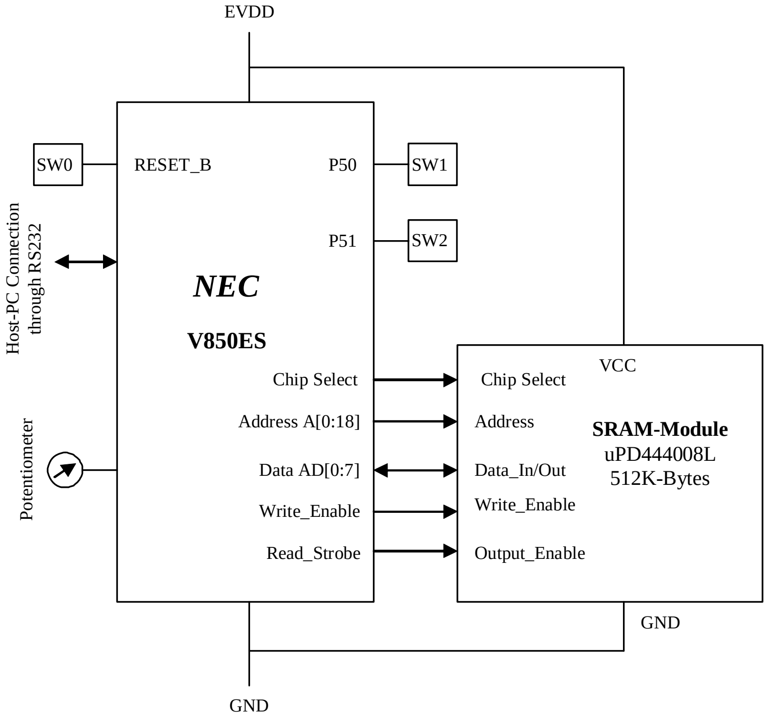
A 512K-byte SRAM module serves as external memory for the V850ES/JJ2 microcontroller. The SRAM module connects to the V850ES/JJ2 external memory interface signals—its address, data, and read/write control signals. The microcontroller functions as the memory controller. The table below shows the truth table for the SRAM module-access control signals.

Table 5. SRAM Module Access Control Truth Table

/CS	/OE	/WE	Mode	I/O	Supply Current
H	X	X	Not selected	High impedance	ISB
L	L	H	Read	Dout	ICC
L	X	L	Write	DIN	
L	H	H	Output disable	High impedance	

Note: X = Don’t care

Figure 3. SRAM Module Connection to V850ES/JJ2 Microcontroller



3.4 A/D Converter Features

The microcontroller’s A/D converter has a resolution of 10 bits and can handle 16 analog input-signal channels by selecting one as input and using successive approximation. The input signal must be no more than 3.6 volts.

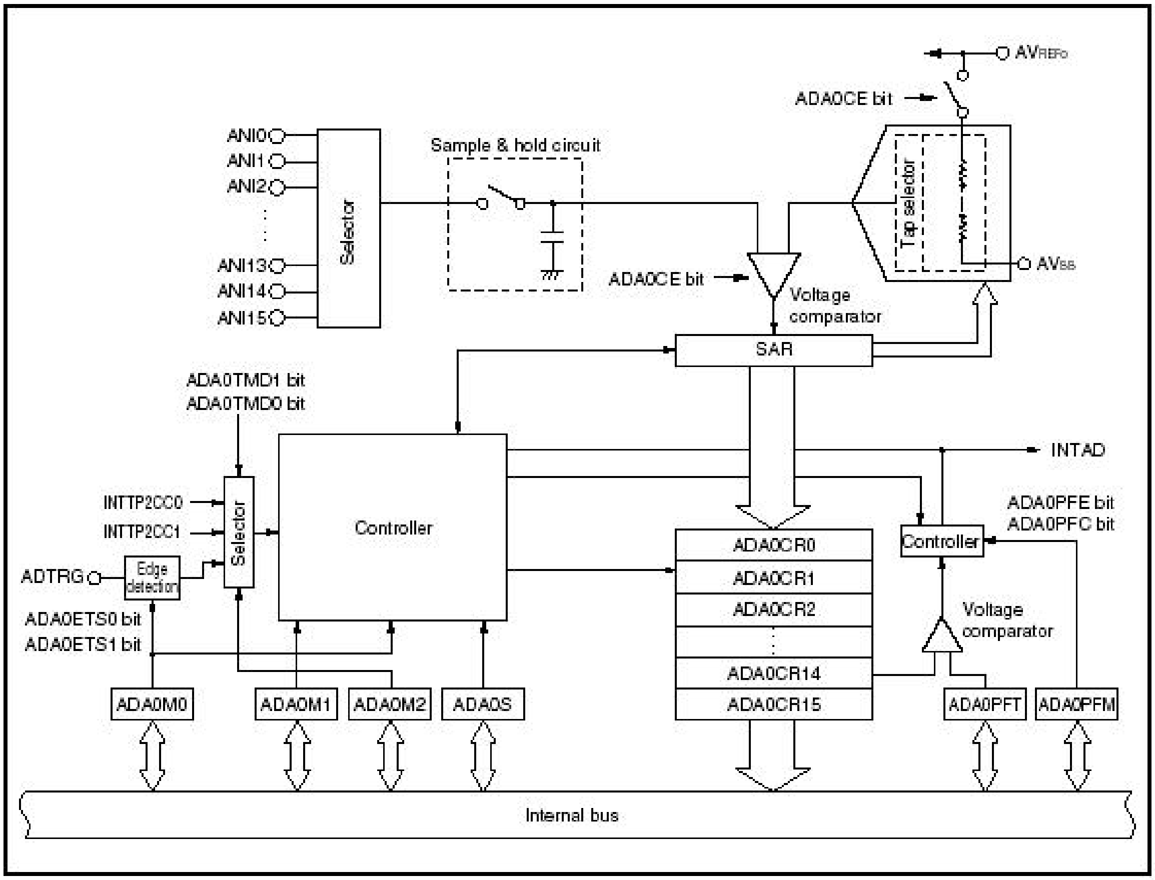
The converter operates in the following modes:

- Continuous select mode
- Continuous scan mode
- One-shot select mode
- One-shot scan mode

Any of the following triggers can initiate conversion:

- Software trigger mode
- External trigger mode
- Timer trigger

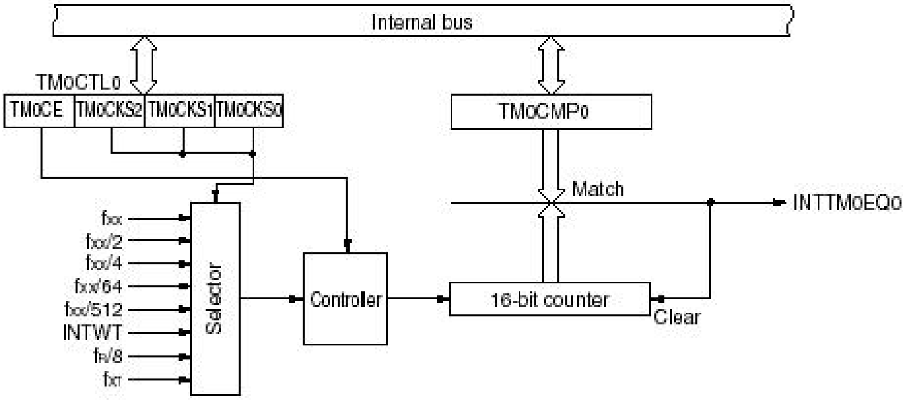
Figure 4. Block Diagram of A/D Converter System



3.5 Timer Module Features

Timers 0 and 2 are set as interval timers for this demonstration. They use the internal system clock fxx, which is divided down. This clock drives a counter. When the counter matches a compare value, the timer generates an interrupt, then the counter resumes counting.

Figure 5. Interval Timer System



Remark f_{XX}: Main clock frequency

f_R: Internal oscillation clock frequency

f_{XT}: Subclock frequency

INTWT: Watch timer interrupt request signal

4. Program Description and Specification

The demonstration program uses a V850ES/KJJ2 board (AF-V850ES/JJ2) with an SD memory card for peripheral I/O and an SRAM module for external memory. The following table shows the DMA transfers demonstrated.

Table 6. DMA Transfers Demonstrated

From	To
Internal Memory	Peripheral I/O (SD Memory Card)
Internal Memory	External Memory (SRAM Module)
External Memory (SRAM Module)	External Memory (SRAM Module)
Peripheral I/O (SD Memory Card)	External Memory (SRAM Module)
Peripheral I/O (A/D Converter)	Internal Memory

The example program is not intended to demonstrate every possible combination of DMA transfer.

4.1 Peripheral Subsystem Setup and Initialization

4.1.1 External Memory Address/Data Bus Controller

The function Bus_init() sets up and initializes the bus controller, which is an alternate function of selected pins on ports D, CM, CS and CT. These pins are listed in the port association list in Appendix F. Reference the port_bus_interface flowchart in Appendix B.

Bus_init() is called by systeminit.c as part of the startup initialization procedure. Variables for bus size configuration, data-wait, address-wait and bus-cycle control are initialized to their default settings, then customized for the current application:

- Register PMCCM of the port CM control is set to ZERO to negate the /CLKOUT, /WAIT, /HLDACK and /HLDRQ lines and to select the normal I/O port function.
- Setting mode-control register PMCCT of port CT to 0x93 selects the alternate functions of memory write /WR0, /WR1 and memory read /RD.
- Setting mode-control register PMCCS of port CS to 0x02 selects the alternate functions for chip select /CS1.
- Setting register EXIMC to 0x01, followed by a NOP instruction, selects the separate bus-addressing mode rather than the multiplexed bus.
- Setting register BSC to 0x5551 (BS10=0) specifies that /CS1 controls an 8-bit bus.
- Setting register DWC0 to 0x3373 establishes the number of wait states for /CS1 memory accesses to seven.
- Setting register AWC to 0xFFF3 does not insert an address hold wait to extend the high period of T1 state by one clock, and specifies that the address-setup wait not be inserted, as this would add an additional cycle to the low period of the T1 state.
- Setting register BCC to 0xAAA2 specifies that no idle state is inserted for /CS1 memory accesses. The external SRAM is connected with chip select 1 /CS1, dictating that the memory is in the range

0x00200000 through 0x003FFFFF. Since the chip is 4M, the effective end address is 0x0028FFFF.

- Setting the port DH mode-control register PMCDH to 0xFF selects an alternate function, which is the output of the address-line data for address lines A16 through A23. For this memory chip, the demonstration uses only lines A16, A17 and A18.
- Setting port DL mode-control register PMCDL to 0x00FF sets the address/data bus I/O lower 8 bits to an alternate function. Because this setting does not use the multiplexed bus method, these lines are strictly data bus I/O. The memory is 8 bits wide, so the demonstration uses only the lower 8 bits—AD0..AD7.
- Setting port 9 function-control register PFC9 to 0x0000 and port 9 extended function control PFCE9 to 0x0000 selects address-line data for output of address lines A0 through A15. Then, setting port 9 mode-control register PMC9H to 0xFFFF selects an alternate function, as previously selected by PFC9 and PFCE9. Setting port 9 function register PF9 to 0x0000 selects normal CMOS output.

4.1.2 DMA Initialization

In this demonstration, accesses to the SD memory card use two DMA controller channels. Only one channel is needed for other functions.

SystemInit calls DMA0_Init() and DMA1_Init() to initialize the channels. These routines modify almost all the settings.

Reference the DMA interface flowchart in Appendix B.

4.1.2.1 DMA Setup

When using the DMA controller, calling DMAn_source_address sets the data-source address and establishes whether the address will increment, decrement or stay fixed. Next, a call to DMAn_destination_address sets the destination address and whether the address will increment, decrement or stay fixed. These functions determine if the address is internal or external; an address within the range of 0x03ff0000 through 0x03ffffff is internal. Onboard peripherals are considered external. You must set the transfer count and trigger factor.

4.1.2.2 DMA Setup for SD

If the device being accessed is CSIB5, the controller used for SD memory via SPI, then two additional functions are available—DMA0_Setup_mmc and DMA1_Setup_mmc. These functions call DMAn_Halt to stop the DMA controller, as well as DMAn_source_address and DMAn_destination_address to set up the appropriate controller. For the SPI interface, every write is accompanied by a read of the same size. These functions also set the transfer count and the trigger factor. Known addresses are supplied, so that the caller needs to provide less information.

Several options, called triggers, can initiate the DMA transfer. Triggers are not required for memory-to-memory transfers, and the operation free-runs to completion. Interrupts can serve as triggers for transfers to or from devices. An operation that is dependent on some software process can transfer based on a manual trigger.

For the demonstration using the A/D converter, the trigger is the A/D conversion-completion interrupt.

4.1.3 A/D Converter Initialization

The AD_Init() function, which is called once by systeminit.c, initializes the A/D converter. Reference the A/D interface flowchart in Appendix B.

AD_Init() selects channel AD1 for input. Channel AD1 connects to the potentiometer on the evaluation board. The DMA interface demonstration described in this application note uses the timer to determine when to take a sample. An interrupt from interval timer 2 triggers A/D conversion. The A/D completion interrupt triggers the DMA transfer of the conversion results to memory.

User selection of demonstration mode 5 converts 256 A/D samples (at a rate dictated by interval timer 2) and stores them in buffer2. For this demonstration, the rate must be slow enough to allow you to change the potentiometer and observe the resulting changes in the data. After taking the samples, the buffer dumps to the console screen.

4.1.3.1 A/D Setup and Initialization

This application uses DMA channel 0 for receive and channel 1 for transmit. Setting register DTFR0 to a value of 0xAC clears any existing trigger and selects the A/D interrupt as the trigger for the DMA0 transfer.

Calling function AD_Start() enables the conversion completion interrupt; calling DMA0_Start enables the DMA0 operation previously set up. The last step in starting automatic, periodic A/D sampling is to call TMP2_Start() to enable the periodic interrupt.

The routine then enters a loop to wait for DMA0 transfer completion, which happens after 256 transfers. The routines sets a done flag when the count has decremented to -1. The variable sample_ticks keeps a count of timer interrupts for use in checking whether transfers have completed. If the transfer does not complete in time, an error message is output and the done flag is set.

When the done flag equals 1, the routine calls AD_Stop() to stop the A/D conversion and disable the interrupt. A call to TMP2_Stop() stops the interval timer and related interrupts. Finally, a call to DMA0_Stop() stops further transfers by the DMA0 controller. The routine returns to the main program loop.

Function ad_to_mem sets the operation-mode register ADAM0 to stop all conversion, and selects between external trigger mode or time trigger mode. This demonstration uses the timer trigger.

The function sets the conversion time. The processor has a system clock source of fxx=20 MHz, but you change the potentiometer slowly. As this sampling does not require fast conversion time, the demonstration uses a time of 20.7 µs. Setting register bit ADA0HS1 to ZERO selects the normal conversion speed.

Converter mode register 2 ADAM02 is set to select the interval timer 2 interrupt for conversion trigger input. This trigger is on interrupt INTTP2CC0. The function then selects the input channel, which is port 7 pin P71.

The function stops DMA controllers 0 and 1. Although the demonstration only uses DMA0 to input data, both DMA controllers are stopped because the routine cannot know what functions ran previously. The function sets register DBC0 to the number of samples to be taken, which was input parameter "samples."

Calling DMA0_source_addr() formats and sets the address of ASA0CR1, which is considered an external device at a fixed address. Calling DMA0_destination_addr formats and sets the address of buffer2, which was passed to the function as an input parameter destination. DMA0 address control increments for each transfer to the destination.

4.1.4 Serial-Interface Functions

Reference the SPI_3.0_serial_interface flowchart in Appendix A. Reference the serial.c and serial_user.c source code in Appendix B.

4.1.4.1 Serial-Interface Initialization

- ♦ UART3_Init()
- ♦ UART3_UserInit()

Selecting the alternate function of port PMC8 accesses the UART3 (Universal Asynchronous Receive Transmit) function, which is set to provide 8 data bits, no parity bit, one stop bit and to send the least significant bit first. The baud rate division is based on a 20-MHz clock, and is 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.

4.1.4.2 Serial-Interface Transmit

- ♦ UART3_SendData()
- ♦ MD_INTUA3T()
- ♦ uart3_tx_msg()

The serial-transmit operation is interrupt driven and initiated by a call to the function UART3_SendData(). Input parameters include the pointer to the string to be sent and the number of characters to be sent. On entry, the 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 string pointer increments by one and the number of characters to be sent decrements by one. Control then returns to the caller, and the remainder of the string is sent as part of the interrupt processing.

Completion of the transmission of the character calls the interrupt vector MD_INTUA3T. This routine checks the count of characters to be sent. If there are no more characters to be sent, the routine sets the done flag and exits the interrupt service. If there are characters to send, the function writes the next character to the UART transmit register, increments the data pointer and the decrements the number of characters remaining to send. Finally, the routine exits the UART transmit interrupt-service routine.

4.1.4.3 Serial-Interface Receive

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

To receive data over the serial connection, the routines enable the UART receiver and the receive interrupt.

Receiving a character causes the UART to generate an interrupt, and program execution vectors to the MD_INTUA3R interrupt-service routine. Immediately, this routine enables interrupts to prevent them from being blocked. The function reads the UART status register and checks for errors. If there are no receive errors, the routine reads the receive register and calls function UART3_Receive() to place the data into the round-robin buffer.

4.1.5 SPI Interface Initialization

- CSIB5_Init()

This routine configures Port PMC6 to function as the SPI interface, first stopping all operations, and turning off and clearing interrupts. The function configures CSIB5 to operate MSB first, with 8-bit data transfers, in single-transfer mode. The routine selects a transmit clock speed of fxx/64, which is 312.5kHz. 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()

Send and receive data are a single function, since the SPI interface requires that you send data to receive data. This routine sends the specified number of bytes from the specified address, then receives the same number of bytes and places them in a SPI receive buffer.

- CSIB5_select_SPI()
- CSIB5_deselect_SPI()

4.1.6 Timer functions

Reference the SPI_5.0_timer_interface flowchart in Appendix A. Reference the timer.c and timer_user.c source code in Appendix B.

4.1.6.1 Timer Initialization

- TMP0_Init()
- TMP0_User_Init()
- TMP0_Start()

The 16-bit timer, P, provides the interval (periodic) timer interrupt, and uses the internal clock `fxx/4` for input. After reaching the set count, the timer restarts. The main function calling TMP0_Start starts the interval timer.

4.1.6.2 Timer Interface

- MD_INTTP0CC0()
- SetMsecTimer()
- CheckMsecTimer()
- delay()

The periodic operations are:

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

The timer counter value increments, and when the counter value reaches the value in the interval-count register, the counter generates an interrupt, vectoring program execution to the interrupt-service routine at MD_INTTP0CC0. The interrupt-service routine calls sw_isr() and led_mux_drive().

4.2 Port Functions (Including Switch Input and LED Output)

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

4.2.1 Port Initialization

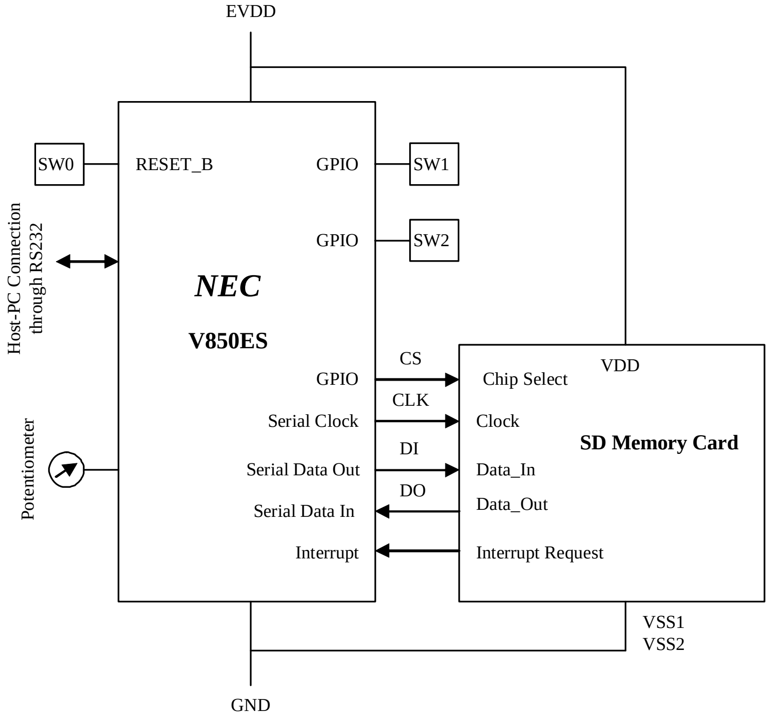
- PORT_Init()

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

4.3 Peripheral Subsystem Operation

4.3.1 DMA Transfer from Internal Memory to Peripheral I/O (SPI)

Figure 6. Data Transfers from Buffer to Memory Card



For use in demonstrating DMA transfers from internal memory to peripheral I/O, you need to create a data buffer on the host PC, as described in the section “Operator Selection of Test Data.” The DMA functions write the buffer contents into an SD memory card. Then the DMA controller reads the data written into the card and displays the data on the host PC’s monitor.

The memory card interface signals connect to a serial I/O port of the V850ES Series microcontroller. The microcontroller connects to the host PC through an RS-232 (UART) interface, using a terminal-emulator program on the host PC. You select this operating mode by entering “1” in response to the demonstration menu.

If you enter “1” or “4” you are prompted for a sector number, because the program calls function get_sector() to get a sector. Then, the program calls function mem_to_mmc to perform the memory-to-memory-card transfer. The function passes parameters of the sector to write to and the start address of buffer1, which contains the data to write. The transfer is always one sector—512 bytes. The next section “SDWiriteSector” describes the SDWriteSector function.

On entry into the mem_to_mmc function, the routine calls DMA0_Stop and DMA1_Stop to stop operations on DMA0 and DMA1. The function then calls SDWriteSector, passing the parameters of the sector number and the address of the data block to be written.

On completion, the function stops DMA0 and DMA1 again before returning to the main loop.

4.3.1.1 SDWriteSector

The SDWiriteSector function writes 512 data bytes to the specified sector. After selecting the device by lowering the chip-select line, the function builds and sends a CMD24 message. The CMD24 message contains the destination block address, which is the sector number multiplied by 512.

At some point after the message has been sent, the SD card responds with an R1 message. The function then sends count NWR pad characters, as specified by document reference 8 (Appendix D), Table 5-11. The routine switches the transfer mode to 16-bit transfers and sets the DMA source and destination addresses. The function then sends the data.

Upon completion, the function switches the transfer mode back to polled 8-bit mode, and checks for receipt of a data-response token. After receiving the response token, the function sends count NEC pad characters and deselects the device. (Reference 8 (Appendix D) Table 5-11 defines the pad characters.)

For more details, refer to the source code and flowcharts.

The routine checks the return status; if there are errors, the program reports those errors and sets the done flag. If the done flag is not set, calling CSIB5_polled_mode(OFF) sets 16-bit mode (remember that these are in little endian order).

Calling CSIB5_ReceiveDataBlock sets up for receiving the data that returns from the SD card (the same data sent to the card). This data is not used, however, this step provides consistency in handling the data. Calling CSIB5_SendDataBlock sends the actual data to be written to the SD card.

On return, calling CSIB5_polled_mode(ON) puts the controller back in 8-bit polled mode. The routine then calls function check_send_done() to check whether the completion flag CSIB5_send_done is set. The DMA completion interrupt sets this flag after all sector data is written to the device. The check_send_done() routine checks for a specified amount of time before declaring timeout. If check_send_done() returns a timeout error, the function deselects the SPI device and returns a timeout error.

If the transfer completes correctly, the routine calls SDmemory_DR_query to poll for receipt of the data-response token. This token is sent after the card completes writing the data to flash memory.

If the response is not received in a specified number of tries, SDmemory_DR_query returns a value of MD_REQ_TIMEOUT. Otherwise, the function decodes the response token and displays the appropriate message.

Calling send_pad sends NEC pad characters. The routine deselects the memory card and returns a status of MD_MASTER_SEND_END.

4.3.1.2 CSIB5_ReceiveDataBlock

On entry, CSIB5_ReceiveDataBlock sets the CSIB5_rcv_done flag to ZERO. Receipt of the DMA0 completion interrupt sets this flag to ONE. The routine saves the input parameter containing the number of bytes to receive in csibf_rcv_size, and the receive buffer address input parameter in csib5_rcv_pbuf. The routine then calls DMA0_Setup_mmc to set up DMA controller 0 to receive from SPI port CSIB5, and store data in the receive buffer specified, and to use a trigger factor of 0x38. On return, calling DMA0_Start enables the DMA0 controller. The routine sets register CB5RIC to enable interrupts on receipt of SPI data. This interrupt is the trigger to the DMA0 controller transfer.

4.3.1.3 DMA0_Setup_mmc

On entry, DMA0_Setup_mmc calls DMA0_Halt to stop DMA0 controller operation. The function then calls DMA0_source_addr with the parameters of the source address of 0xfffffd54, which is register CB5RX, and the mode parameter of DMA_FIXED. Thus, the source does not increment or decrement after each transfer.

On return, DMA0_Setup_mmc calls DMA0_destination_addr with the destination buffer address of the SPI read. The mode parameter is DMA_INCR, indicating that the address increments after every transfer to the receive buffer. On return, the routine puts in register DBC0 the number of transfers still to be performed minus one. The transfers stop when the count goes negative, not when reaching zero. The routine sets register DTDR0 to the trigger-factor input parameter and returns to the caller.

4.3.1.4 CSIB5_SendDataBlock

On entry, CSIB5_SendDataBlock calls DMA1_Stop() to disable channel 1 operation, then initialize the control counters and flags. The function next calls DMA1_Setup_mmc to initialize the DMA1 source and destination addresses, transfer count and trigger factor. Remember that the transfer ends when the count decrements to negative, not zero. At that point the program calls DMA1_Start() to enable the DMA completion interrupt. On completion of all data transfers, the routine calls interrupt-service function MD_INTDMA1() and sets the CSIB5_send_done flag.

CSIB5_SendDataBlock then enables the CSIB5 transmit interrupt and sets the interrupt priority to six. The routine sets the start trigger in register DHC1 to begin the first transfer. From then on, after sending a word of data, the CSIB5 transmit-complete interrupt (vector 0x39) triggers the DMA controller to load the next word into the CSIB5 transmit register. The routine then returns control to the caller.

4.3.1.5 DMA1_Setup_mmc

On entry, DMA1_Setup_mmc calls DMA1_Halt to stop DMA1 controller operation. Then DMA1_Setup_mmc calls function DMA1_source_addr with the parameters of the source address (which is the transmit data buffer address) and the mode parameter of DMA_INCR (i.e., the source increments after each transfer).

On return, the function calls DMA1_destination_addr with the destination address of 0xfffffd56, the SPI transmit register CB5TX. The mode parameter is DMA_FIXED, to indicate no address change after every transfer to the transmit register. On return, the function sets in register DBC1 the number of transfers to be performed minus one. The transfers stop when the count goes negative, not when reaching zero. The function sets register DTDR1 to the trigger-factor input parameter and returns to the caller.

This function sets DMA1 to increment the source address and fixed destination address, which is the CSIB5 transmit register and 16-bit transfer data. Trigger-factor register DTFR1 is set to the input parameter factor, which is the transmit interrupt vector 0x39. Thus, a transmit interrupt causes the DMA controller to load the transmit register with the next data word from the buffer.

The console display and buffer contents are as follows:

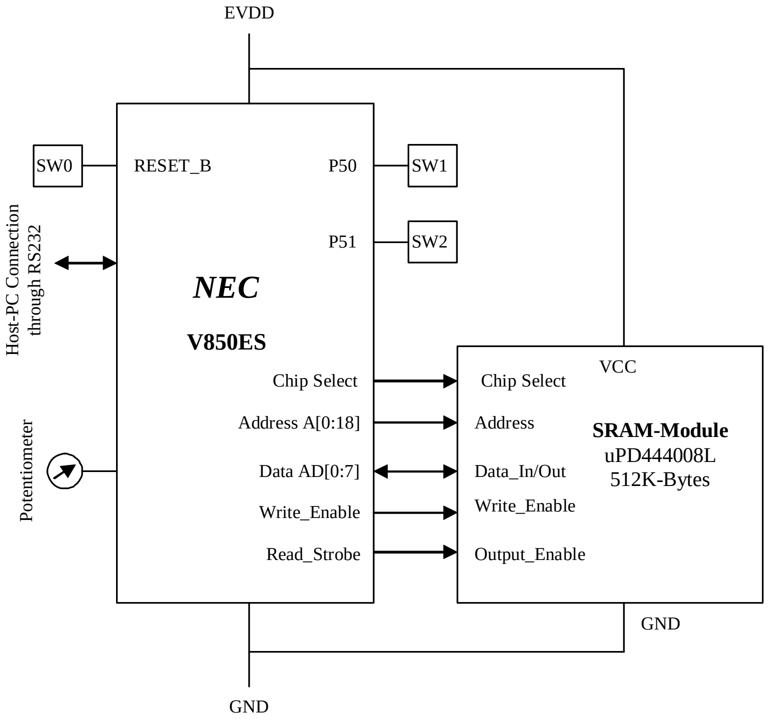
```
DMA Demonstration program (using interrupt I/O)
Step 1 done (retry 1) now in idle mode
Step 2 done (retry 2) now in SPI mode
SDmemory_init done
read status after init 0x00 card status 0x0000

 demonstrate read and write memory via DMA interface
 1 - Internal Memory -> Peripheral I/O (Memory Card)
 2 - Internal Memory -> External Memory (SRAM module)
 3 - External Memory -> External Memory
 4 - Peripheral I/O (Memory Card) -> Internal Memory
 5 - Peripheral I/O (A/D converter) -> Internal Memory
 6 - fill buffer with test data
 7 - fill buffer with random data
 8 - dump memory
 enter number of operation to perform - 7
random data
buffer1

fe20 d3e2 8594 e25f 032d 5edc 3e55 7572
f070 6e65 88ed ce10 a50e 2d91 5178 d535
fd7a 20a2 411d 8ec7 50f0 a884 3c90 9818
eff9 854e e590 b067 aaec 50c6 1768 2d5e
4c69 ffdc 6c70 8493 1cde 632b b98a c309
5b05 ab7e 8ea8 18af ca62 e047 b843 49de
24c7 6827 c0e3 3bdc 9ed1 866b 6b9c 6e5d
6d6b d68b 3a8d 7dff 3e48 d4ab ea61 a0cc
bc6b 541c f4d0 2cba 10a1 09da 0c1d 0f2c
5b03 000d a397 5770 3c76 248b 671b ab41
4e2d ba52 bf8e ce44 a824 e510 5265 218d
5ea5 219e 801f 1f1a fdc5 c97e e5c7 e055
10e5 9362 db75 9994 a034 11a7 f6cc 1999
ae29 30d4 887c 4c14 b90b bb1d 1dbf ba1f
3c6c d6e4 fede 06bf 2ea7 8535 b1ab 7168
8268 2547 7506 5778 a822 f000 c75a afb8
0898 7c70 e21f 8de0 8c57 3a53 3959 a111
1439 f991 fd15 b75b 01e1 63bf 9cf1 3738
ae43 7c9d 3d92 a50d f01b 2898 472d 20ae
9b73 a448 d902 e4d8 fe21 09f1 52da cbb6
6444 cf04 c98e c65f 93cc 469d 9381 6654
4ff1 1c05 c223 5705 d6b8 dc2f a26f e24b
6373 6c3d 3c6b 69ed ad40 8cfa d5ad ec1e
6788 5c5f 6ed2 88fb c080 d411 4406 f40f
e3a9 4bdf 4f82 05d1 7651 f346 c507 2922
```

```
1d11 59ef 9666 eed1 f64f e82e eff9 7a1a
1cbc 6583 ba29 1221 26d6 721b 1be9 9679
a865 0d4d f237 01a1 afff 101f 5da0 ec83
4686 b1c1 35ba 09f5 f5a7 020f 8ea9 aa3a
3f5b 7011 3a9d 3a81 2268 457c 4551 c063
99de 2830 788c 6266 1b9c 9aba d8a1 de7f
1bcb 79d2 26f1 1089 8860 7fdc 5f66 71d1
4dff ffff

 demonstrate read and write memory via DMA interface
 1 - Internal Memory -> Peripheral I/O (Memory Card)
2 - Internal Memory -> External Memory (SRAM module)
 3 - External Memory -> External Memory
 4 - Peripheral I/O (Memory Card) -> Internal Memory
 5 - Peripheral I/O (A/D converter) -> Internal Memory
 6 - fill buffer with test data
 7 - fill buffer with random data
 8 - dump memory
 enter number of operation to perform - 1
 enter sector number to use (00-99) 11

DR_message 0x05
data accepted

SDReadSector from addr 0x00001600 (sector 11) into buffer 0x03ff988c

 after sector write & read

 *** buffer data equal ***

 demonstrate read and write memory via DMA interface
 1 - Internal Memory -> Peripheral I/O (Memory Card)
 2 - Internal Memory -> External Memory (SRAM module)
 3 - External Memory -> External Memory
 4 - Peripheral I/O (Memory Card) -> Internal Memory
 5 - Peripheral I/O (A/D converter) -> Internal Memory
 6 - fill buffer with test data
 7 - fill buffer with random data
 8 - dump memory
 enter number of operation to perform -
```

4.3.2 DMA Transfers from Internal to External Memory

To demonstrate DMA transfers from internal memory to external memory, you need to create a text file on the host PC. This demonstration writes the text file into the SRAM module (external memory) using DMA controller functions. The microcontroller then reads from the SRAM module and displays the text on the host PC monitor.

Figure 7. DMA Transfers from Internal to External Memory



The SRAM module interface signals connect to the external-memory-bus control signals of the V850ES Series microcontroller. The microcontroller connects to a host PC through an RS-232 (UART) interface, using a terminal-emulation program running on the host PC.

You request this operation by entering “2” in response to the menu. This operation uses the function mem_to_mem, which transfers the text with DMA controller 0. On entry, the function stops DMA controller 0 by calling DMA0_Stop. The routine sets the transfer size in register DBC0, then sets the source and destination addresses, both incrementing. No trigger operation is required.

The routine calls function DMA0_Start to enable DMA controller operation. Finally, the function sets the STGn bit in register DCHC0 to ONE, starting the transfer. The function enters a loop to wait for completion of the transfer.

4.3.3 DMA Transfers from External to External Memory

The demonstration shows how to move a data block from one location in the SRAM module to another location in the module, using the same functions as the previous example. You request this operation by responding to the menu prompt with “3.”

4.3.4 DMA Transfers from Peripheral I/O (SPI) to External Memory

This example demonstrates moving a block of data from an SD memory card to an SRAM module using DMA transfers. The program writes a text file to the memory card, then moves the data to the SRAM module. The program then reads the data block from the SRAM module and displays the data on the PC monitor.

You select this function by entering “4” in response to the menu prompt. Then the function mmc_to_mem is called with parameters of the sector number to read, and the address of buffer2 for storing the data.

Function mmc_to_mem stops both DMA channels, then calls SDReadSector with the passed parameters, sector number and buffer address. On return, the function checks for errors, then stops both DMA channels and returns to the main program loop.

4.3.4.1 Function SDReadSector

The SDReadSector function requests a read of a sector from the SD memory card into the specified buffer. The function selects the device, builds command CMD17, and sends the command with the specified block address. The block address is the sector number multiplied by 512.

The card responds with an R1 message. If this response does not indicate an error, the SDReadSector function begins sending pad characters and looks for a data token. The data token indicates that the start of data follows. After receiving the data token, the function switches to 16-bit mode by calling CSIB5_polled_mode.

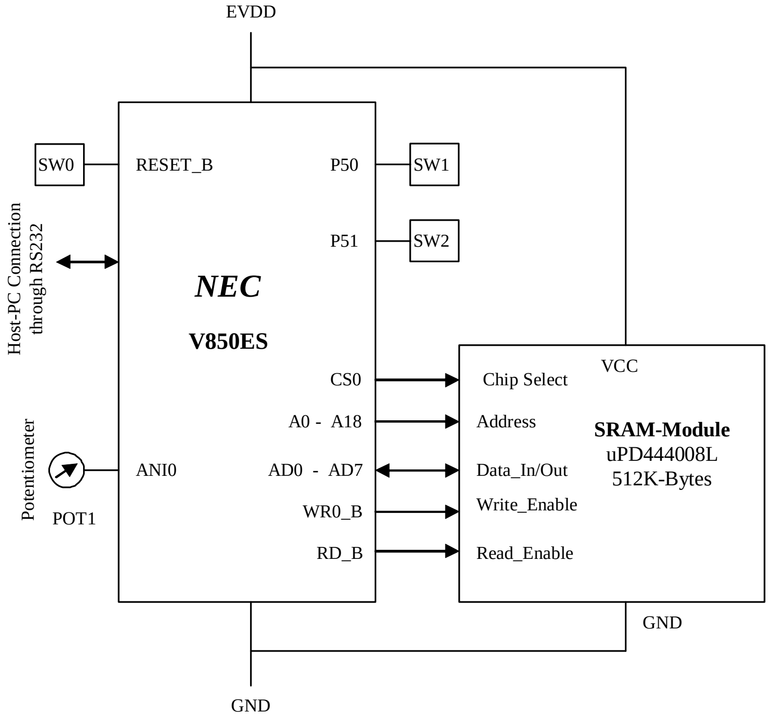
The function then calls CSIB5_ReceiveDataBlock, passing the pointer to the buffer and the number of bytes to read, which is 516. On return, the function calls CSIB5_SendDataBlock to send the 516 bytes that clock in the data. The routine then switches back to 8-bit mode by calling CSIB5_polled mode and sends NEC count pad characters. The routine deselects the device and returns the status of the read-sector request.

4.3.5 DMA Transfers from Peripheral I/O (A/D Converter) to Internal Memory

This example demonstrates moving data from the A/D converter, after conversion, to internal memory. The contents of internal memory can be read to the host PC for display. You select this operation by entering “5” at the menu prompt.

A potentiometer (POT1) on the AF-EV850ES/JJ2 evaluation board provides values for A/D conversion as you turn POT1 up or down. The potentiometer's tap connects to ANI0 of the V850ES/JJ2 microcontroller. As the A/D converter provides the digital values, the routine moves the data to internal memory using DMA.

Figure 8.   Transferring A/D Data to Internal Memory



The program calls function ad_to_mem with the parameters for the number of samples to be taken and a pointer to the destination for storing the results. The function stops the A/D controller before making any changes. The program sets the conversion speed, and selects interval timer 2 trigger and input from channel 1.

Although this demonstration uses only controller 0, the function stops both DMA controllers. The function stores the number of samples to be taken in register DBC0, sets the source and destination addresses for DMA, and selects the trigger factor of the A/D completion interrupt INTAD.

The function enables the A/D controller by calling AD_Start function, and then DMA0_Start to enable DMA controller channel 0 operation. Finally, the routine starts interval timer 2 by calling TMP2_Start. The routine monitors the DMA completion bits and the number of timer ticks, to ensure that the DMA completes. Upon completion, the function stops the A/D converter, the interval timer 2 controller and DMA controller 0, and control is returned to the main loop of the program.

Below is an example of DMA transfer from the A/D converter to internal memory:

```
DMA Demonstration program (using interrupt I/O)
Step 1 done (retry 1) now in idle mode

Step 2 done (retry 2) now in SPI mode
SDmemory_init done
read status after init 0x00 card status 0x0000

 demonstrate read and write memory via DMA interface
 1 - Internal Memory -> Peripheral I/O (Memory Card)
 2 - Internal Memory -> External Memory (SRAM module)
 3 - External Memory -> External Memory
 4 - Peripheral I/O (Memory Card) -> Internal Memory
 5 - Peripheral I/O (A/D converter) -> Internal Memory
 6 - fill buffer with test data
 7 - fill buffer with random data
 8 - dump memory
 enter number of operation to perform - 5

takes 10 seconds

95c0 9540 9500 9500 9540 9380 92c0 90c0
8dc0 8d80 8b40 8700 8580 8380 8380 81c0
8040 7c80 7bc0 7bc0 7940 77c0 7980 7640
7500 73c0 7200 6f40 6e40 6b80 6b80 6a80
6b00 6ac0 6b40 6b40 6b80 6b40 6b00 6b40
6b40 6b40 6b40 6b40 6b40 6b40 6b40 6b00
6b00 6b00 6ac0 6b00 6b40 6b40 6b40 6b40
6b40 6b40 6b40 6b00 6b40 6b00 6b00 6ac0
6ac0 6a80 6ac0 6ac0 6ac0 6ac0 6a80 6ac0
6a80 6ac0 6ac0 6a80 6ac0 6b00 6b00 6b00
6ac0 6ac0 6ac0 6b00 6b40 6b40 6b40 6b40
6b40 6b40 6b40 6b40 6b40 6b40 6b80 6bc0
6c80 6e40 6fc0 7100 7480 7900 7bc0 81c0
8340 8440 8700 8b00 8c40 8d00 9240 9640
96c0 96c0 9d00 9dc0 9dc0 9e00 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9d80 9dc0 9dc0
9dc0 9d80 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9d80 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9d80 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0
9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0 9dc0

 demonstrate read and write memory via DMA interface
```

```
1 - Internal Memory -> Peripheral I/O (Memory Card)
2 - Internal Memory -> External Memory (SRAM module)
3 - External Memory -> External Memory
4 - Peripheral I/O (Memory Card) -> Internal Memory
5 - Peripheral I/O (A/D converter) -> Internal Memory
6 - fill buffer with test data
7 - fill buffer with random data
8 - dump memory
enter number of operation to perform -
```

In this example, the potentiometer was moved back and forth randomly to show that different readings were taken.

4.3.6 Operator Selection of Test Data

4.3.6.1 Operator-Entered Data

Selecting “6” in response to the menu prompt allows you to enter 516 characters into buffer1. You must enter all the data. The software appends a start token to the buffer.

4.3.6.2 PC-Generated Data

If you select “7” in response to the menu prompt, the PC fills buffer 1 with 516 characters of random data. The software appends a start token to the buffer.

4.3.7 Operator-Requested Data Dump

If you select “8” in response to the menu prompt, the program calls function dump_buffers to dump buffer1, buffer2, ebuffer1 and ebuffer2 to the PC display.

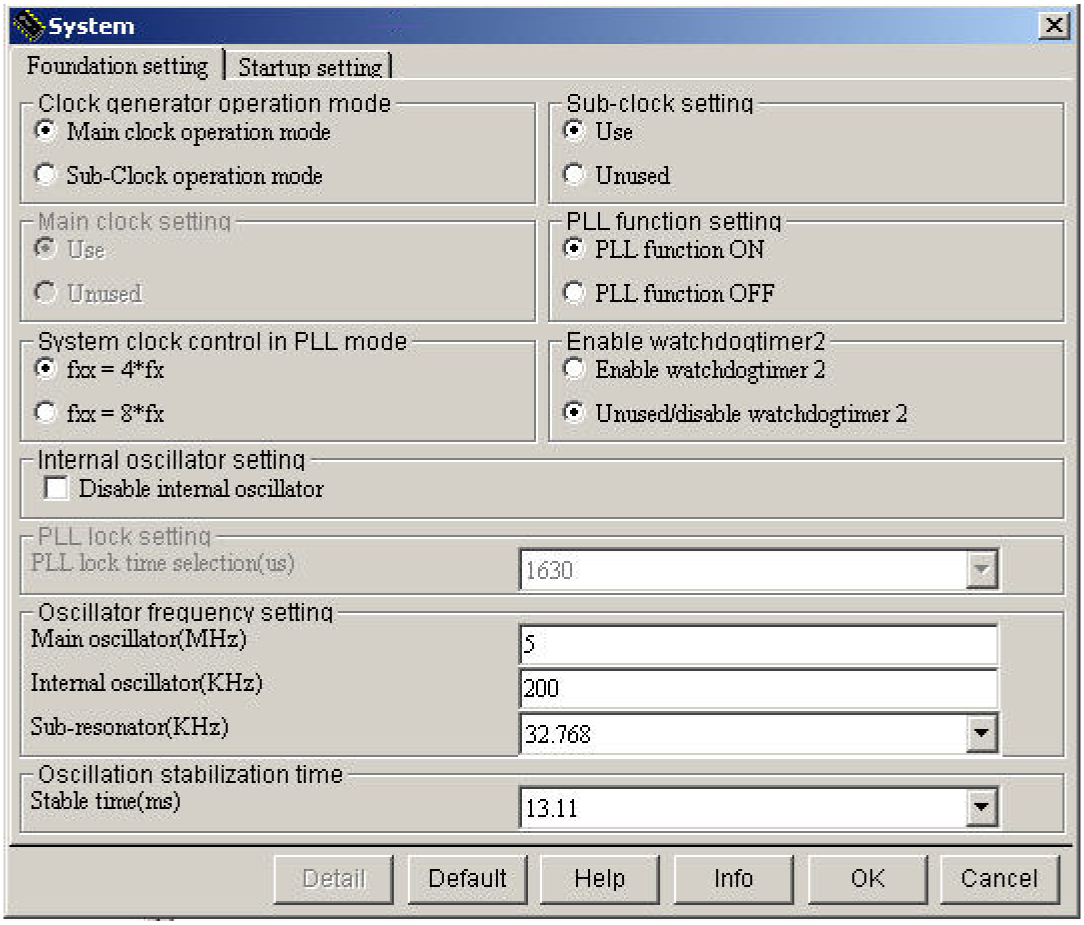
5. Applilet Selections

The Applilet tool helps you build a basic code framework to begin a development project. Using a graphical user interface, you select from a menu of options. On completion, the Applilet generates the applicable code. Save these files and modify a copy of them, then if you need to go back and use the Applilet to modify an option or add a new feature, the tool will not overwrite your modified application code.

In this demonstration, the following selections are made for the initial generation of code using the Applilet tool.

Chose main clock operation (see the dialog box below), because the demonstration board provides an external 5-MHz crystal for use as a system clock. Having the PLL function ON multiplies the 5-MHz clock (fxx = 4*fx), which provides a 20-MHz system clock. The demonstration does not use watchdogtimer 2.

Figure 9. 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

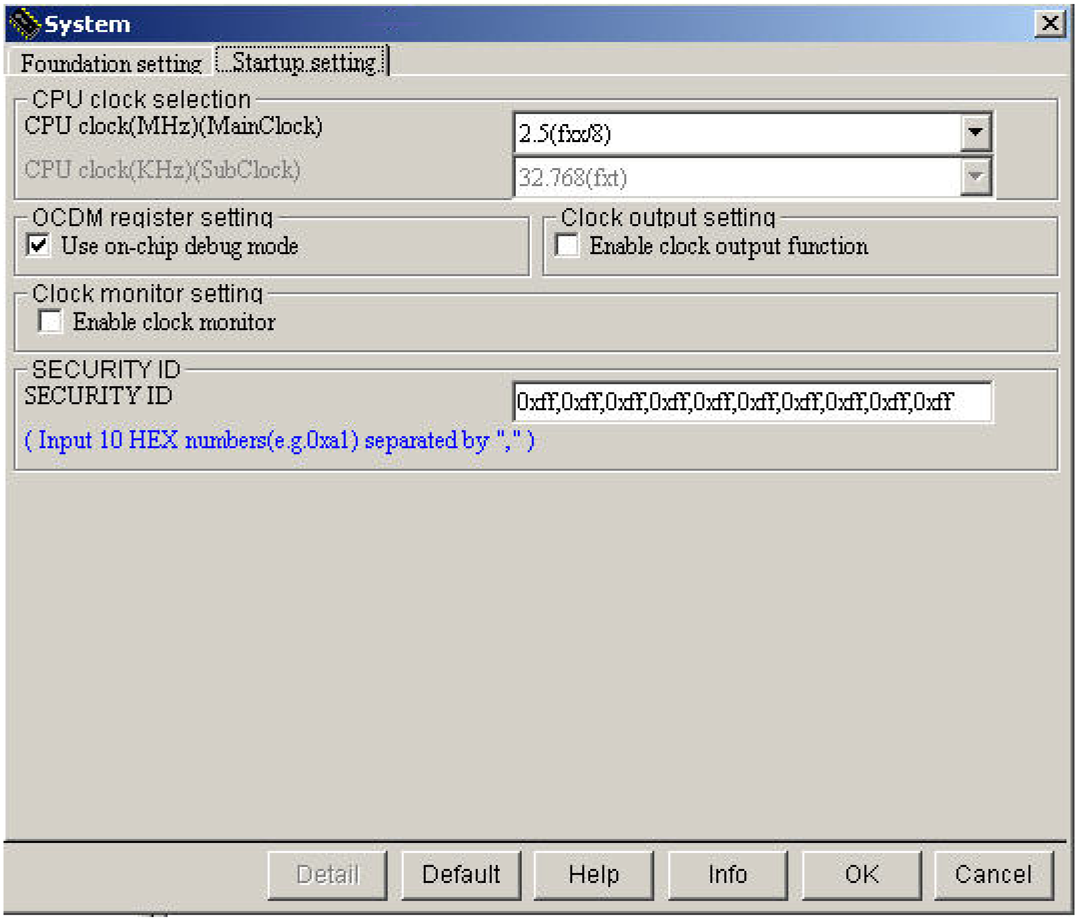
PLL lock setting
PLL lock time selection(us) 1630

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

Oscillation stabilization time
Stable time(ms) 13.11

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

Figure 10. System Startup Settings



CPU clock selection

CPU clock(MHz)(MainClock) 2.5(fxx/8)

CPU clock(KHz)(SubClock) 32.768(fxt)

OCDM register setting

☑ Use on-chip debug mode

Clock output setting

☐ Enable clock output function

Clock monitor setting

☐ Enable clock monitor

SECURITY ID

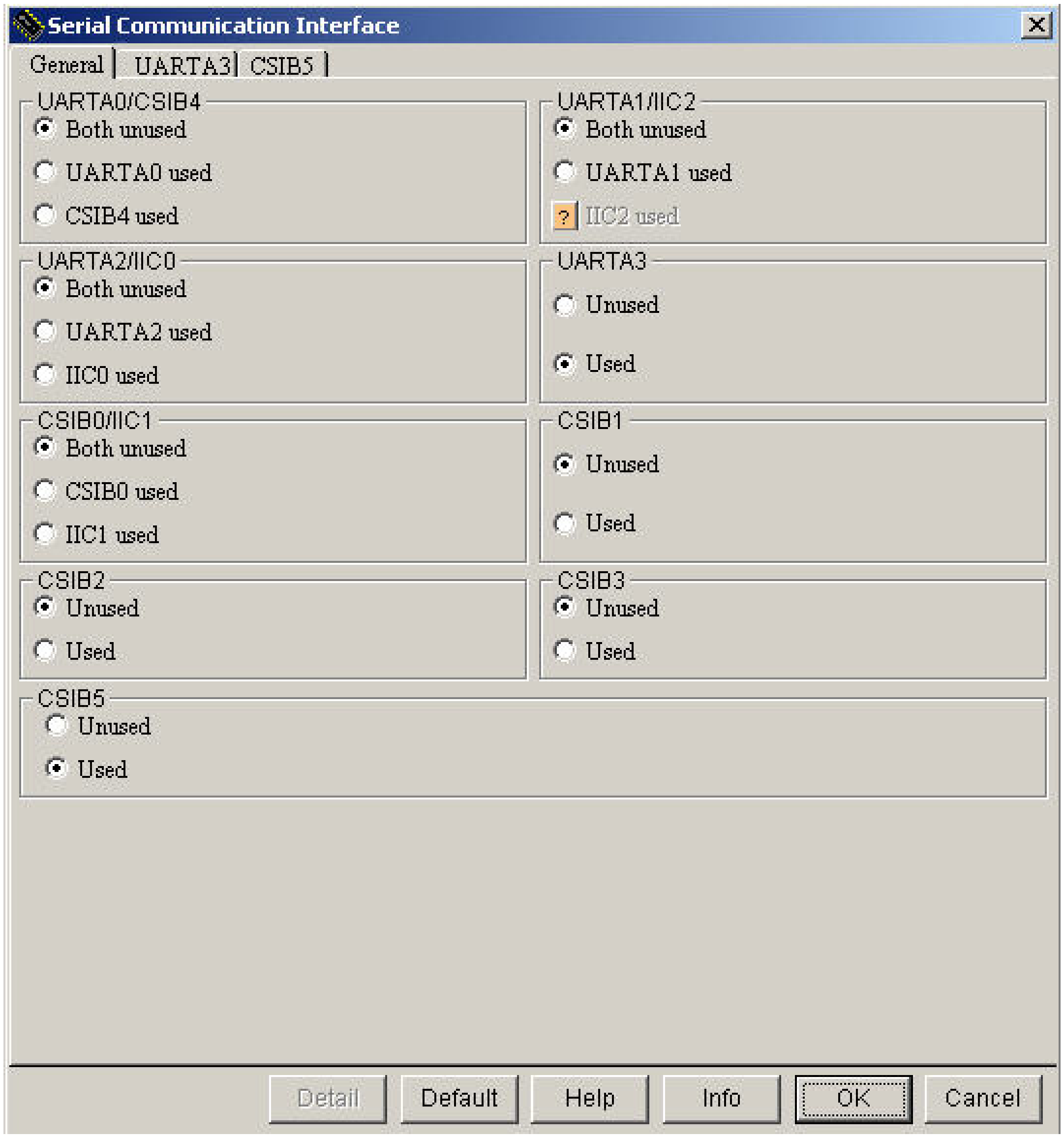
SECURITY ID 0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff

(Input 10 HEX numbers(e.g.0xa1) separated by ",")

5.1 Serial-Communications Interface Selections

The dialog box shown below lets you pick which serial devices to use. The demonstration uses UART3 and CSIB5.

Figure 11. General Settings for Serial Communication



Serial Communication Interface

General | UARTA3 | CSIB5

UARTA0/CSIB4
● Both unused
○ UARTA0 used
○ CSIB4 used

UARTA1/IIC2
● Both unused
○ UARTA1 used
? IIC2 used

UARTA2/IIC0
● Both unused
○ UARTA2 used
○ IIC0 used

UARTA3
○ Unused
● Used

CSIB0/IIC1
● Both unused
○ CSIB0 used
○ IIC1 used

CSIB1
● Unused
○ Used

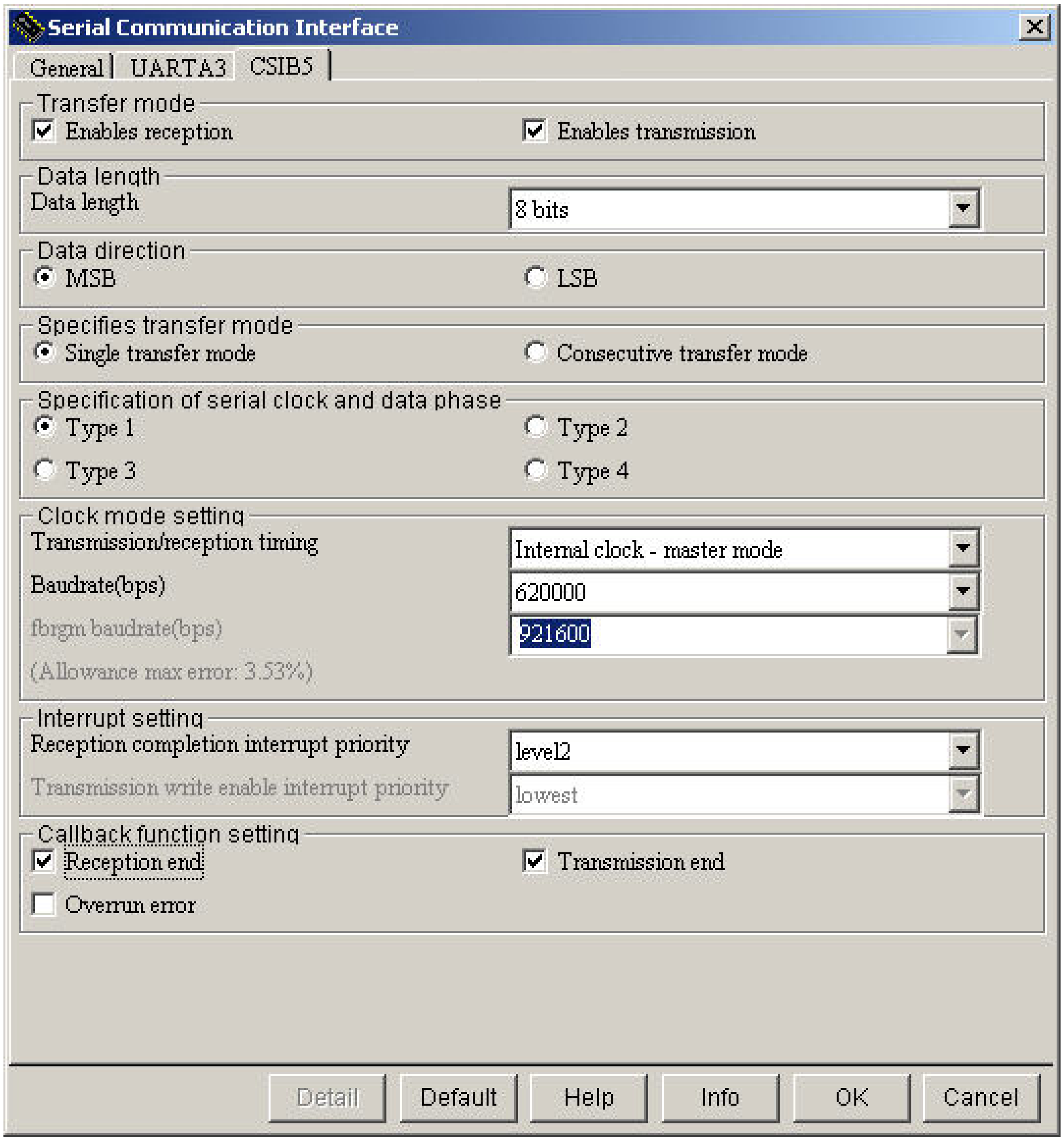
CSIB2
● Unused
○ Used

CSIB3
● Unused
○ Used

CSIB5
○ Unused
● Used

The next screen shows how to configure the Applilet for CSI. The demonstration involves sending and receiving data. The data format is 8 bits, MSB first, single transfers, Type 1 for SPI. The microcontroller is the master and supplies the clock. Using a slow speed avoids problems. Interrupts are used for both send and receive.

Figure 12. Serial Communications—CSIB5 Settings



Transfer mode
☑ Enables reception ☑ Enables transmission

Data length
Data length: 8 bits

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

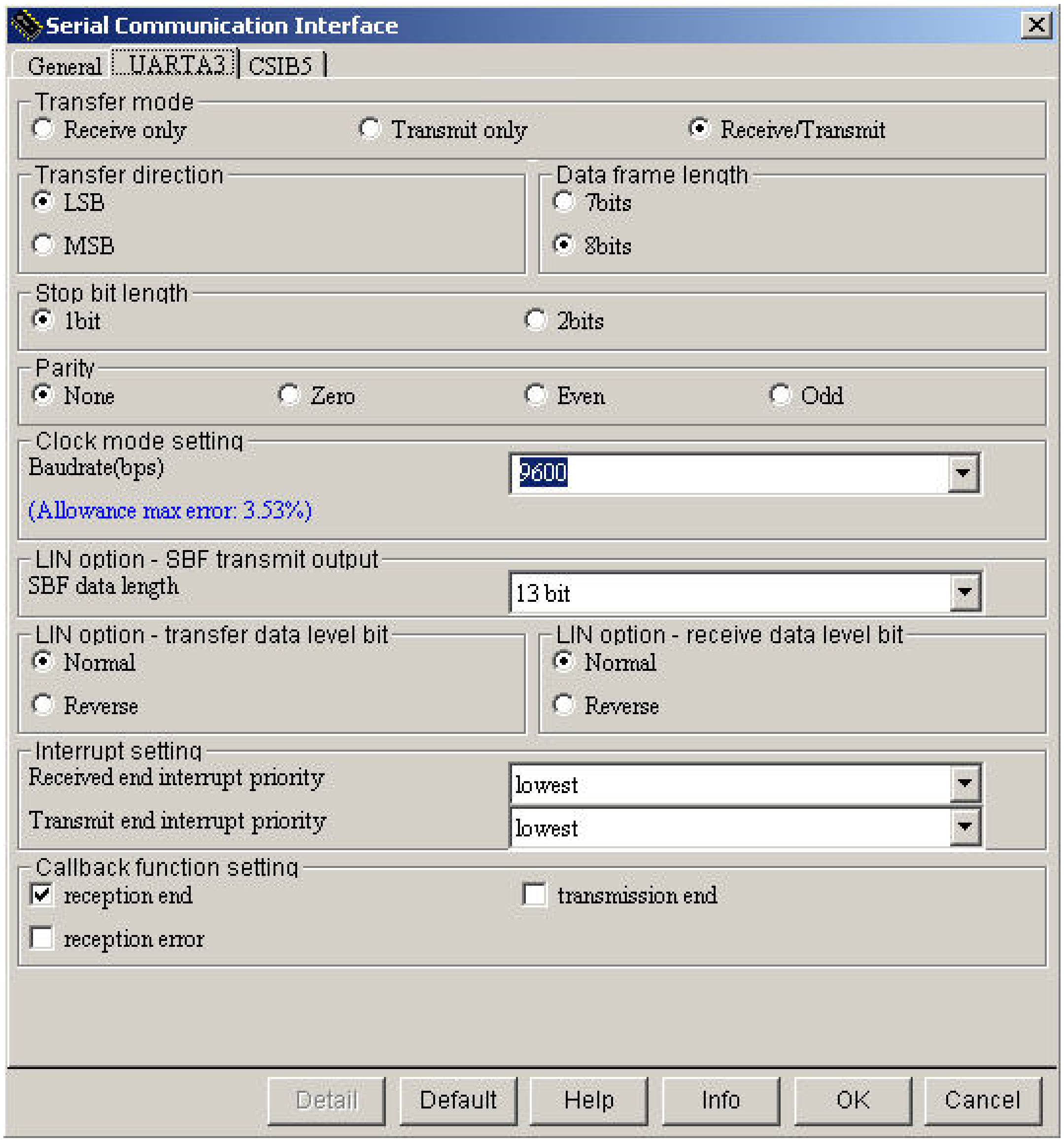
Clock mode setting
Transmission/reception timing: Internal clock - master mode
Baudrate(bps): 620000
fbrgm baudrate(bps): 921600
(Allowance max error: 3.53%)

Interrupt setting
Reception completion interrupt priority: level2
Transmission write enable interrupt priority: lowest

Callback function setting
☑ Reception end ☑ Transmission end
☐ Overrun error

The next dialog box configures the Applilet for UART3. The demonstration uses the normal asynchronous serial-communications settings. The software polls for completion on output but uses interrupt-driven input to supply the round-robin buffer.

Figure 13. *Serial-Communications Interface UARTA3*



Serial Communication Interface

Transfer mode
○ Receive only ○ Transmit only ● Receive/Transmit

Transfer direction
● LSB
○ MSB

Data frame length
○ 7bits
● 8bits

Stop bit length
● 1bit ○ 2bits

Parity
● None ○ Zero ○ Even ○ Odd

Clock mode setting
Baudrate(bps) 9600
(Allowance max error: 3.53%)

LIN option - SBF transmit output
SBF data length 13 bit

LIN option - transfer data level bit
● Normal
○ Reverse

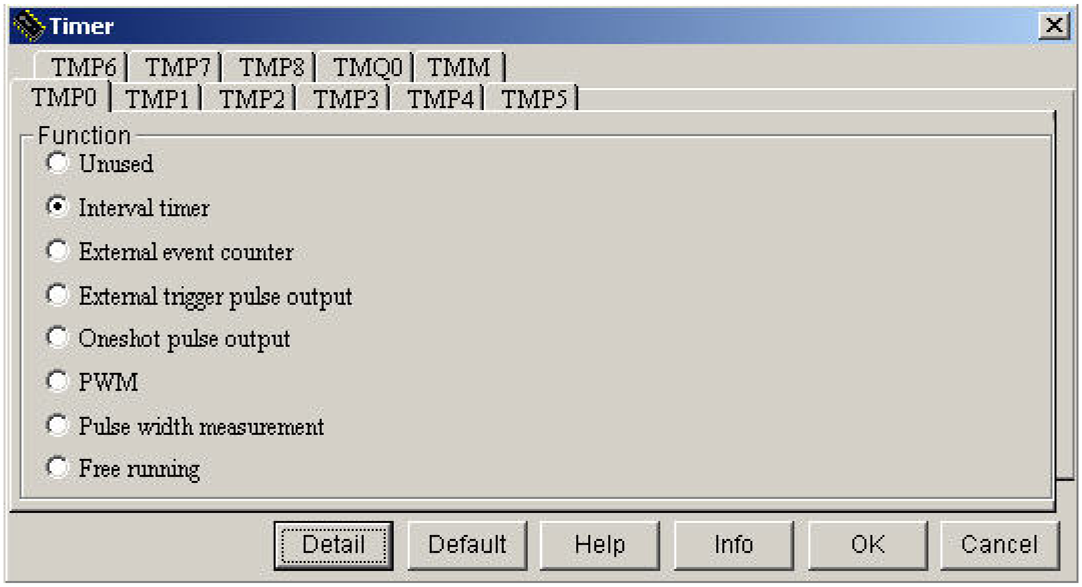
LIN option - receive data level bit
● Normal
○ Reverse

Interrupt setting
Received end interrupt priority lowest
Transmit end interrupt priority lowest

Callback function setting
☑ reception end ☐ transmission end
☐ reception error

The next dialog box configures the Applilet for Timer 00 (TMP0). The demonstration uses the 16-bit timer TMP0 as an interval timer for the delay function.

Figure 14. Configuring Timer 00.



Function

○ Unused

● Interval timer

○ External event counter

○ External trigger pulse output

○ Oneshot pulse output

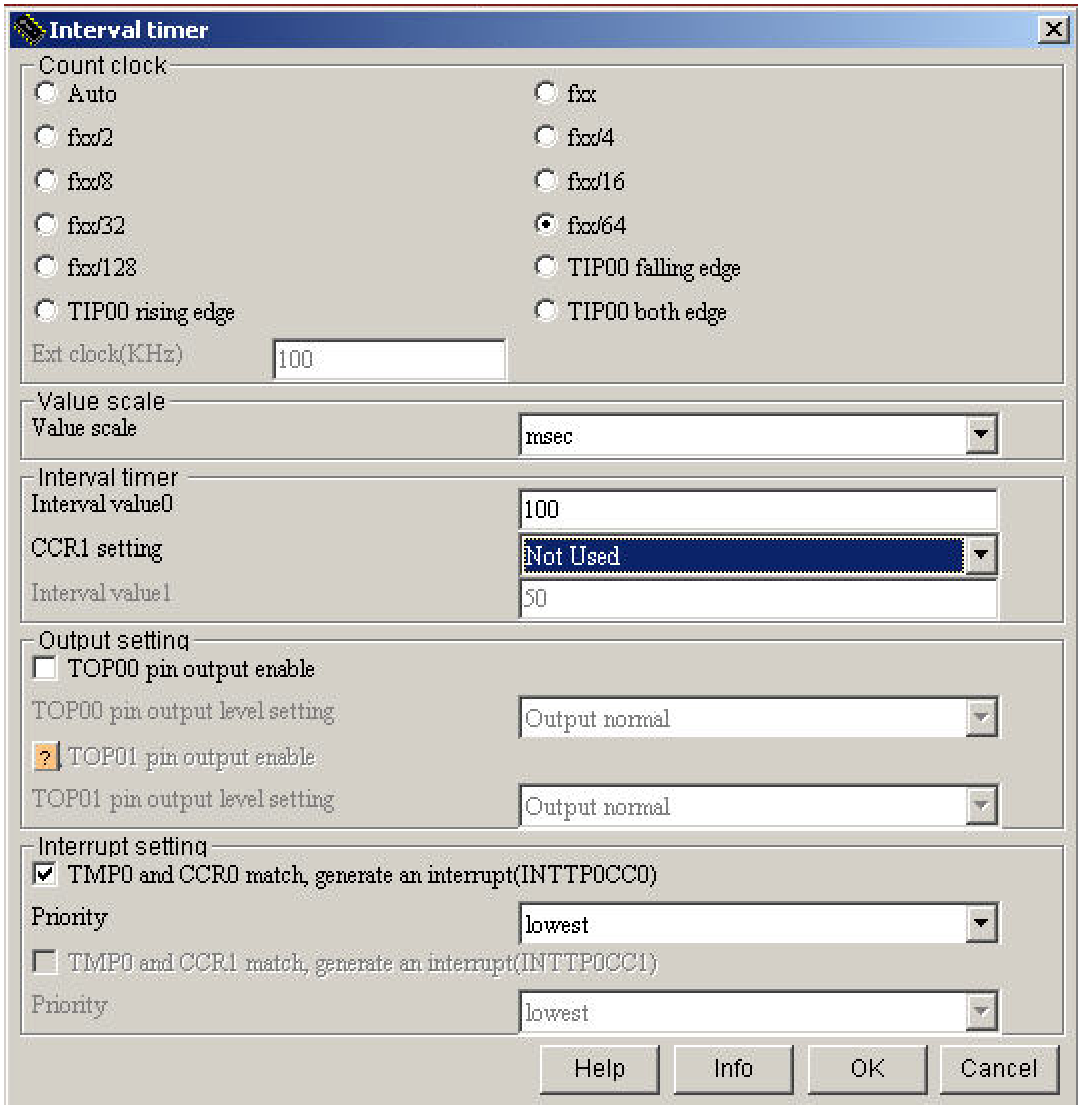
○ PWM

○ Pulse width measurement

○ Free running

The next screen shows how to set up the interval timer with a rather long interval. The interrupt can be the lowest priority, because these functions are the least important.

Figure 15. Setting Up Interval Timer



Interval timer

Count clock

○ Auto ○ fxx
○ fxx/2 ○ fxx/4
○ fxx/8 ○ fxx/16
○ fxx/32 ● fxx/64
○ fxx/128 ○ TIP00 falling edge
○ TIP00 rising 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)

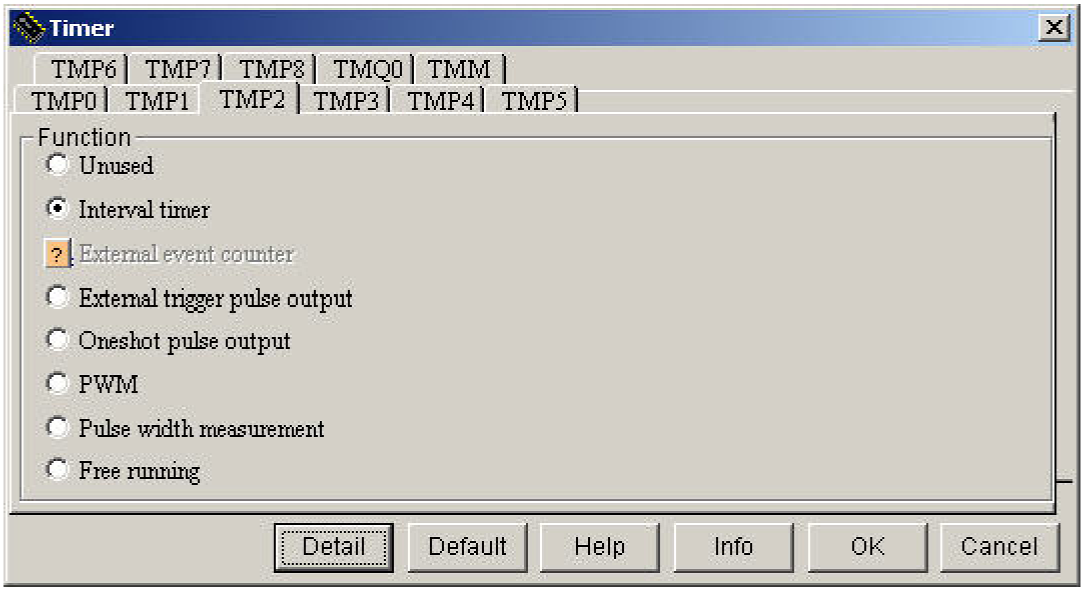
Priority: lowest

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

Priority: lowest

The demonstration requires another timer to drive the timed A/D conversions, so the next dialog box selects timer 2 as an interval timer to serve this interrupt.

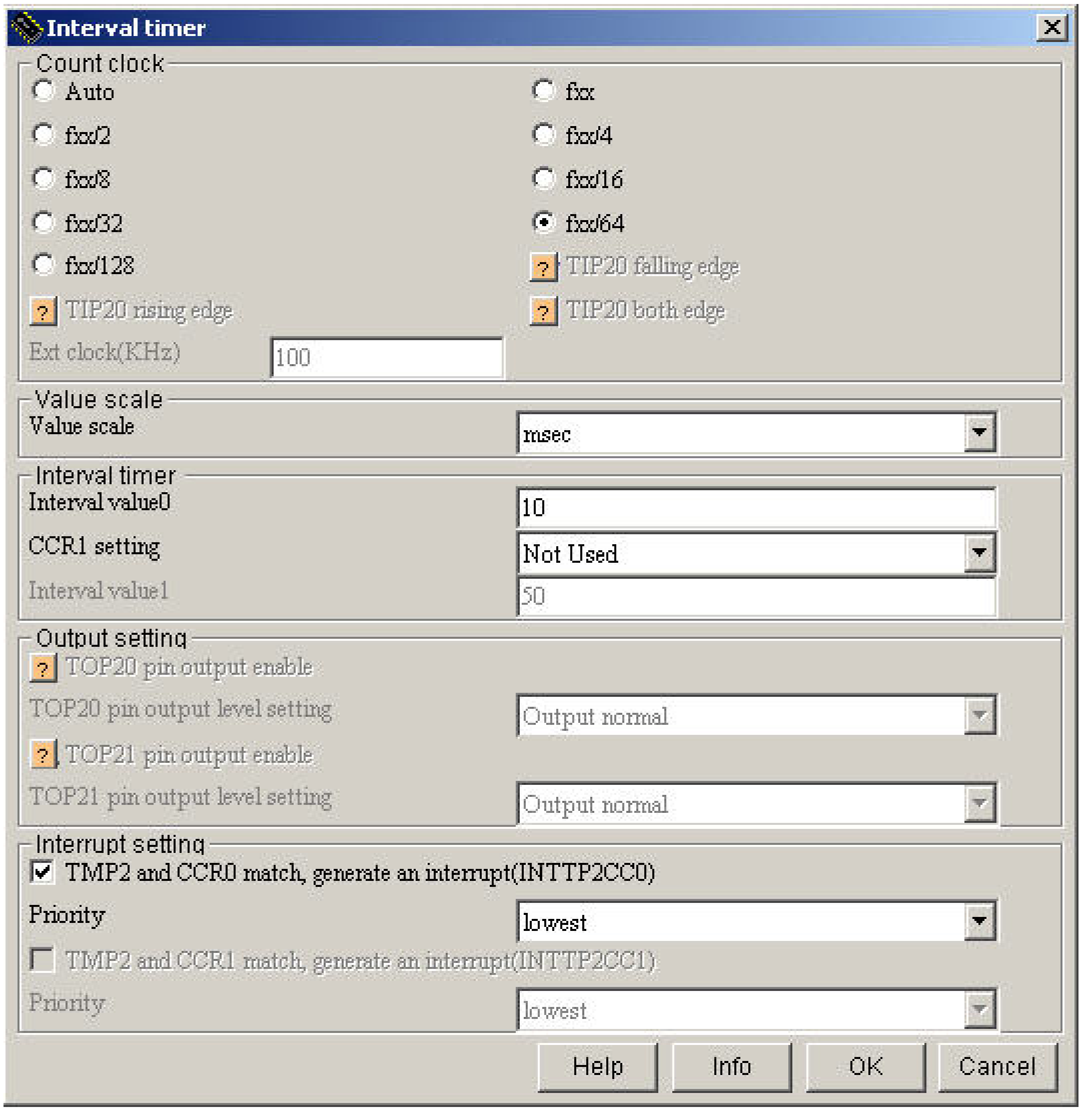
Figure 16. Setting Second Timer for A/D Conversions



Function
- ○ Unused
- ● Interval timer
- ? External event counter
- ○ External trigger pulse output
- ○ Oneshot pulse output
- ○ PWM
- ○ Pulse width measurement
- ○ Free running

The next Applilet dialog box provides the setup for the interval between analog samples. The chosen internal must allow sufficient time to take the sample. Make the time slow enough so that you can rotate the potentiometer in the time required to convert all of the samples.

Figure 17. Setting Interval Between Analog Samples



Interval timer

Count clock

○ Auto ○ fxx

○ fxx/2 ○ fxx/4

○ fxx/8 ○ fxx/16

○ fxx/32 ● fxx/64

○ fxx/128 ? TIP20 falling edge

? TIP20 rising edge ? TIP20 both edge

Ext clock(KHz) 100

Value scale

Value scale: msec

Interval timer

Interval value0: 10

CCR1 setting: Not Used

Interval value1: 50

Output setting

? TOP20 pin output enable

TOP20 pin output level setting: Output normal

? TOP21 pin output enable

TOP21 pin output level setting: Output normal

Interrupt setting

☑ TMP2 and CCR0 match, generate an interrupt(INTTP2CC0)

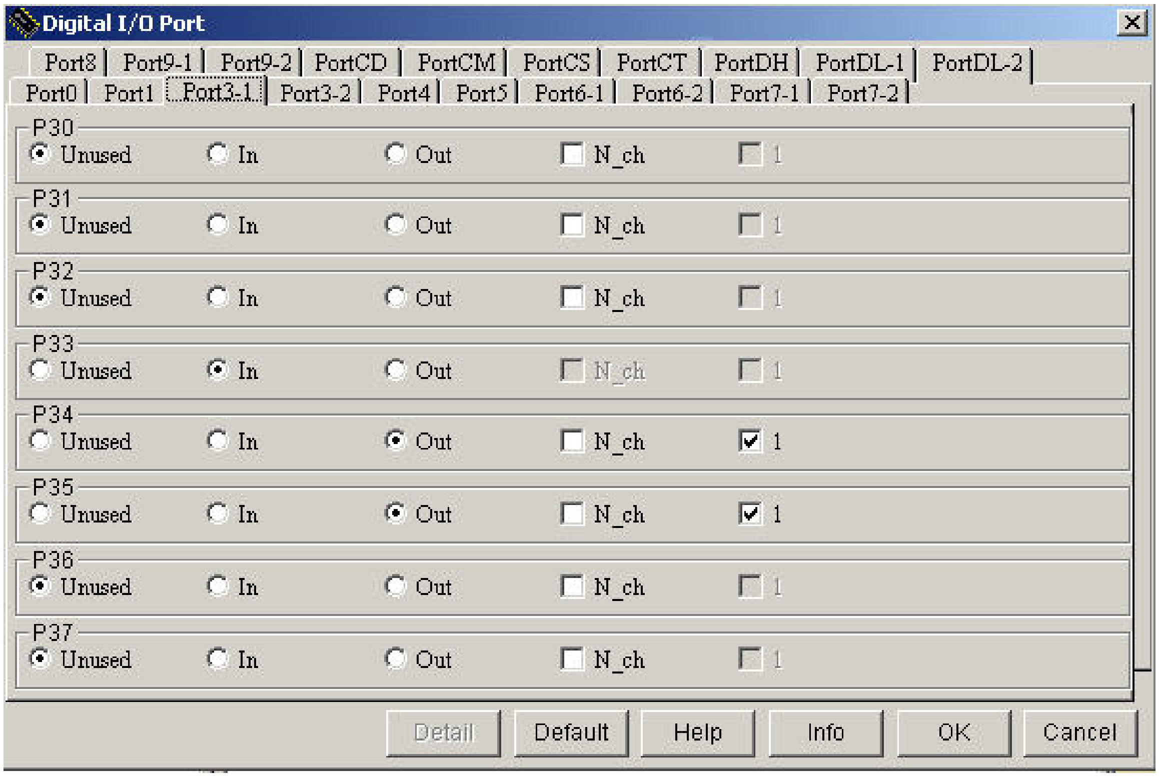
Priority: lowest

□ TMP2 and CCR1 match, generate an interrupt(INTTP2CC1)

Priority: lowest

Next, configure the Applilet for I/O ports. Port 3 bit 4 is the SPI chip select for Zigbee (the standard wireless interface), and Port 3 bit 5 is the chip select for the SD memory card. The default is set to HIGH as these are active-LOW selects.

Figure 18. Configuring I/O Ports


Digital I/O Port
Port8 | Port9-1 | Port9-2 | PortCD | PortCM | PortCS | PortCT | PortDH | PortDL-1 | PortDL-2
Port0 | Port1 | Port3-1 | Port3-2 | Port4 | Port5 | Port6-1 | Port6-2 | Port7-1 | Port7-2
Detail | Default | Help | Info | OK | Cancel

P30: ● Unused ○ In ○ Out □ N_ch □ 1

P31: ● Unused ○ In ○ Out □ N_ch □ 1

P32: ● Unused ○ In ○ Out □ N_ch □ 1

P33: ○ Unused ● In ○ Out □ N_ch □ 1

P34: ○ Unused ○ In ● Out □ N_ch ☑ 1

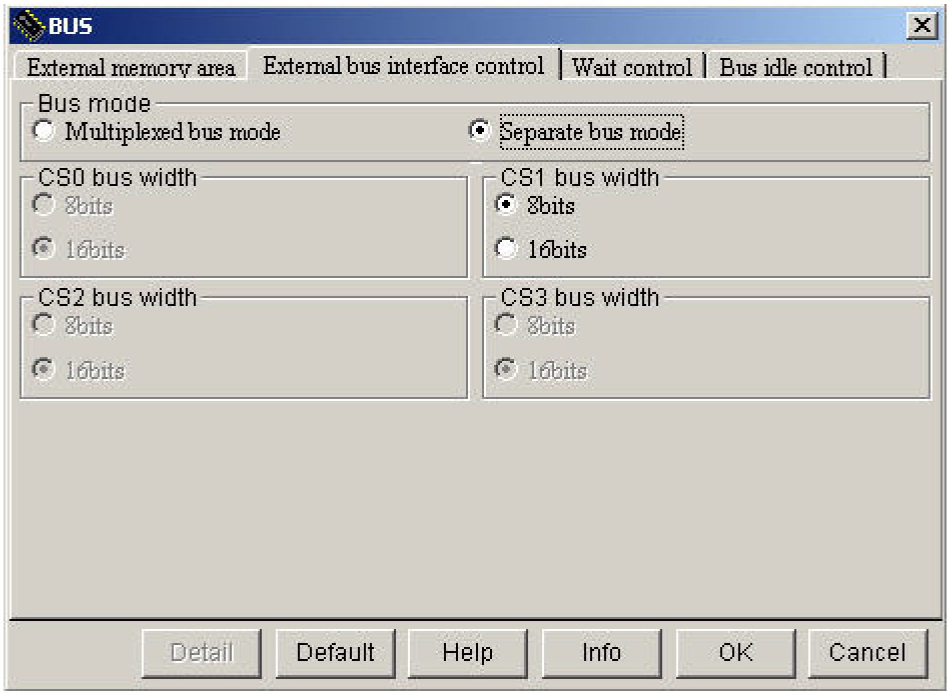
P35: ○ Unused ○ In ● Out □ N_ch ☑ 1

P36: ● Unused ○ In ○ Out □ N_ch □ 1

P37: ● Unused ○ In ○ Out □ N_ch □ 1

The external SRAM is an 8-bit-wide device. The demonstration does not include logic for a multiplexed bus, so choose the microcontroller's simpler separate address and data line bus mode, as shown below.

Figure 19. Configuring Bus



Bus mode
○ Multiplexed bus mode ● Separate bus mode

CS0 bus width
○ 8bits
● 16bits

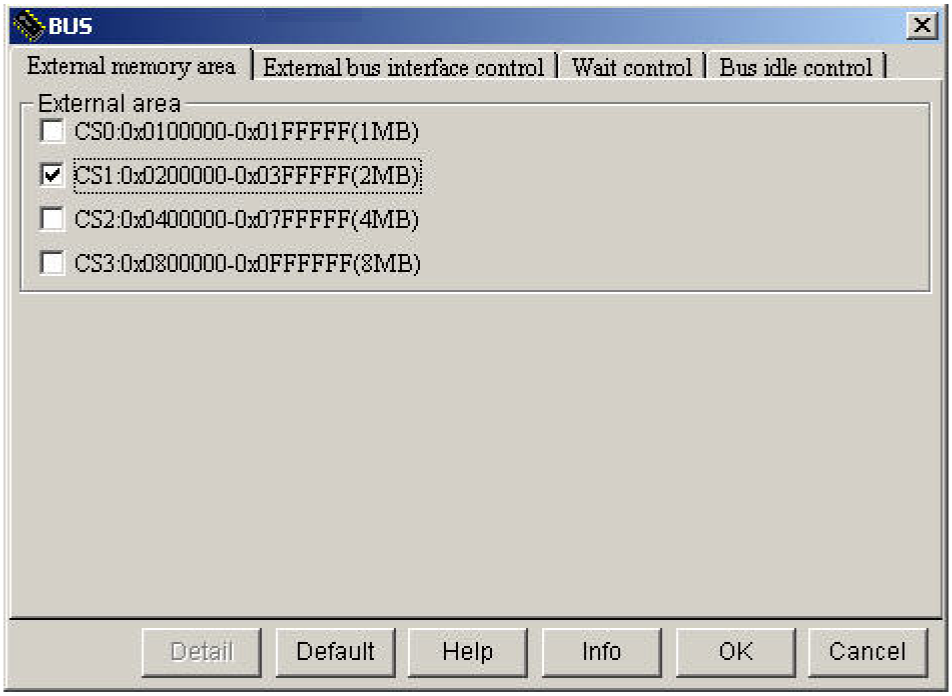
CS1 bus width
● 8bits
○ 16bits

CS2 bus width
○ 8bits
● 16bits

CS3 bus width
○ 8bits
● 16bits

The chip-select line for the external SRAM is wired to chip-select 1, so choose CS1 in the next dialog box.

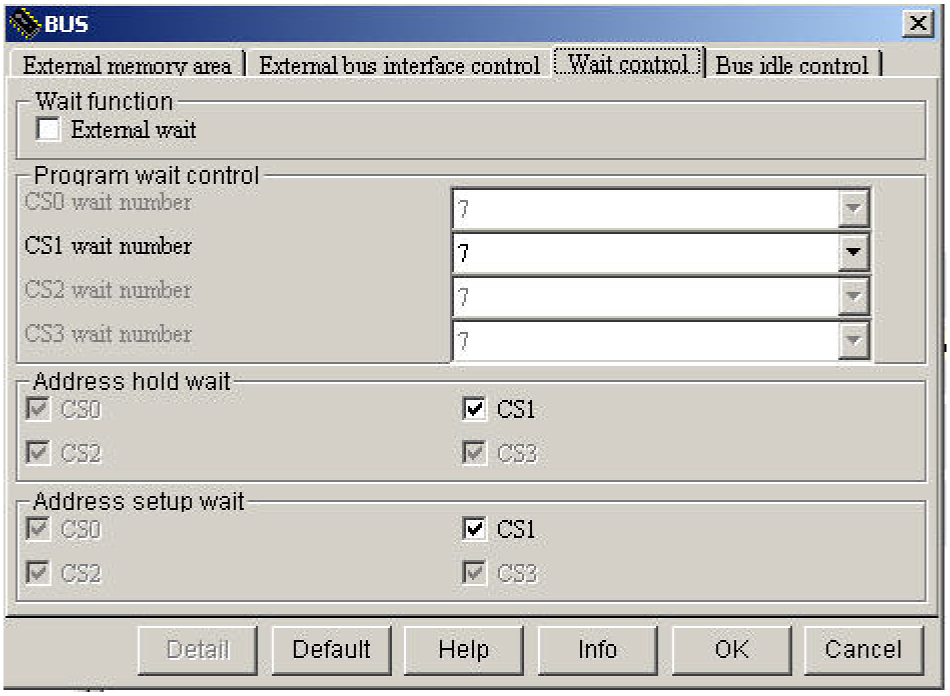
Figure 20. Defining Memory Area



External area
□ CS0:0x0100000-0x01FFFFF(1MB)
☑ CS1:0x0200000-0x03FFFFF(2MB)
□ CS2:0x0400000-0x07FFFFF(4MB)
□ CS3:0x0800000-0x0FFFFFF(8MB)

The wait states for CS1 are set as follows.

Figure 21. Setting Up Wait States for Chip Select CS1



Wait function

□ External wait

Program wait control

CS0 wait number 7

CS1 wait number 7

CS2 wait number 7

CS3 wait number 7

Address hold wait

☑ CS0 ☑ CS1

☑ CS2 ☑ CS3

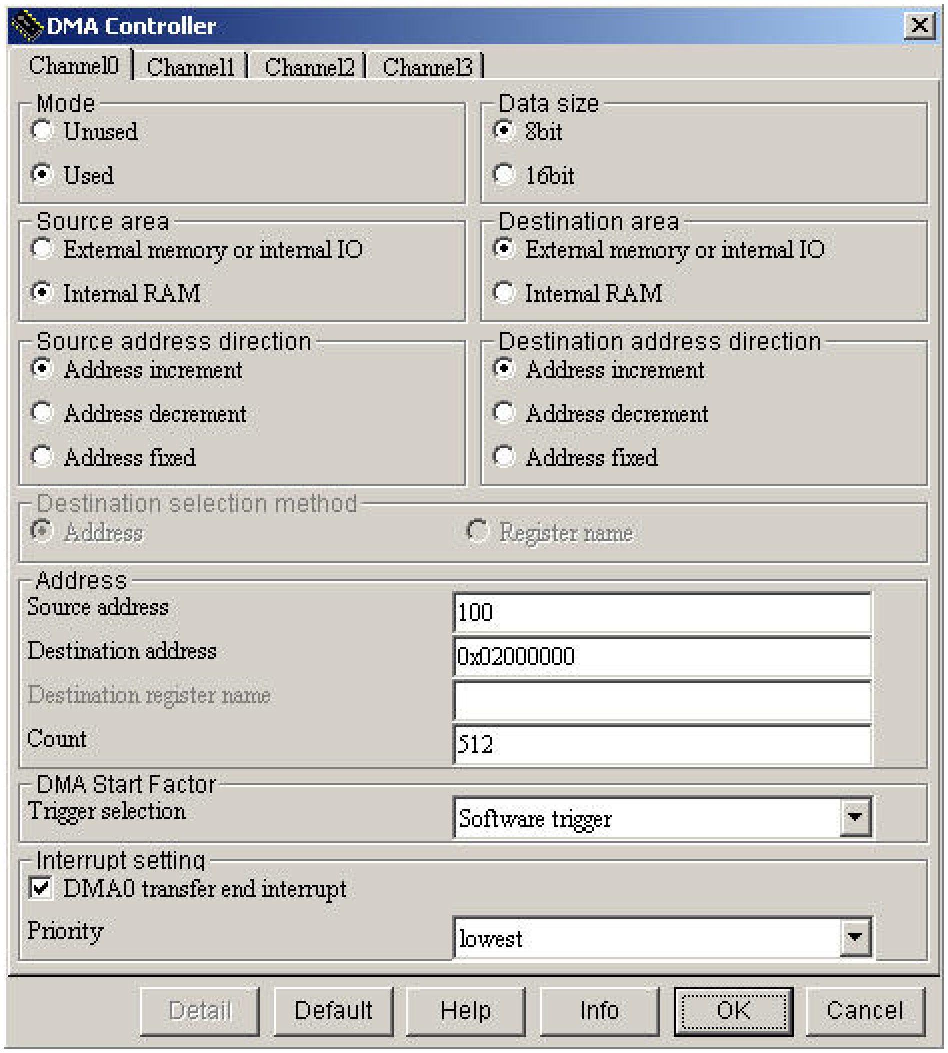
Address setup wait

☑ CS0 ☑ CS1

☑ CS2 ☑ CS3

The next dialog box only sets the basic functions to handle channel 0 of the DMA controller. The software modifies these settings to suit the function performed.

Figure 22. Configuring DMA Controller Channel 0


DMA Controller

Channel0 | Channel1 | Channel2 | Channel3

Mode
○ Unused
● Used

Data size
● 8bit
○ 16bit

Source area
○ External memory or internal IO
● Internal RAM

Destination area
● External memory or internal IO
○ Internal RAM

Source address direction
● Address increment
○ Address decrement
○ Address fixed

Destination address direction
● Address increment
○ Address decrement
○ Address fixed

Destination selection method
● Address
○ Register name

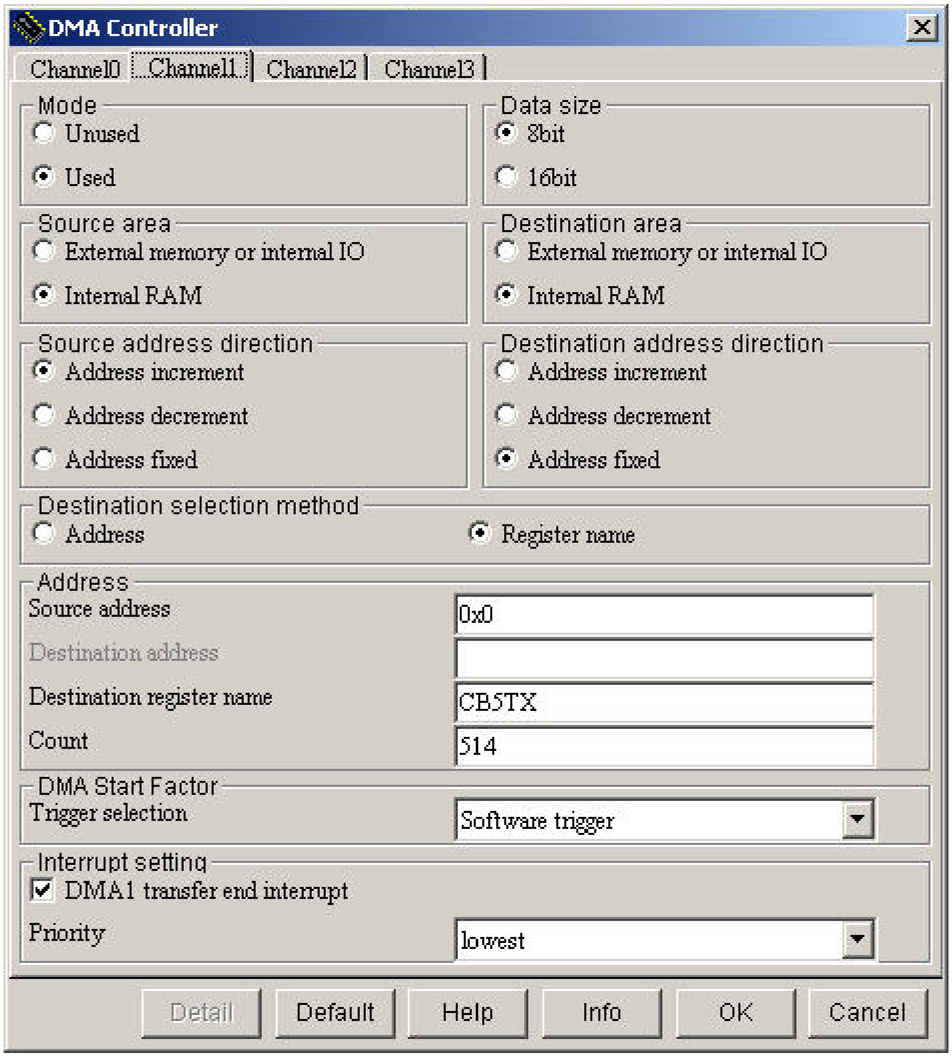
Address
Source address: 100
Destination address: 0x02000000
Destination register name:
Count: 512

DMA Start Factor
Trigger selection: Software trigger

Interrupt setting
☑ DMA0 transfer end interrupt
Priority: lowest

As with the previous dialog box, the next one simply sets the basic functions to handle channel 1 of the DMA controller. Software modifies these settings to suit the function performed.

Figure 23. Configuring DMA Controller Channel 1



Mode
○ Unused
● Used

Data size
● 8bit
○ 16bit

Source area
○ External memory or internal IO
● Internal RAM

Destination area
○ External memory or internal IO
● Internal RAM

Source address direction
● Address increment
○ Address decrement
○ Address fixed

Destination address direction
○ Address increment
○ Address decrement
● Address fixed

Destination selection method
○ Address ● Register name

Address

Source address	0x0
Destination address	
Destination register name	CB5TX
Count	514

DMA Start Factor

Trigger selection: Software trigger

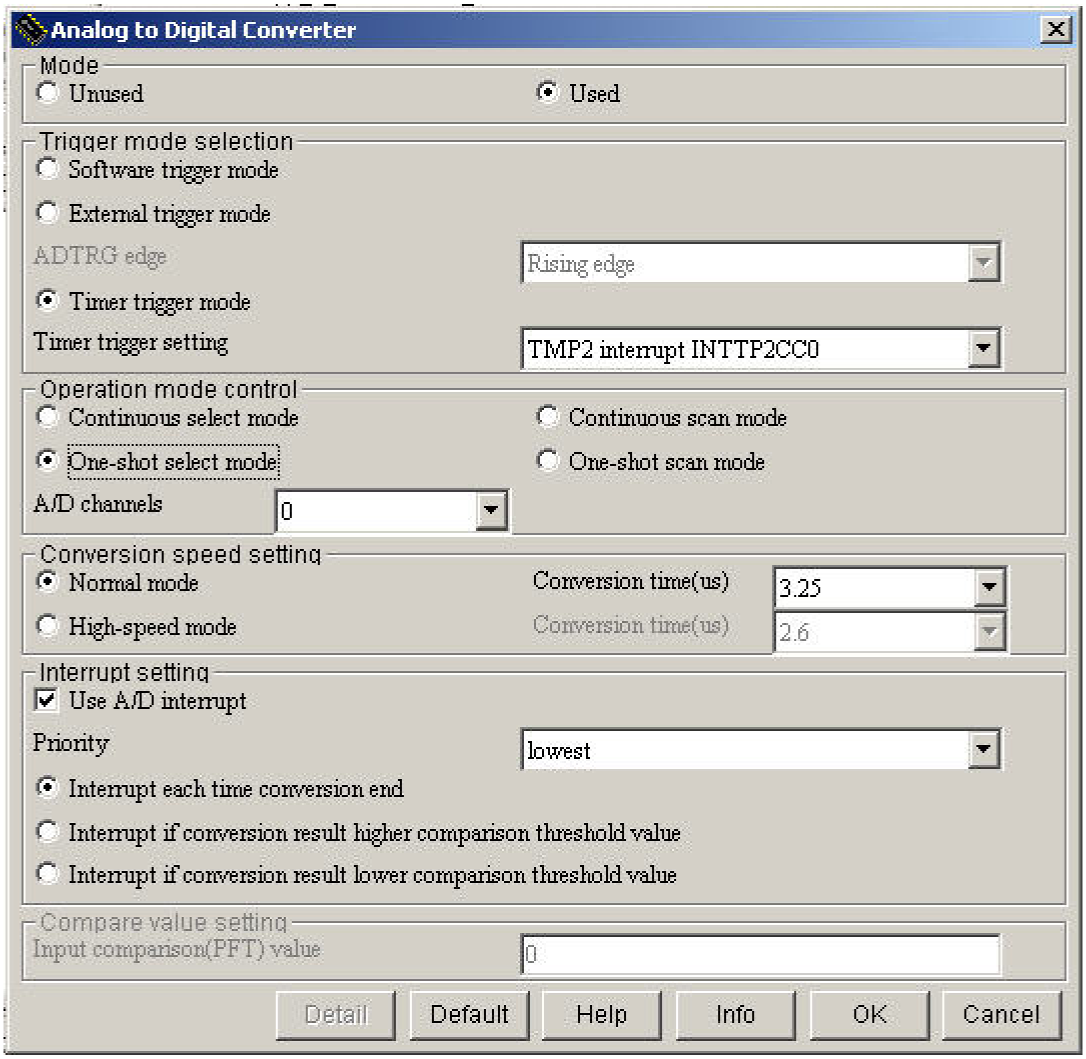
Interrupt setting

☑ DMA1 transfer end interrupt

Priority: lowest

The next selections connect interval timer 2 as the trigger to start A/D conversion. Completion of the conversion generates an interrupt.

Figure 24. Connecting Interval Timer as Conversion Trigger



Analog to Digital Converter

Mode
○ Unused ● Used

Trigger mode selection
○ Software trigger mode
○ External trigger mode
ADTRG edge: Rising edge
● Timer trigger mode
Timer trigger setting: TMP2 interrupt INTTP2CC0

Operation mode control
○ Continuous select mode ○ Continuous scan mode
● One-shot select mode ○ One-shot scan mode
A/D channels: 0

Conversion speed setting
● Normal mode Conversion time(us): 3.25
○ High-speed mode Conversion time(us): 2.6

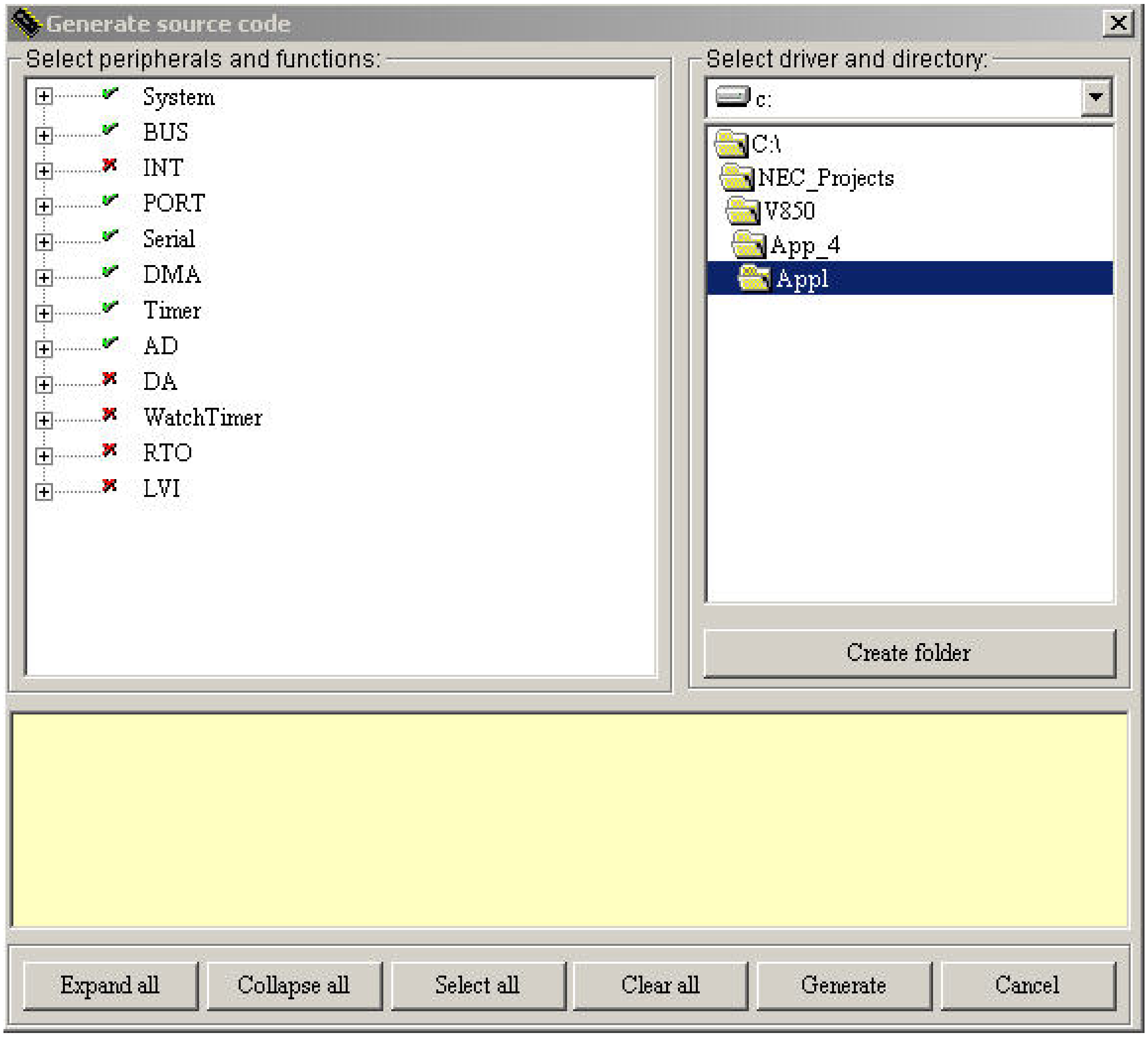
Interrupt setting
☑ Use A/D interrupt
Priority: lowest
● Interrupt each time conversion end
○ Interrupt if conversion result higher comparison threshold value
○ Interrupt if conversion result lower comparison threshold value

Compare value setting
Input comparison(PFT) value: 0

5.2 Generating Code with Applilet

After making all the selections, you are ready to generate the base code. Click **Generate** to create the code for the selected peripherals and system initialization.

Figure 25. Generating Code with Applilet



Generate source code

Select peripherals and functions:

- ✓ System
- ✓ BUS
- ✗ INT
- ✓ PORT
- ✓ Serial
- ✓ DMA
- ✓ Timer
- ✓ AD
- ✗ DA
- ✗ WatchTimer
- ✗ RTO
- ✗ LVI

Select driver and directory:

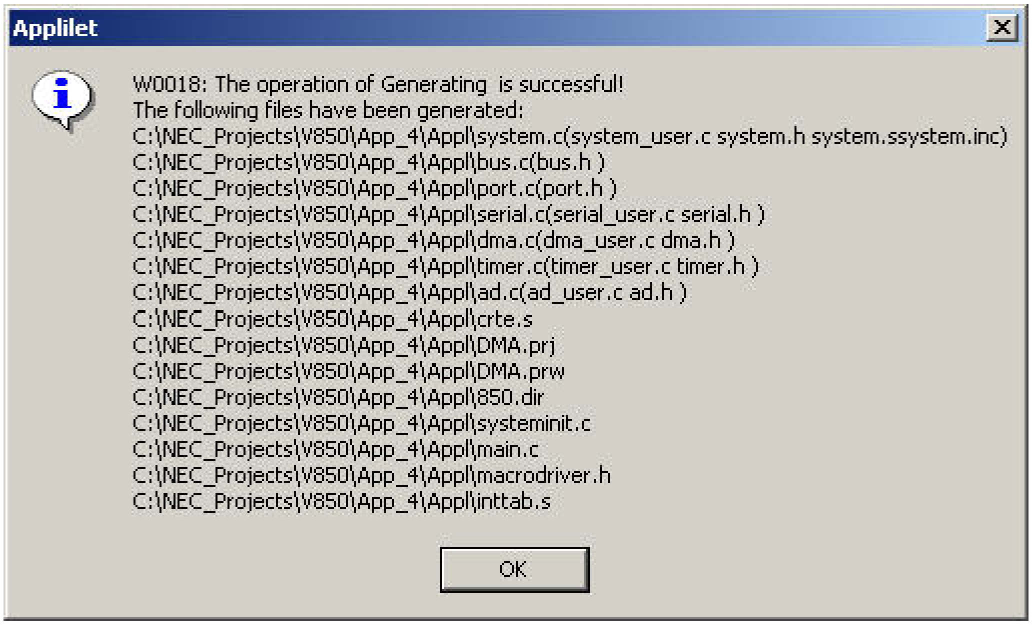
c:

C:\
NEC_Projects
V850
App_4
Appl

5.3 Applilet Generated Files

The screen below shows the list of files generated by the Applilet.

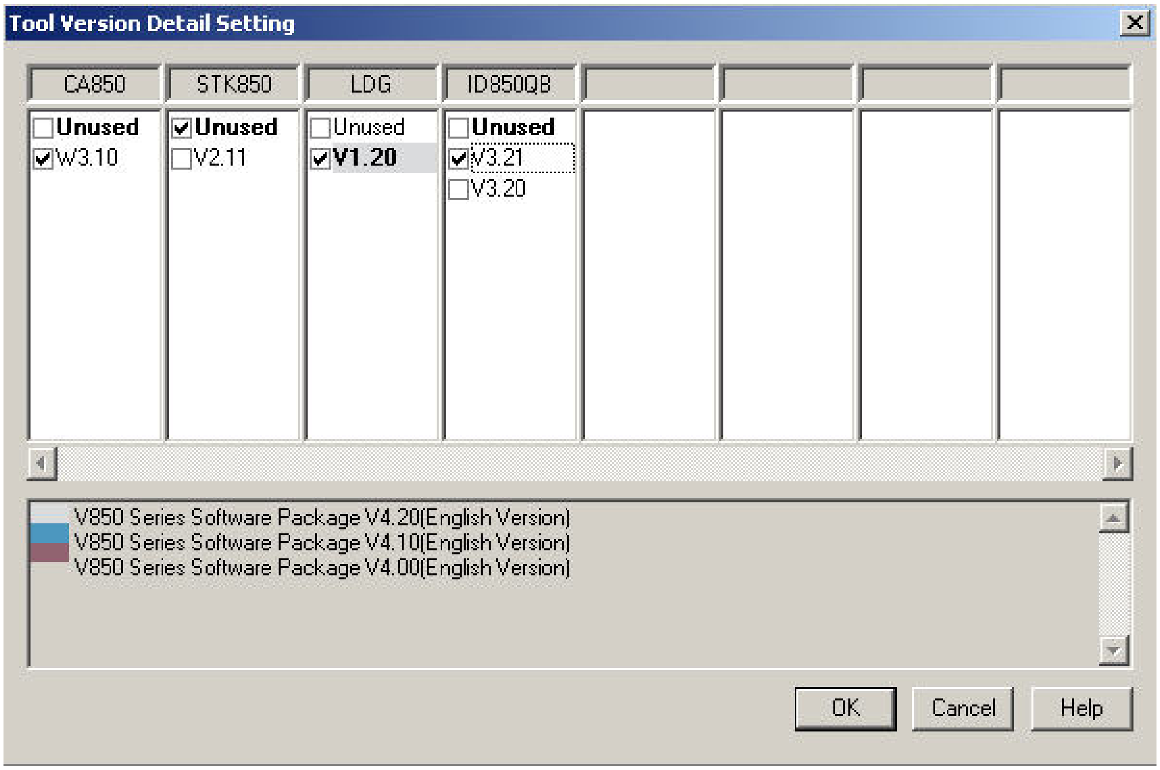
Figure 26. Files Generated by Applilet



```
W0018: The operation of Generating  is successful!
The following files have been generated:
C:\NEC_Projects\V850\App_4\Appl\system.c(system_user.c system.h system.ssystem.inc)
C:\NEC_Projects\V850\App_4\Appl\bus.c(bus.h )
C:\NEC_Projects\V850\App_4\Appl\port.c(port.h )
C:\NEC_Projects\V850\App_4\Appl\serial.c(serial_user.c serial.h )
C:\NEC_Projects\V850\App_4\Appl\dma.c(dma_user.c dma.h )
C:\NEC_Projects\V850\App_4\Appl\timer.c(timer_user.c timer.h )
C:\NEC_Projects\V850\App_4\Appl\ad.c(ad_user.c ad.h )
C:\NEC_Projects\V850\App_4\Appl\crte.s
C:\NEC_Projects\V850\App_4\Appl\DMA.prj
C:\NEC_Projects\V850\App_4\Appl\DMA.prw
C:\NEC_Projects\V850\App_4\Appl\850.dir
C:\NEC_Projects\V850\App_4\Appl\systeminit.c
C:\NEC_Projects\V850\App_4\Appl\main.c
C:\NEC_Projects\V850\App_4\Appl\macrodriver.h
C:\NEC_Projects\V850\App_4\Appl\inttab.s
```

Double-click on the DMA.prw file to bring up the project manager. You may be asked to select the tools the manager should use, as shown in the screen below.

Figure 27. Tool Version Details

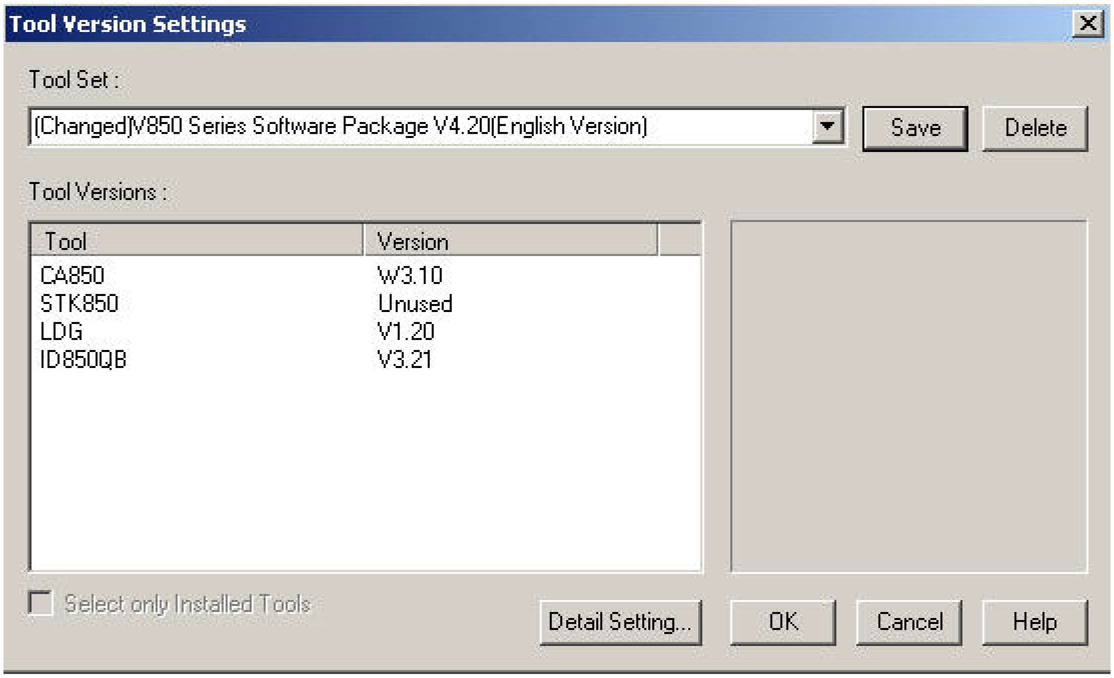


CA850	STK850	LDG	ID850QB
□ Unused	☑ Unused	□ Unused	□ Unused
☑ W3.10	□ V2.11	☑ V1.20	☑ V3.21
			□ V3.20

V850 Series Software Package V4.20(English Version)
V850 Series Software Package V4.10(English Version)
V850 Series Software Package V4.00(English Version)

The next screen summarizes the tools used to compile, link and debug the example code.

Figure 28. Tool Version Summary



Tool Set : [Changed]V850 Series Software Package V4.20(English Version)

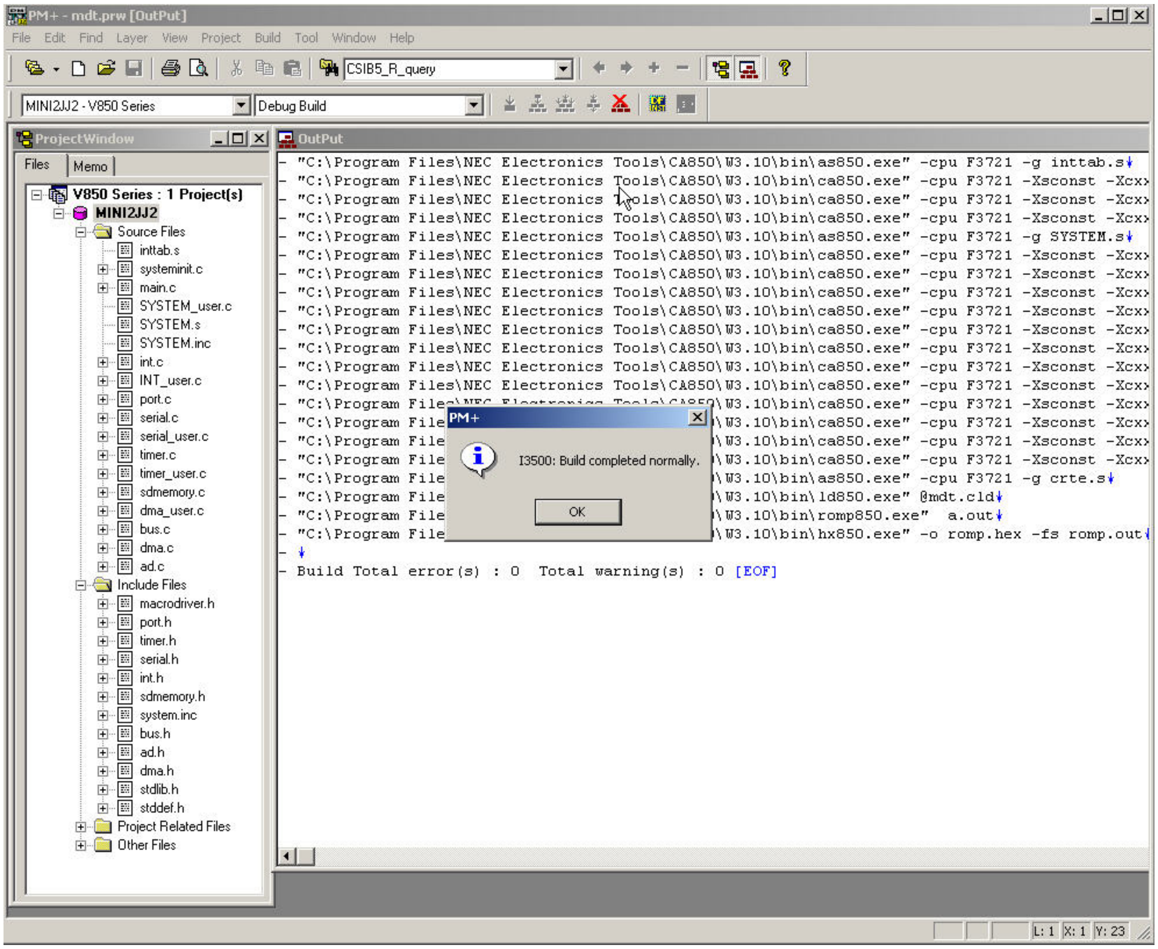
Tool Versions :

Tool	Version
CA850	W3.10
STK850	Unused
LDG	V1.20
ID850QB	V3.21

□ Select only Installed Tools

The screen shot below shows an example of the project manager listing all the files in the project.

Figure 29. Project Manager Shows All Files Used

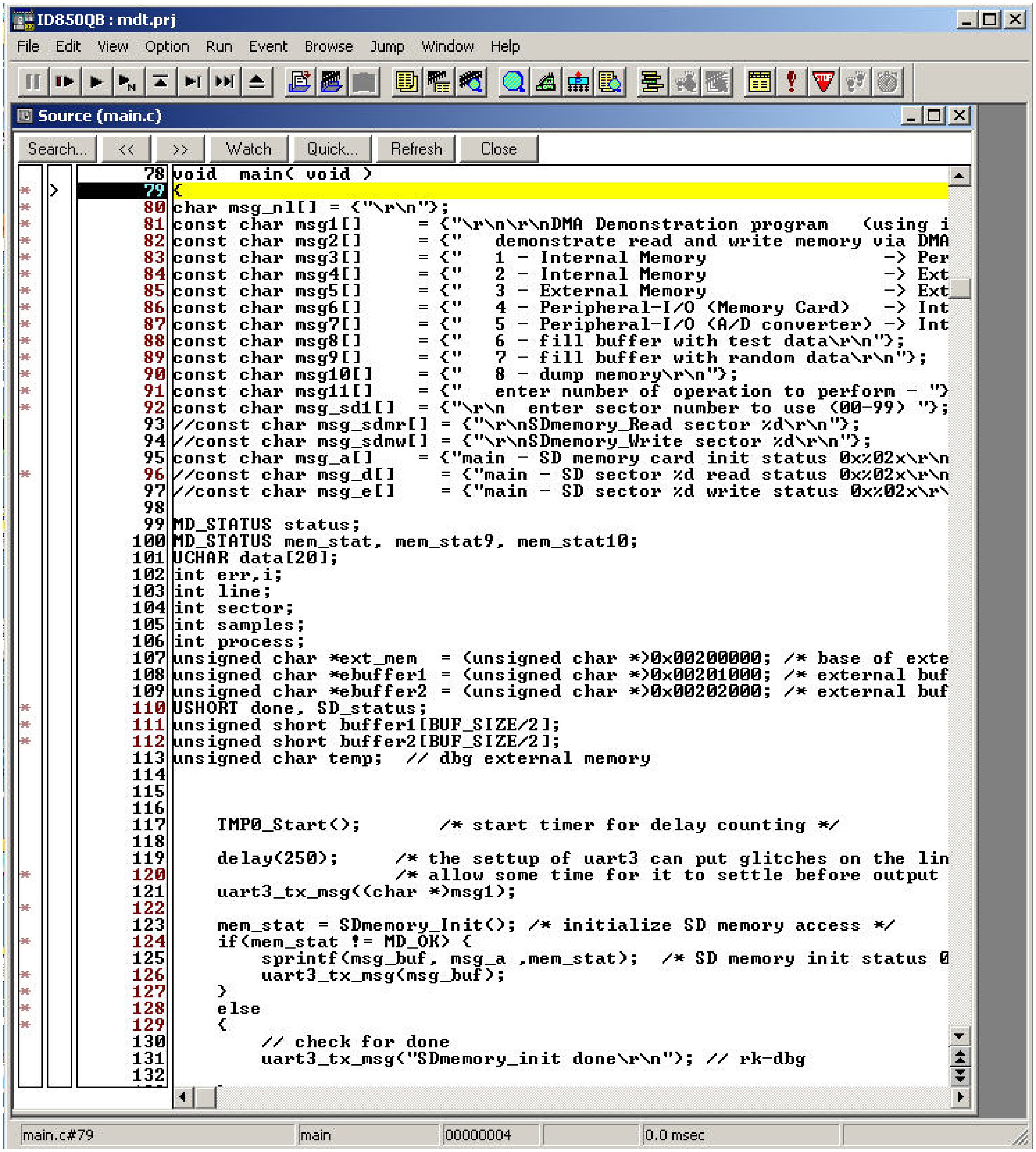


```
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\as850.exe" -cpu F3721 -g inttab.s↓
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\as850.exe" -cpu F3721 -g SYSTEM.s↓
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program Files\NEC Electronics Tools\CA850\W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program File                             \W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program File                             \W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program File                             \W3.10\bin\ca850.exe" -cpu F3721 -Xsconst -Xcx›
- "C:\Program File                             \W3.10\bin\as850.exe" -cpu F3721 -g crte.s↓
- "C:\Program File                             \W3.10\bin\ld850.exe" @mdt.cld↓
- "C:\Program File                             \W3.10\bin\romp850.exe"  a.out↓
- "C:\Program File                             \W3.10\bin\hx850.exe" -o romp.hex -fs romp.out↓
- ↓
- Build Total error(s) : 0  Total warning(s) : 0 [EOF]
```

I3500: Build completed normally.

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

Figure 30.   Program Downloaded Via Debugger and MiniCube2


ID850QB : mdt.prj
File Edit View Option Run Event Browse Jump Window Help
Source (main.c)
Search... << >> Watch Quick... Refresh Close
main.c#79
main
00000004
0.0 msec

```
void  main( void )
{
char msg_nl[] = {"\r\n"};
const char msg1[]     = {"\r\n\r\nDMA Demonstration program  (using i
const char msg2[]     = {"   demonstrate read and write memory via DMA
const char msg3[]     = {"   1 - Internal Memory                -> Per
const char msg4[]     = {"   2 - Internal Memory                -> Ext
const char msg5[]     = {"   3 - External Memory                -> Ext
const char msg6[]     = {"   4 - Peripheral-I/O (Memory Card)   -> Int
const char msg7[]     = {"   5 - Peripheral-I/O (A/D converter) -> Int
const char msg8[]     = {"   6 - fill buffer with test data\r\n"};
const char msg9[]     = {"   7 - fill buffer with random data\r\n"};
const char msg10[]    = {"   8 - dump memory\r\n"};
const char msg11[]    = {"   enter number of operation to perform - "}
const char msg_sd1[]  = {"\r\n  enter sector number to use (00-99) "};
//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 - SD memory card init status 0x%02x\r\n
//const char msg_d[]    = {"main - SD sector %d read status 0x%02x\r\n
//const char msg_e[]    = {"main - SD sector %d write status 0x%02x\r\

MD_STATUS status;
MD_STATUS mem_stat, mem_stat9, mem_stat10;
UCHAR data[20];
int err,i;
int line;
int sector;
int samples;
int process;
unsigned char *ext_mem  = (unsigned char *)0x00200000; /* base of exte
unsigned char *ebuffer1 = (unsigned char *)0x00201000; /* external buf
unsigned char *ebuffer2 = (unsigned char *)0x00202000; /* external buf
USHORT done, SD_status;
unsigned short buffer1[BUF_SIZE/2];
unsigned short buffer2[BUF_SIZE/2];
unsigned char temp;  // dbg external memory



    TMP0_Start();        /* start timer for delay counting */

    delay(250);      /* the settup of uart3 can put glitches on the lin
                     /* allow some time for it to settle before output
    uart3_tx_msg((char *)msg1);

    mem_stat = SDmemory_Init(); /* initialize SD memory access */
    if(mem_stat != MD_OK) {
        sprintf(msg_buf, msg_a ,mem_stat);  /* SD memory init status 0
        uart3_tx_msg(msg_buf);
    }
    else
    {
        // check for done
        uart3_tx_msg("SDmemory_init done\r\n"); // rk-dbg
```

6. Demonstration Platform

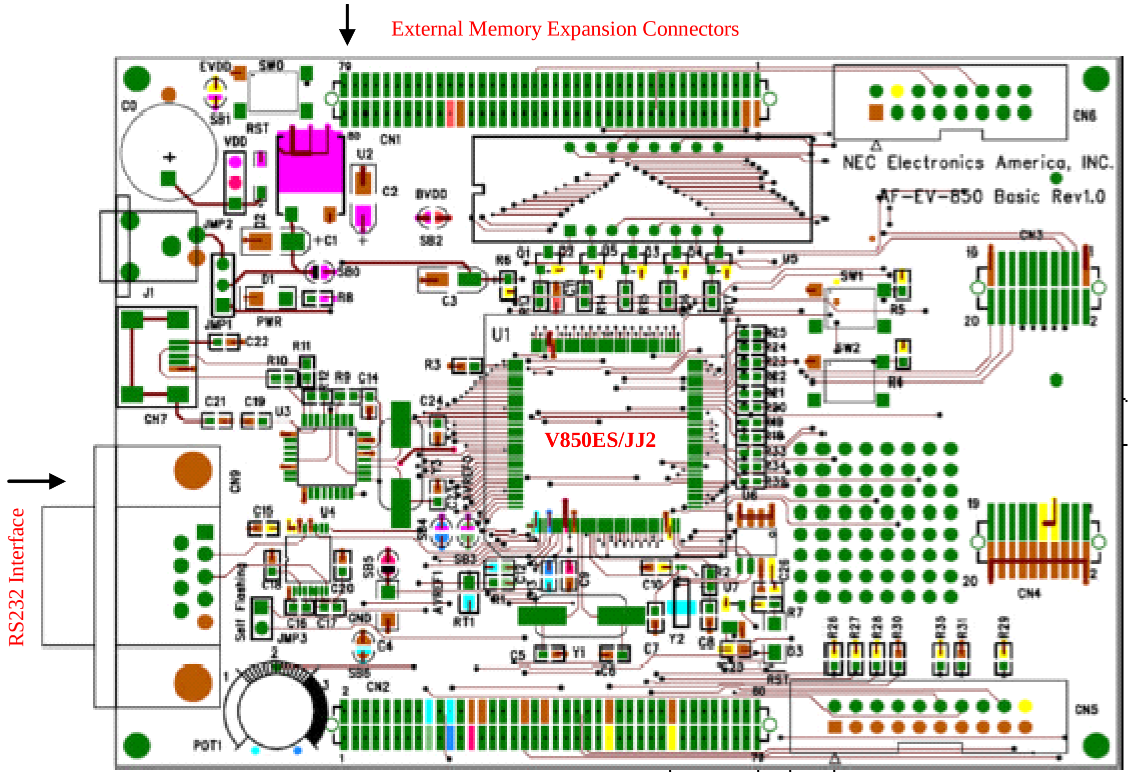
The demonstration uses a development board from NEC Electronics. You may be able to duplicate the same hardware using off-the-shelf components along with the NEC Electronics microcontroller of interest.

This demonstration uses the V850ES Demo-Board, AF-V850ES/JJ2, which has an SD memory card interface and an RS-232 connection to the host PC. The board also provides pushbutton switches. An SRAM module is attached to external memory expansion connectors, providing 4 Mbits of storage.

6.1 Resources

The AF-EV850ES/JJ2 board is designed for evaluation of Flash V850ES Series microcontrollers from NEC Electronics.

Figure 31. Demonstration Board



External Memory Expansion Connectors

RS232 Interface

V850ES/JJ2

The AF-EV850ES/JJ2 board features:

- NEC Electronics’ 850ES/JJ2 D70F3721GJ microcontroller
- USB FTDI chip
- MAXIM MAX232 RS-232 driver for UART host interface
- 4-digit, 7-segment LED

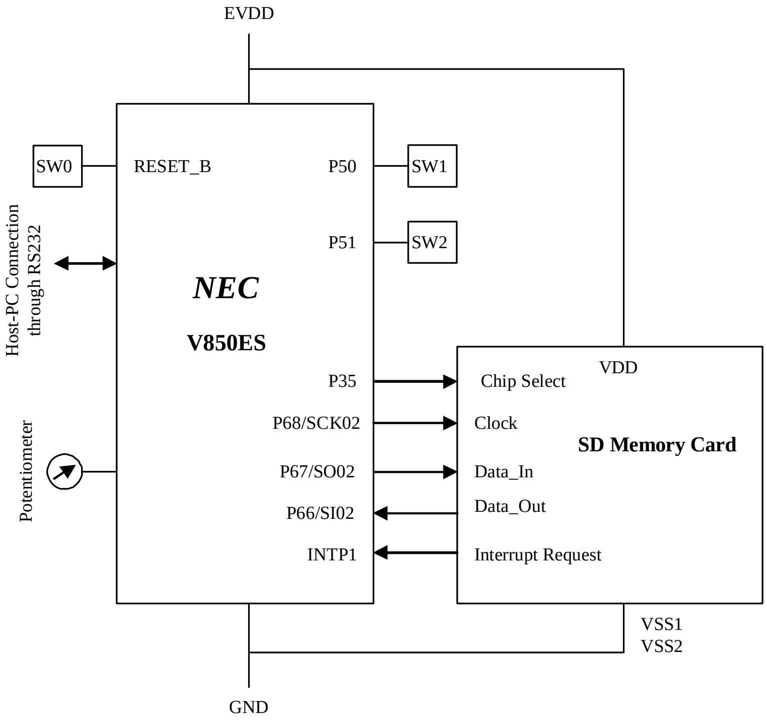
- A/D potentiometer
- SD memory card connector
- MiniCUBE debugging interface
- MiniCUBE2 flash programming interface
- External memory and I/O expansion connectors
- USB-to-UART interface
- User switches

The AF-EV850ES/JJ2 board also provides many other features that are useful for evaluating Flash V850ES Series devices.

6.2 SD Memory Card Interface

The diagram below shows the connections to the SD memory card. The interrupt-request line is not used and is only available when using the proprietary protocol, and not the SPI interface.

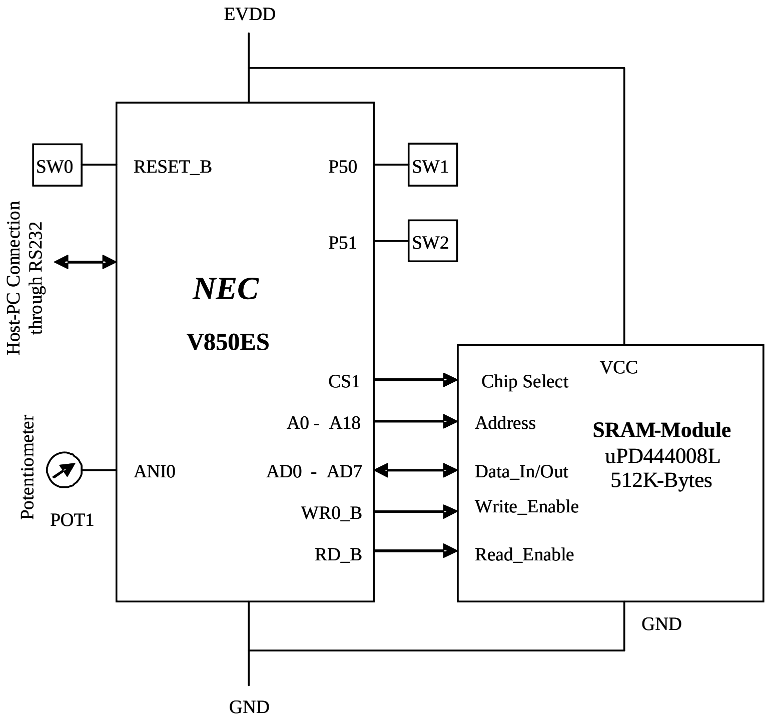
Figure 32. SD Memory Card Interface



6.3 SRAM Module Interface

The following diagram shows the external memory connection, using chip select 1.

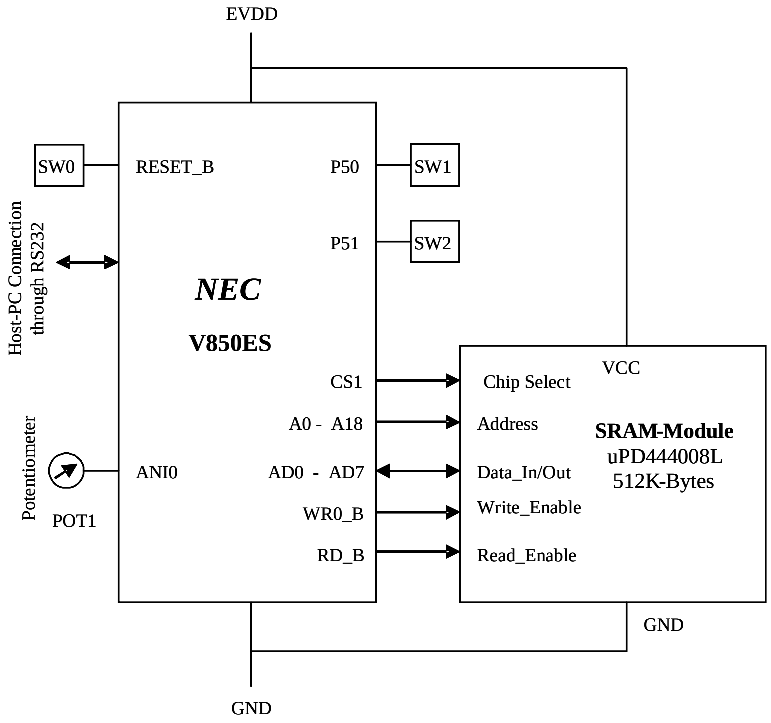
Figure 33. SRAM Module Interface



6.4 A/D Module Interface

The following diagram shows the connection for the A/D channel to the potentiometer.

Figure 34. A/D Module Interface Connections



7. Software Modules

The files shown in the table below comprise the software modules for the demonstration program. The table shows which files are generated by the Applilet and which need modification to create the demonstration program. The listings for these files are in Appendix B.

Table 7. Software Modules

File	Generated by the 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
ad.c	Applilet	modified
ad.h	Applilet	modified
bus.c	Applilet	
bus.h	Applilet	
dma.c	Applilet	modified
dma_user.c	Applilet	modified
int.c	Applilet	
int.h	Applilet	
int_user.c	Applilet	
main.c	Applilet	modified
sdmemory.c		
sdmemory.h		
serial.c	Applilet	modified
serial.h	Applilet	
serial_user.c	Applilet	modified
port.c	Applilet	
port.h	Applilet	
system_user.c		
timer.c	Applilet	
timer_user.c	Applilet	modified
timer.h	Applilet	

8. Appendix A — Software Flowcharts

8.1 System Initialize Flow

Figure 35. DMA_1.0_system_initialize_flow Part 1

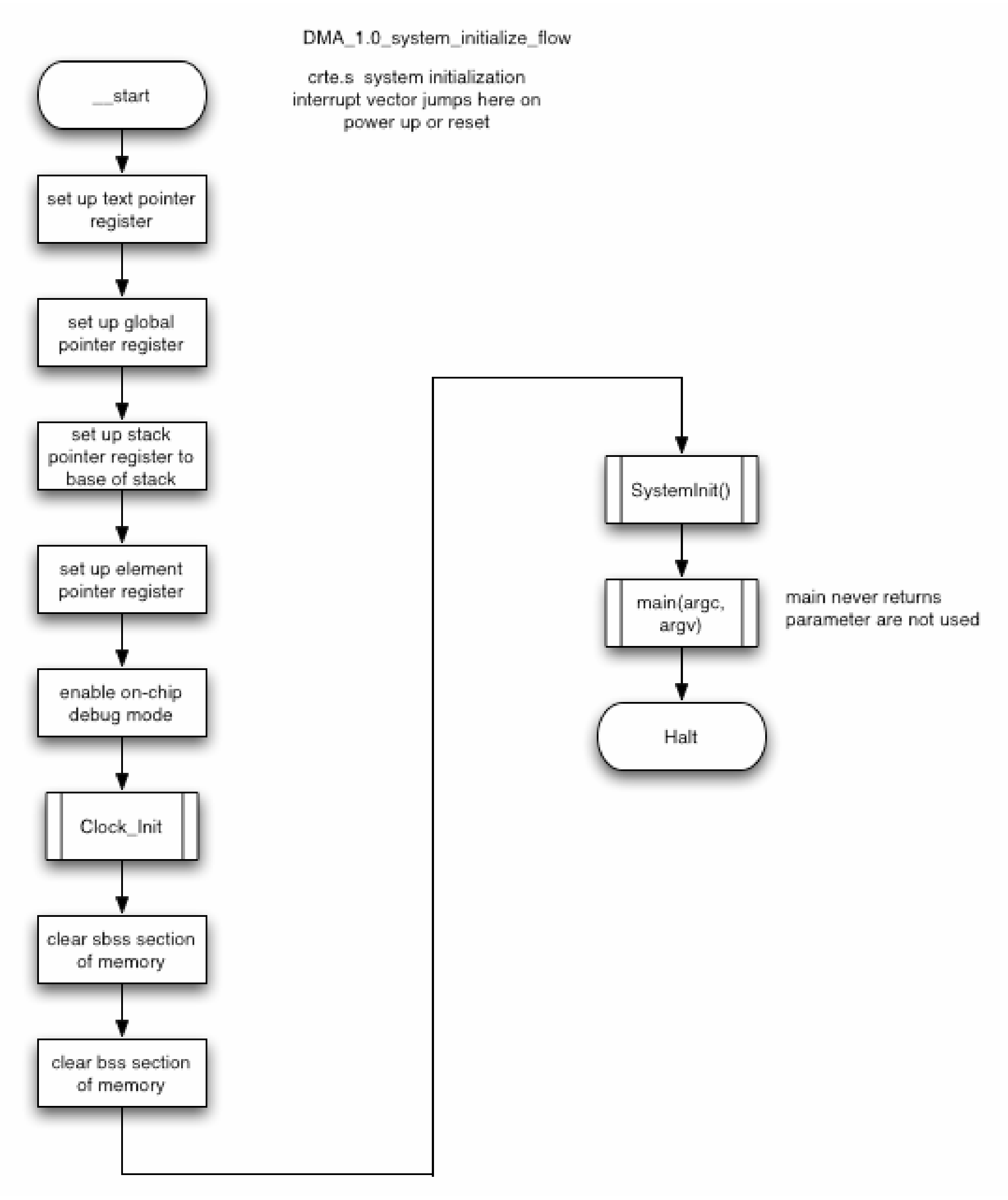
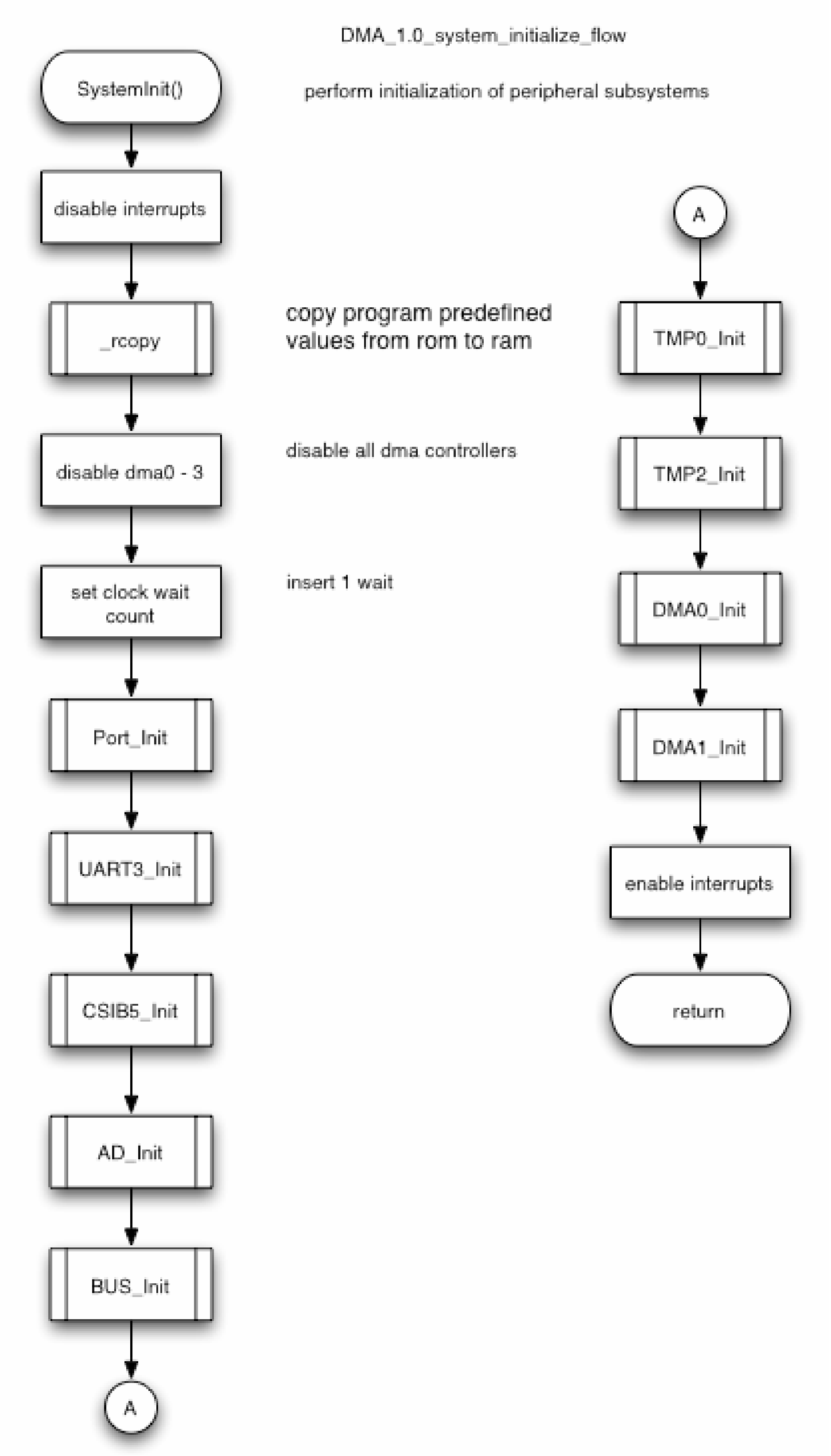

DMA_1.0_system_initialize_flow
crte.s system initialization interrupt vector jumps here on power up or reset
__start
set up text pointer register
set up global pointer register
set up stack pointer register to base of stack
set up element pointer register
enable on-chip debug mode
Clock_Init
clear sbss section of memory
clear bss section of memory
SystemInit()
main(argc, argv)
main never returns parameter are not used
Halt

Figure 36. DMA_1.0_system_initialize_flow Part 2



8.2 DMA Main Flow

Figure 37. DMA_1.1.0_main_flow

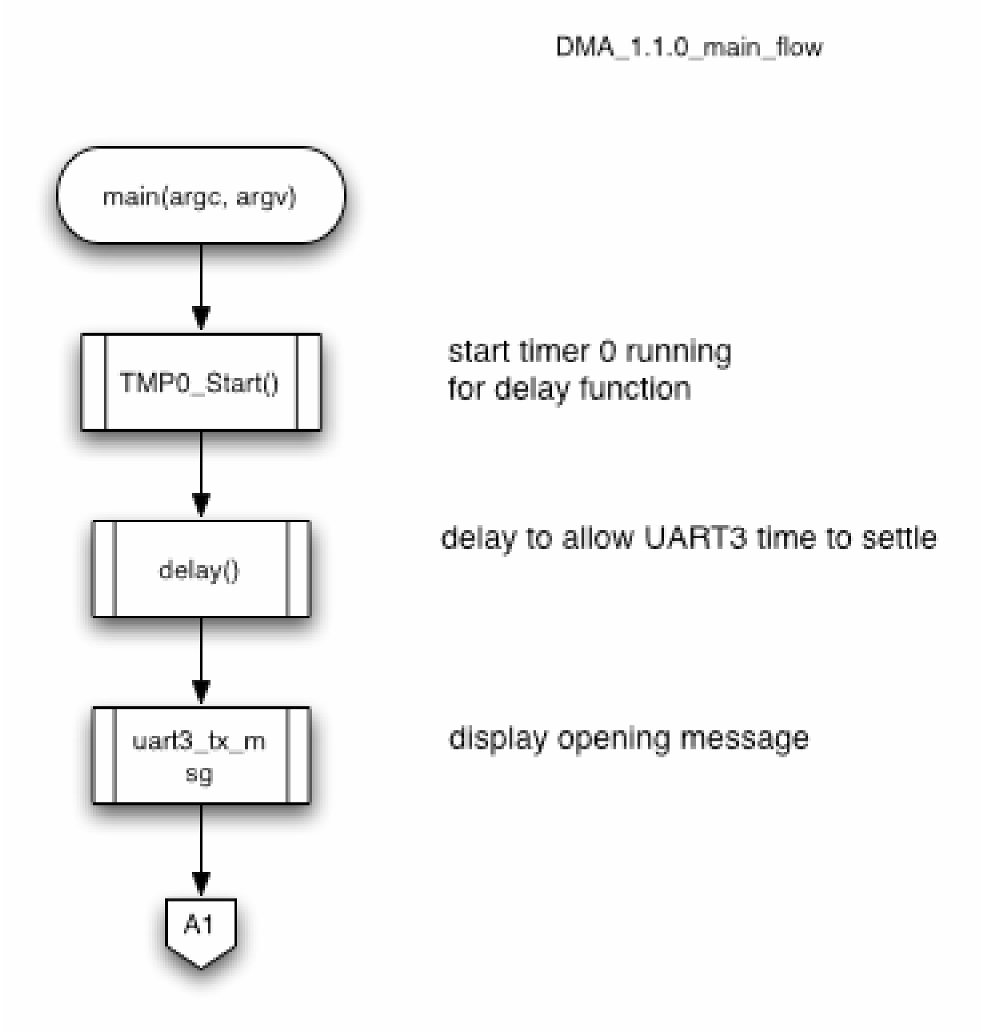


Figure 38. DMA_1.1.1_main_flow

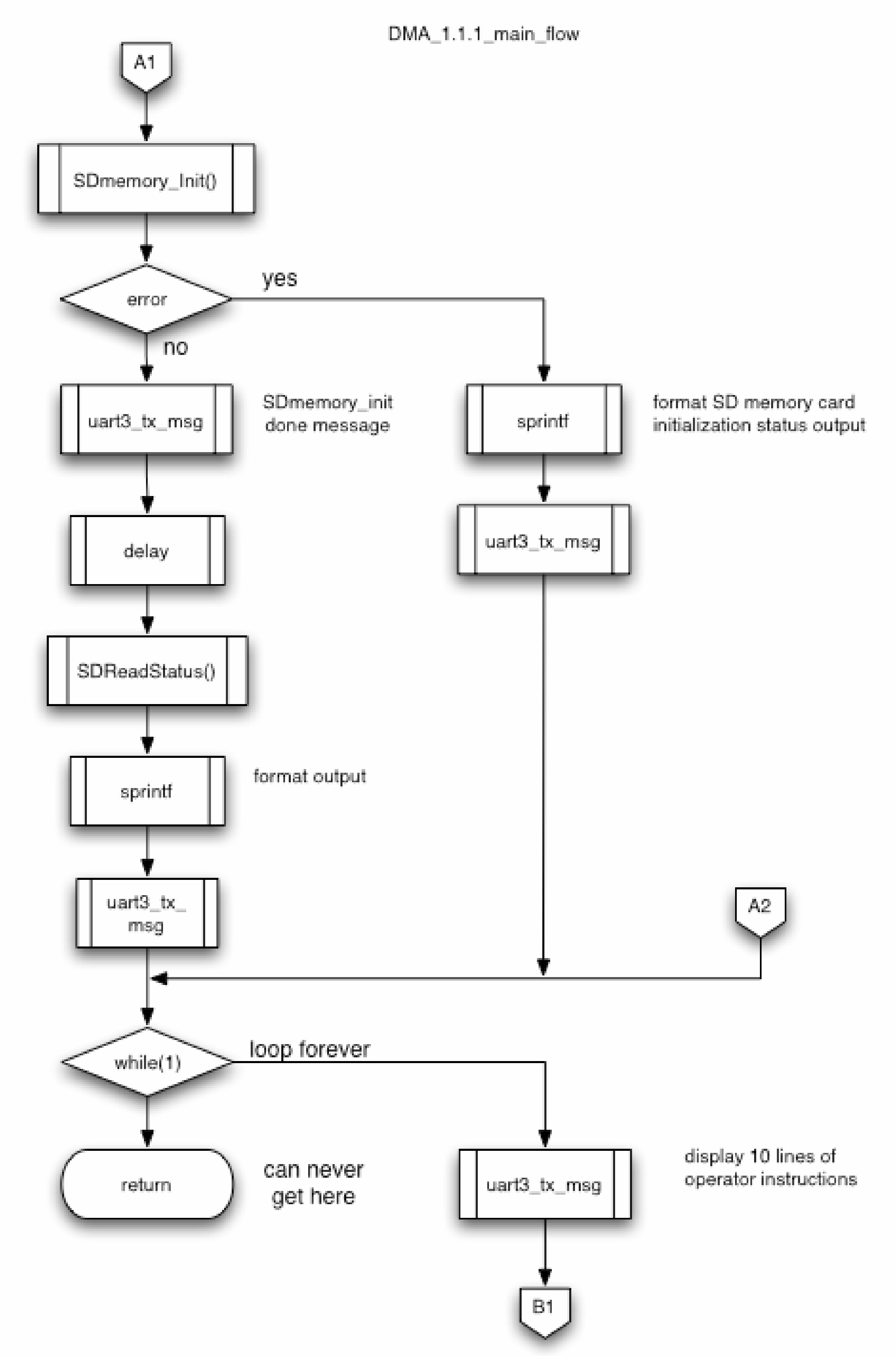


Figure 39.   DMA_1.1.2_main_flow

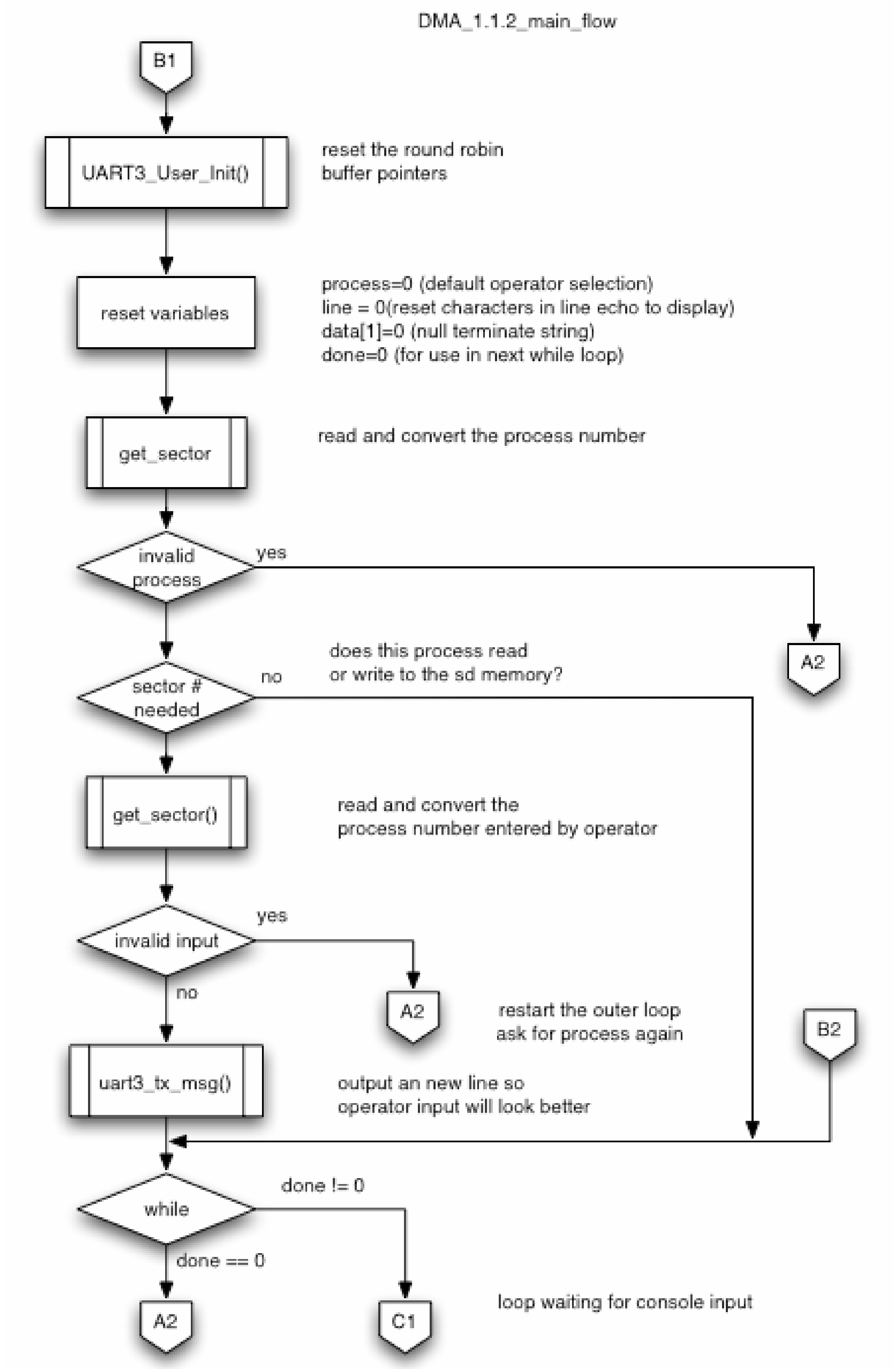


Figure 40.   DMA_1.1.3_main_flow

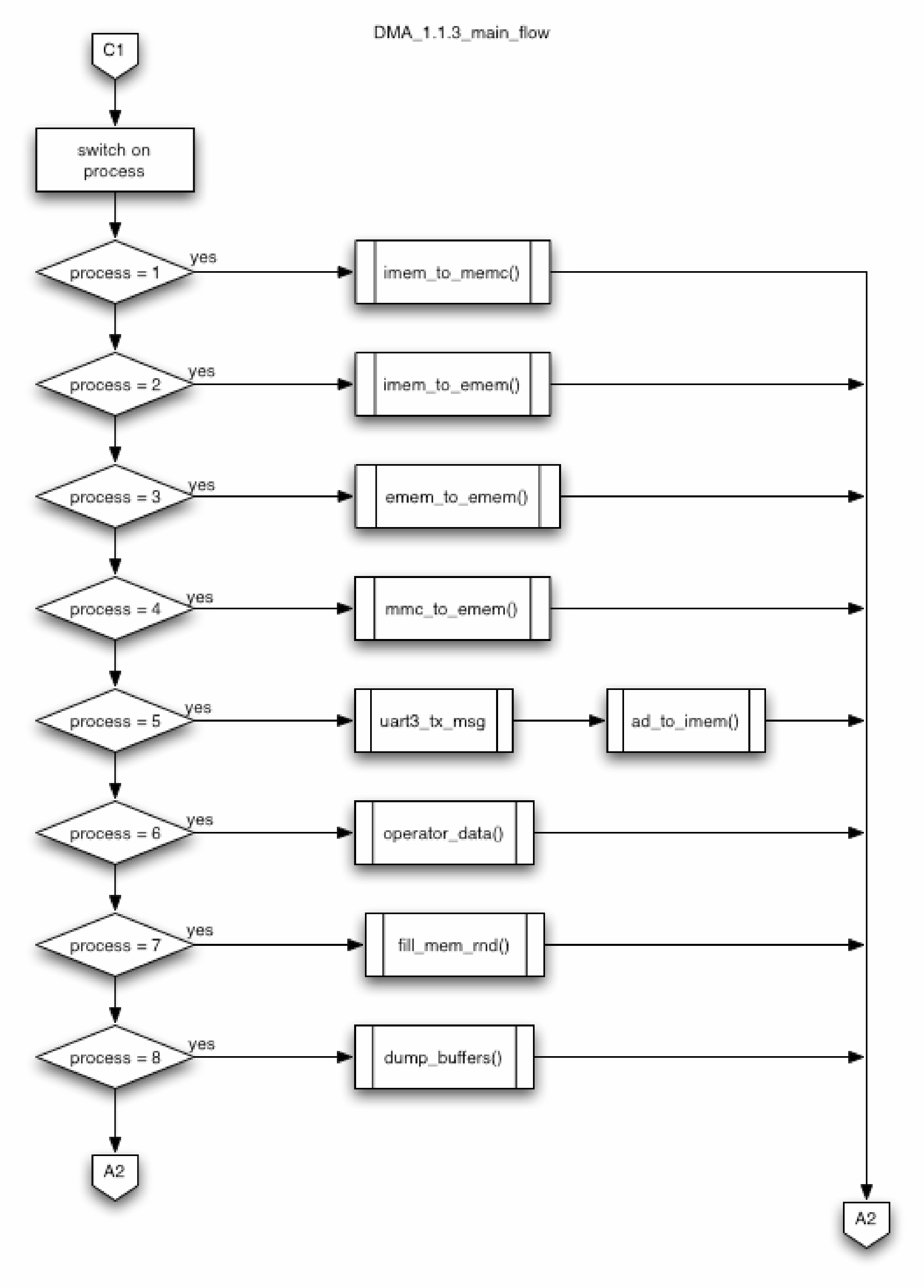


Figure 41. DMA_1.1.4_main_flow

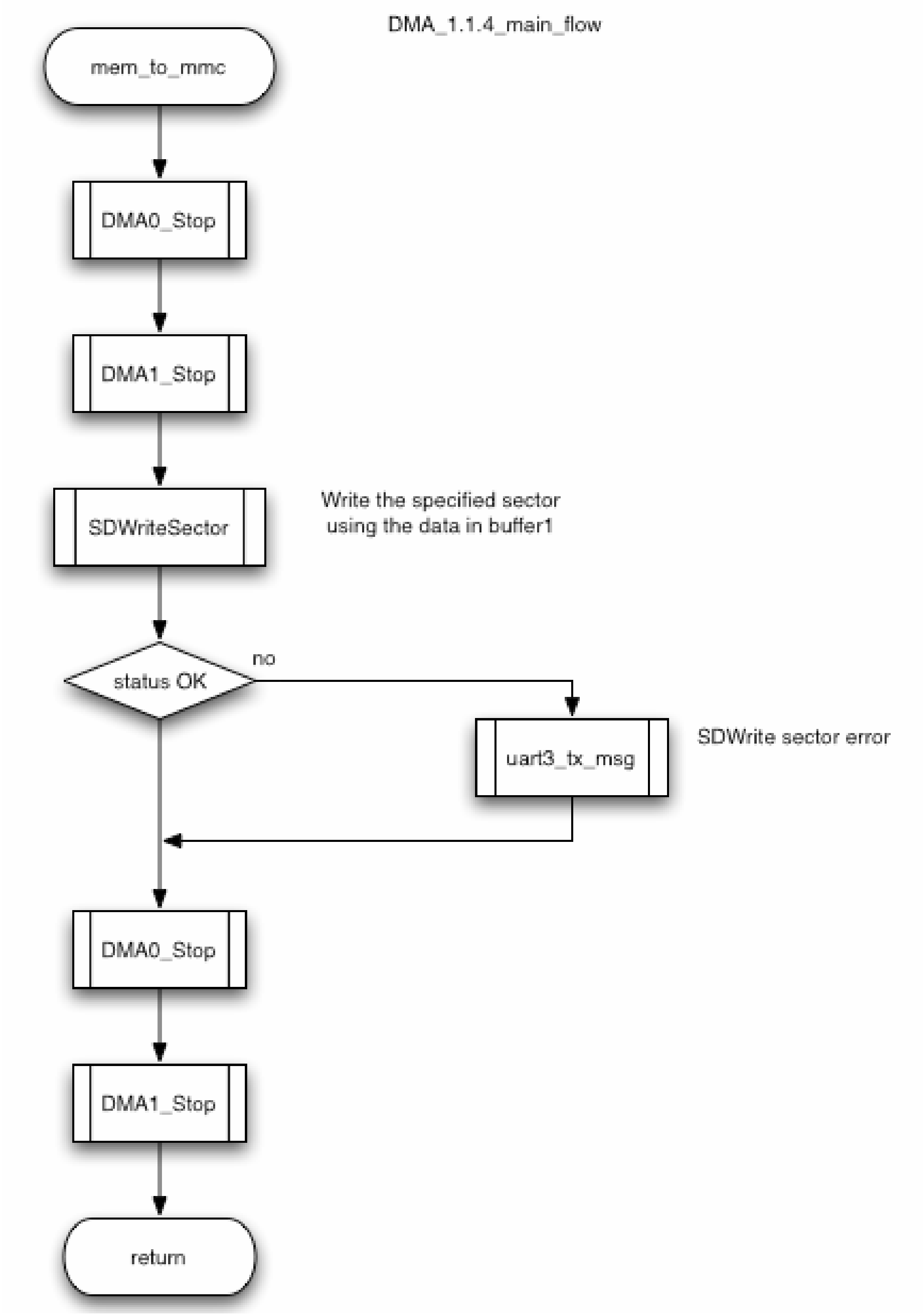


Figure 42.   DMA_1.1.5_main_flow

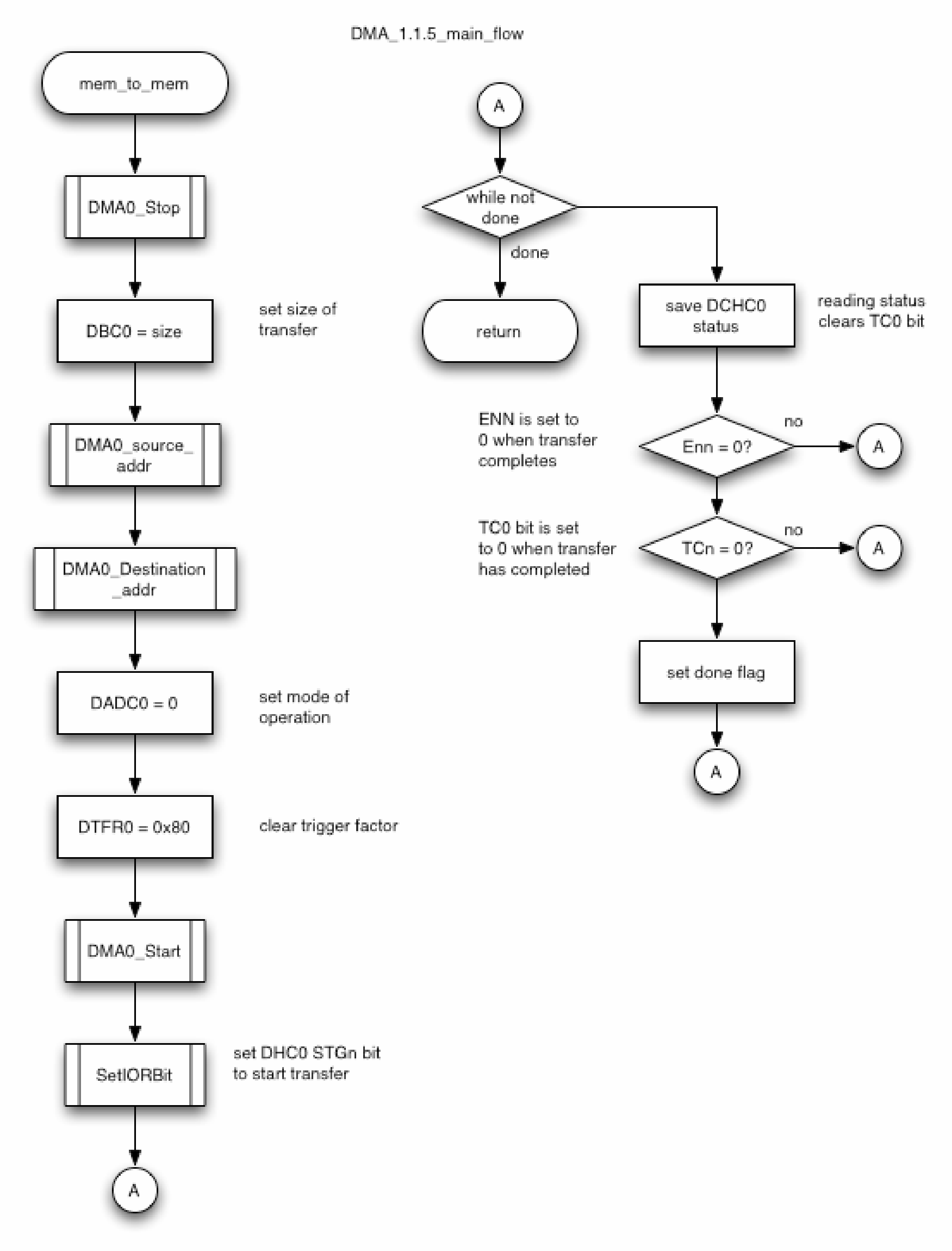


Figure 43. DMA_1.1.6_main_flow

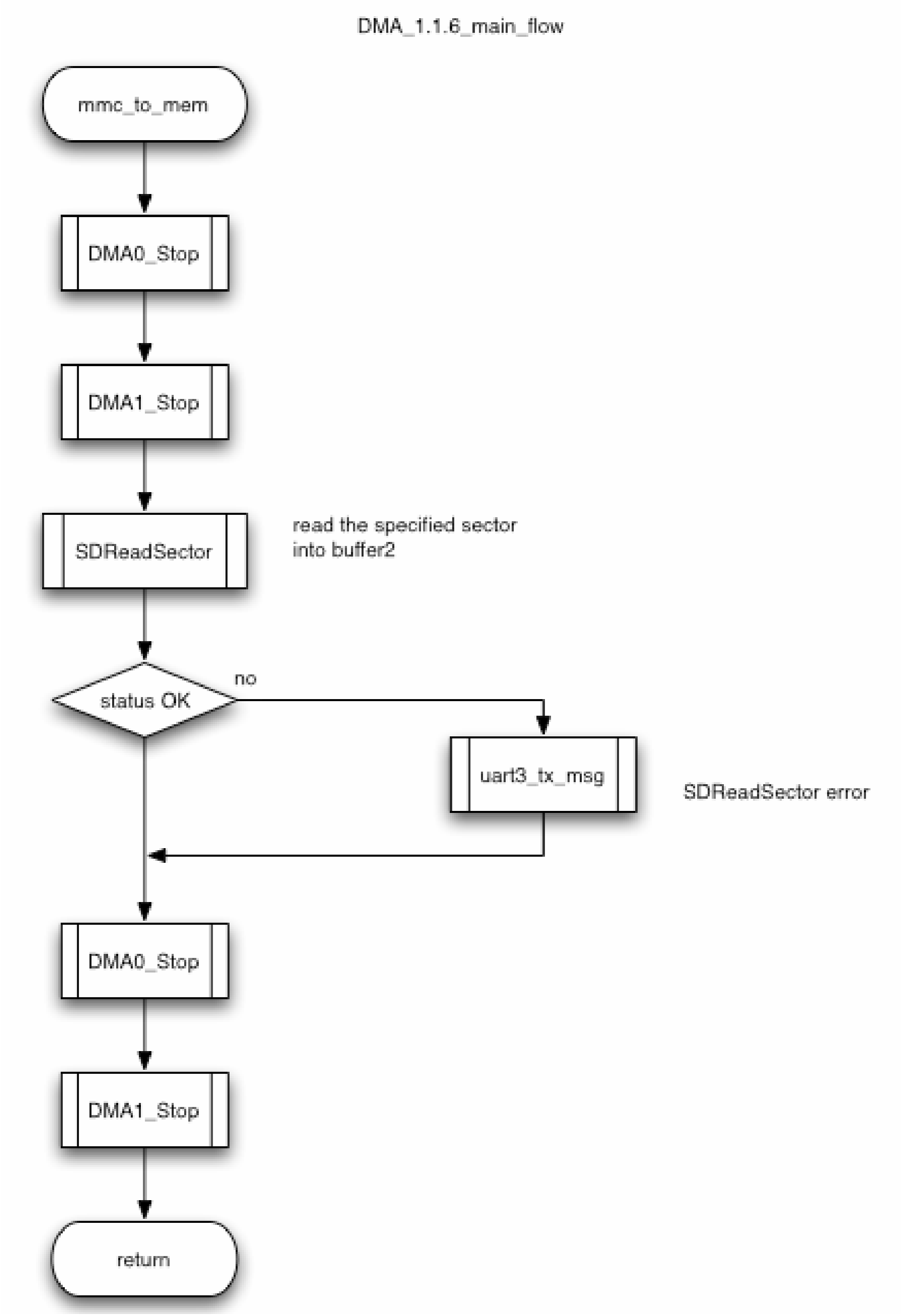


Figure 44.   DMA_1.1.8_main_flow

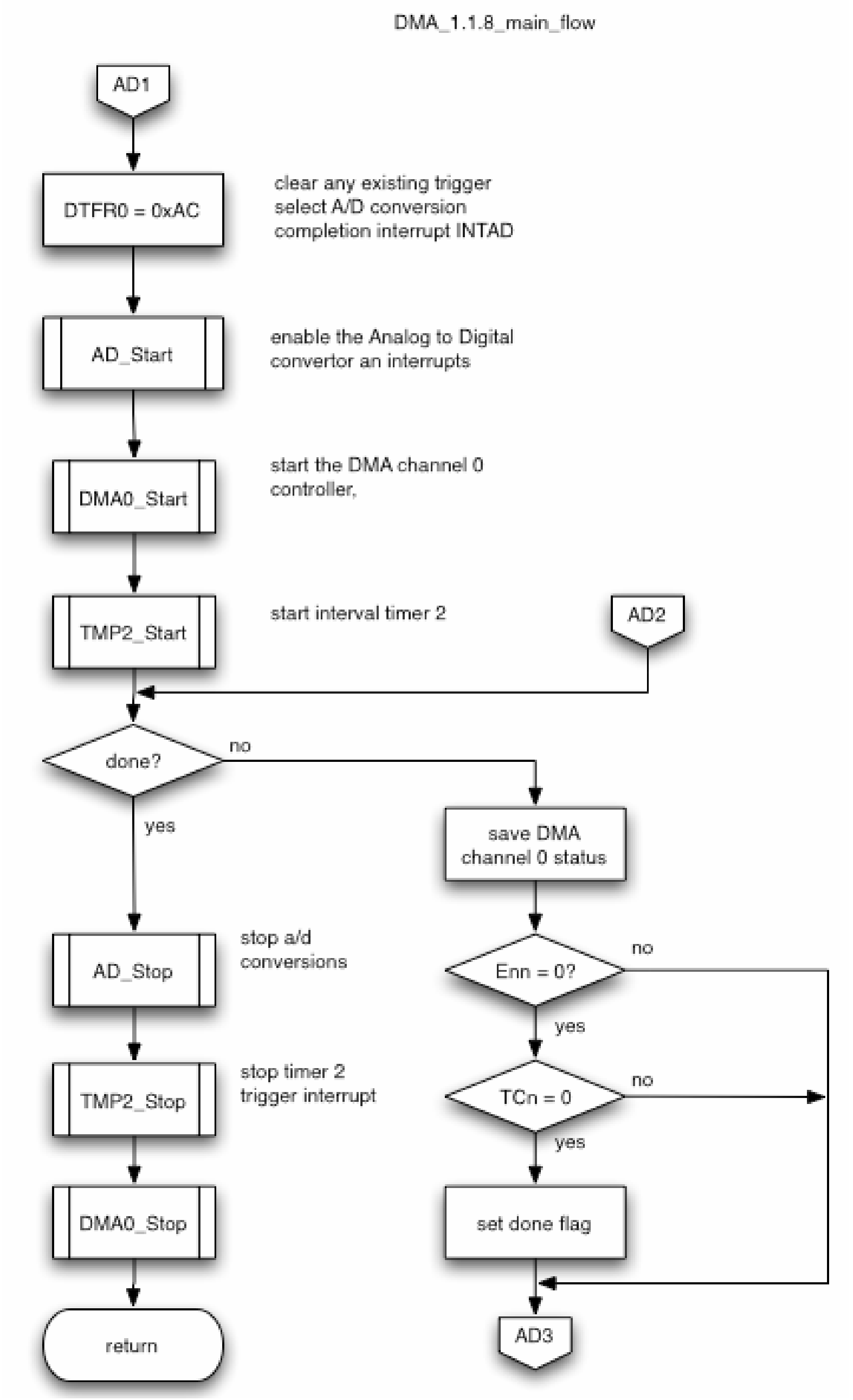


Figure 45. DMA_1.1.9_main_flow

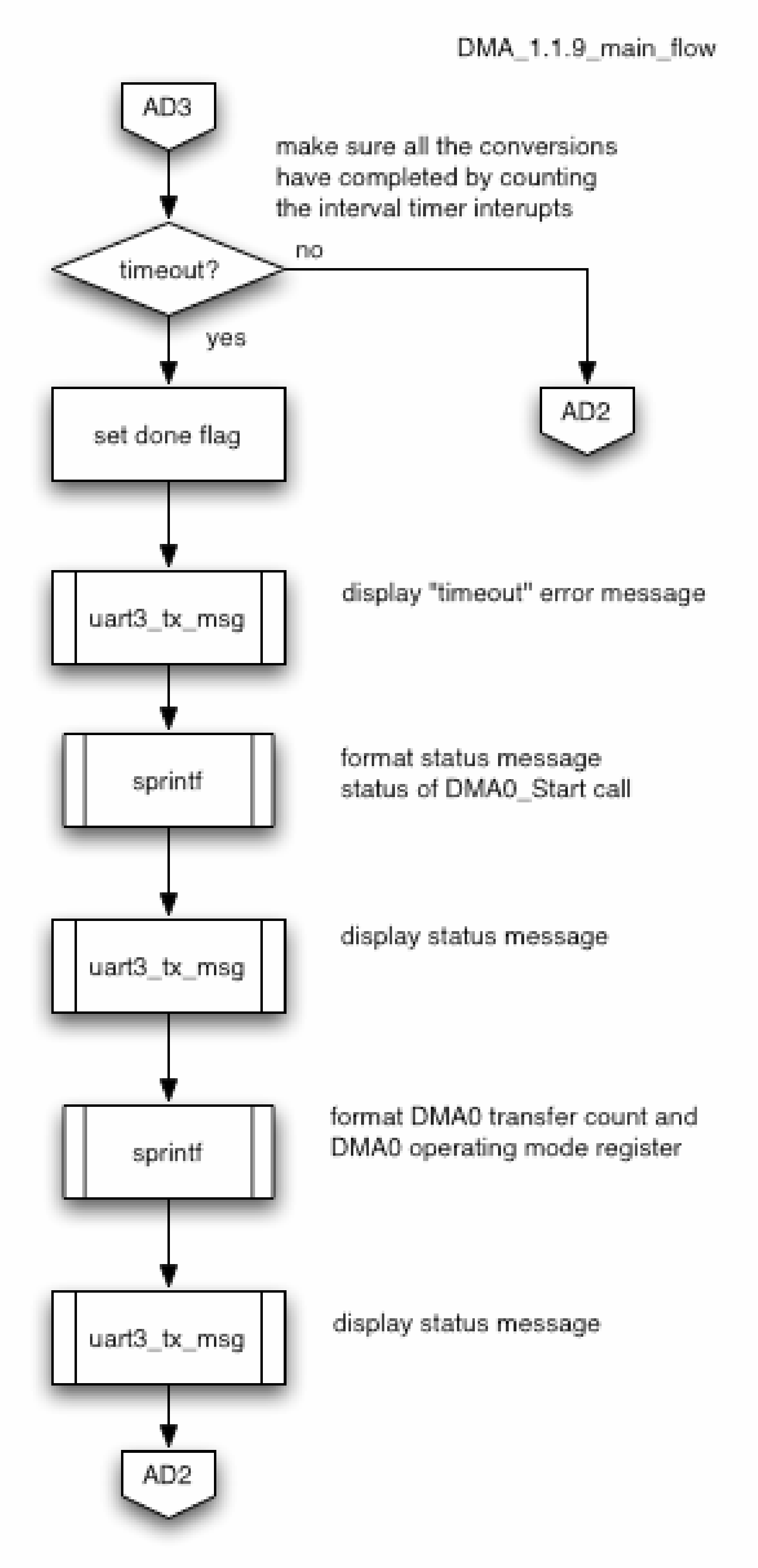

DMA_1.1.9_main_flow
AD3
make sure all the conversions have completed by counting the interval timer interupts
timeout?
no
yes
set done flag
AD2
uart3_tx_msg
display "timeout" error message
sprintf
format status message status of DMA0_Start call
uart3_tx_msg
display status message
sprintf
format DMA0 transfer count and DMA0 operating mode register
uart3_tx_msg
display status message
AD2

Figure 46. DMA_1.1.10_main_flow

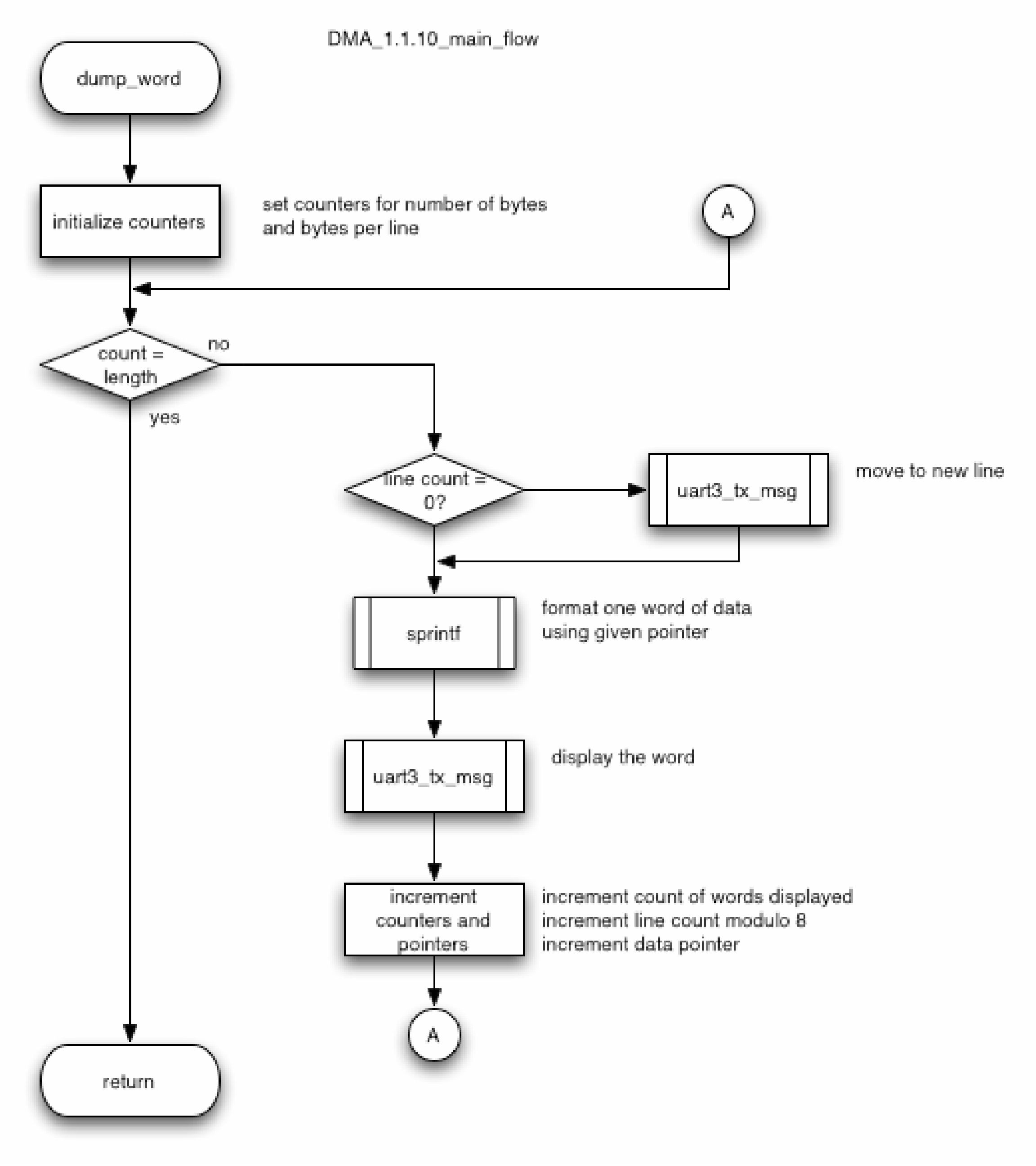


Figure 47. *DMA_1.1.11_main_flow*

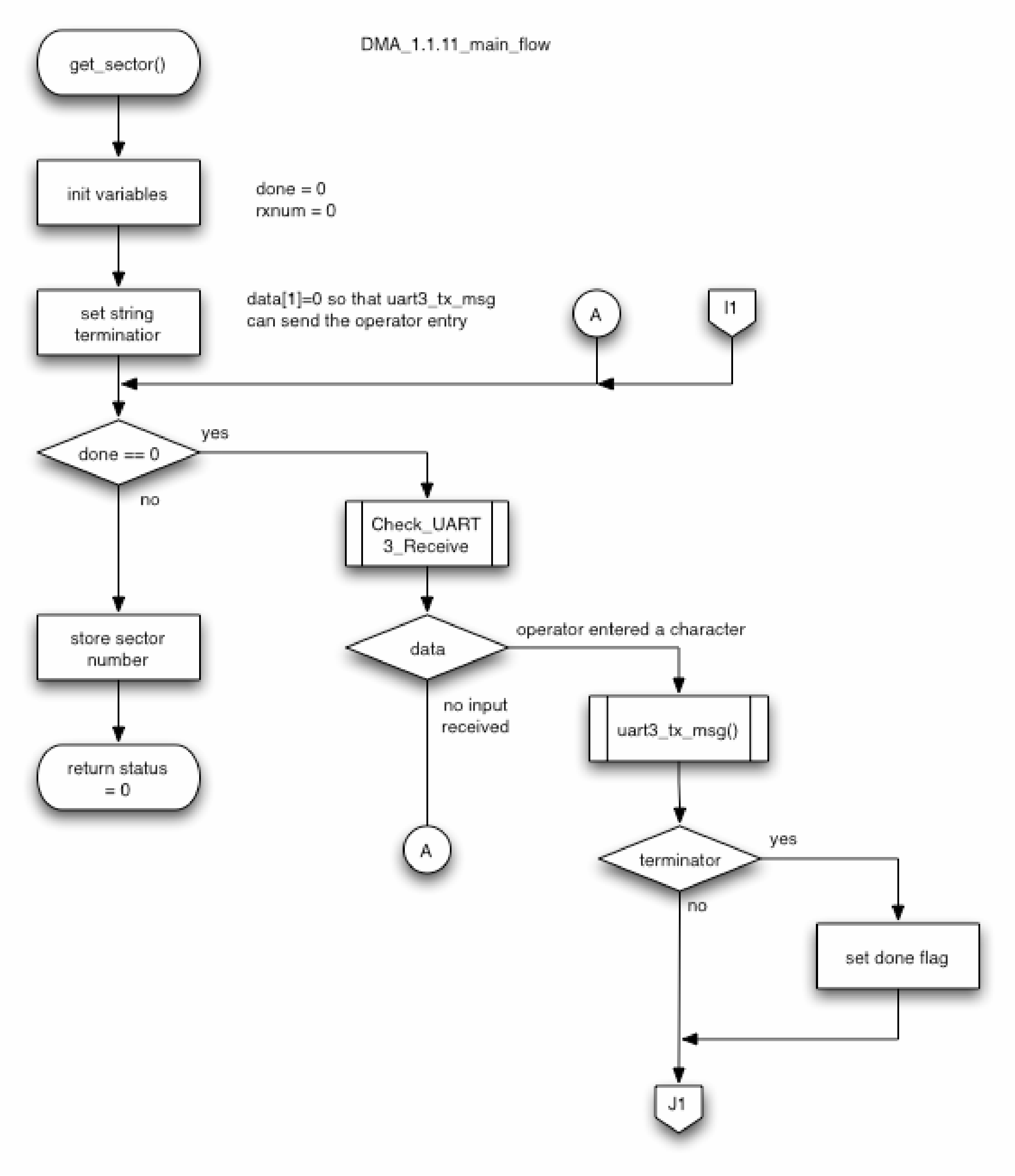


Figure 48. DMA_1.1.12_main_flow

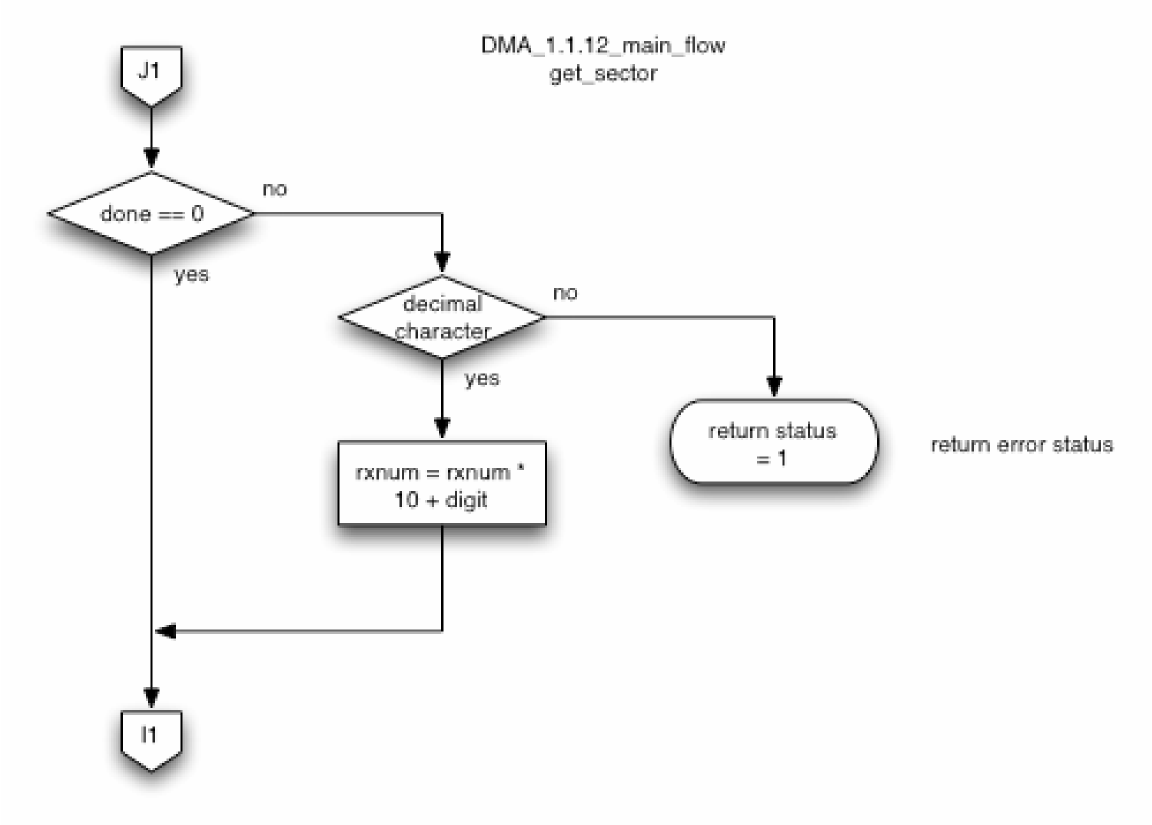


Figure 49. DMA_1.1.13_main_flow

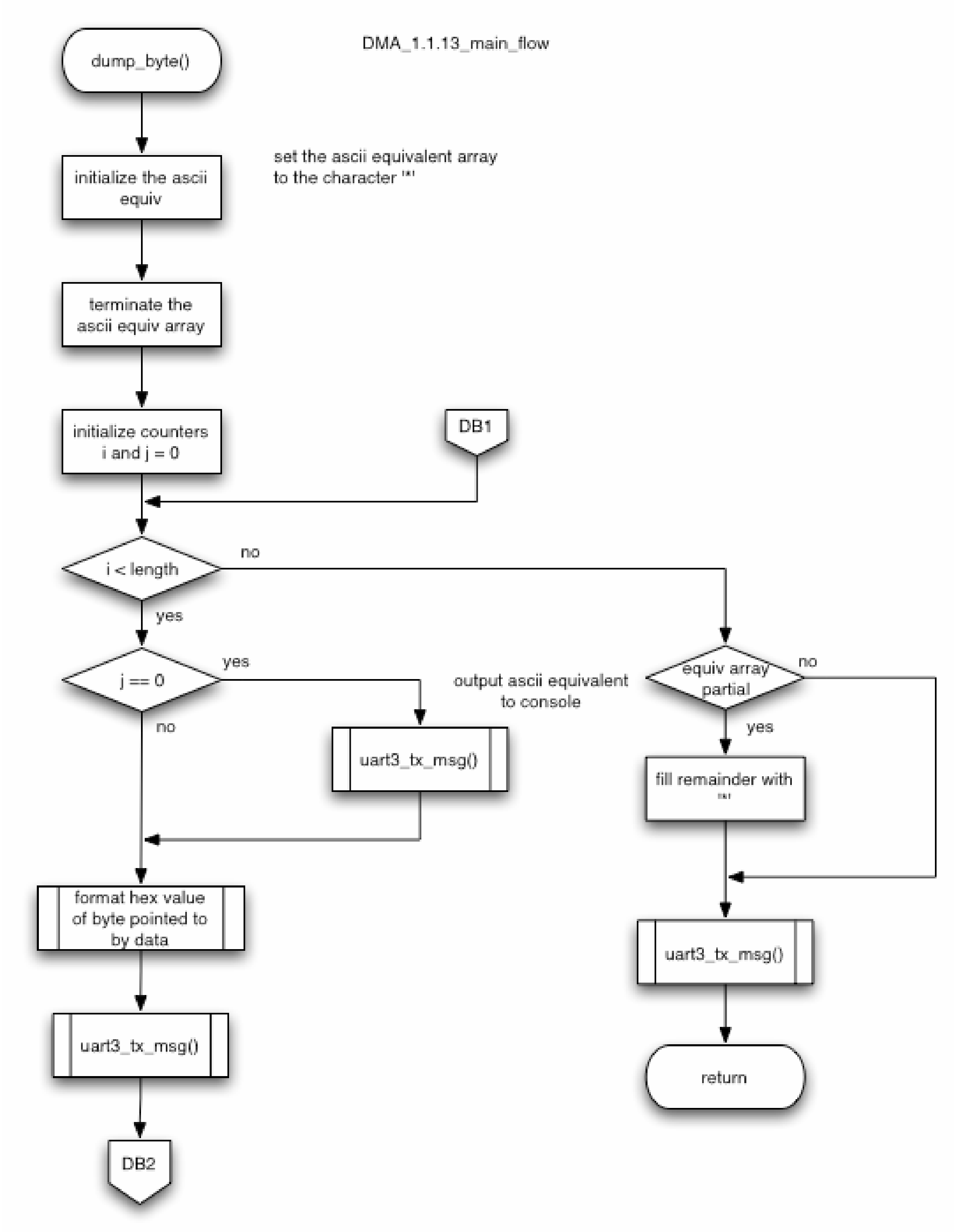

DMA_1.1.13_main_flow
dump_byte()
initialize the ascii equiv
set the ascii equivalent array to the character '*'
terminate the ascii equiv array
initialize counters i and j = 0
DB1
i < length
no
yes
j == 0
yes
no
uart3_tx_msg()
output ascii equivalent to console
equiv array partial
no
yes
fill remainder with '*'
format hex value of byte pointed to by data
uart3_tx_msg()
uart3_tx_msg()
DB2
return

Figure 50. DMA_1.1.14_main_flow

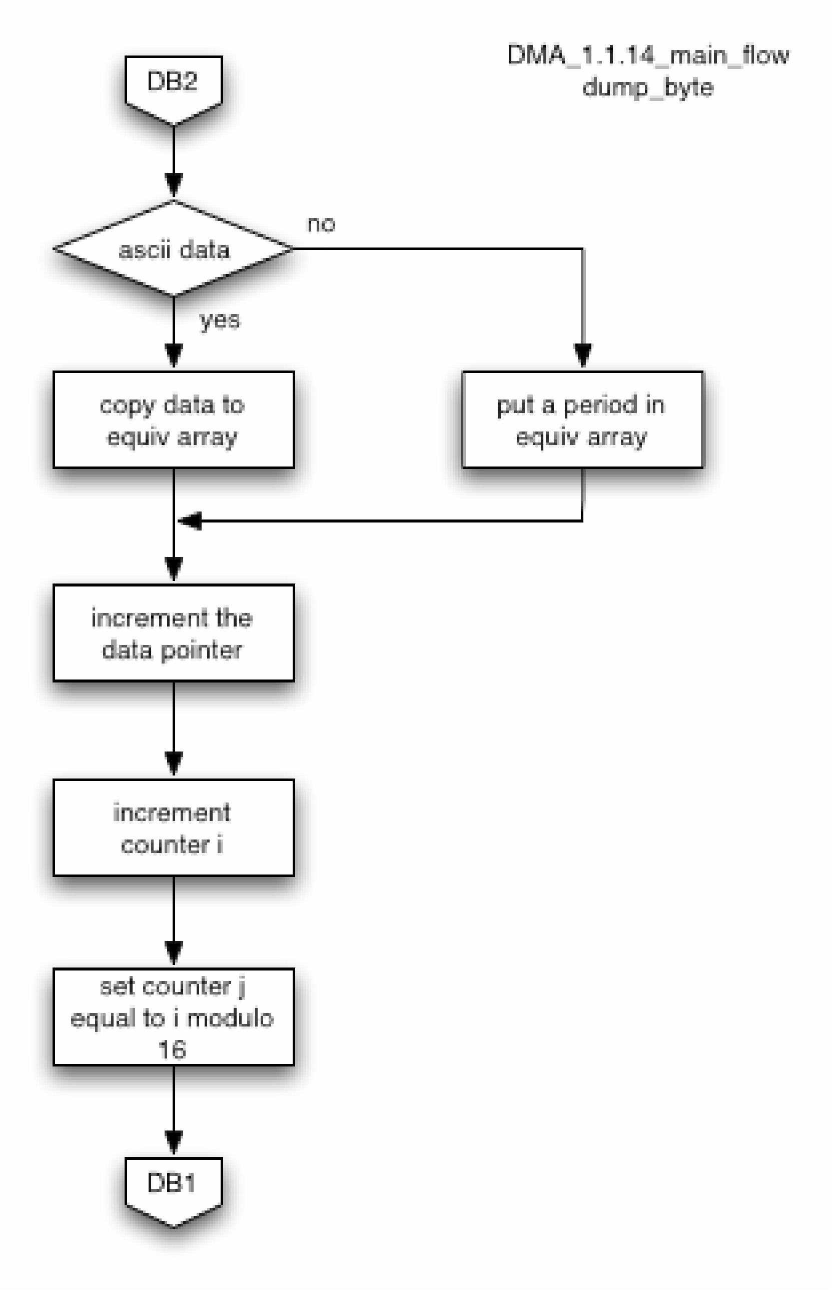


Figure 51. DMA_1.1.15_main_flow

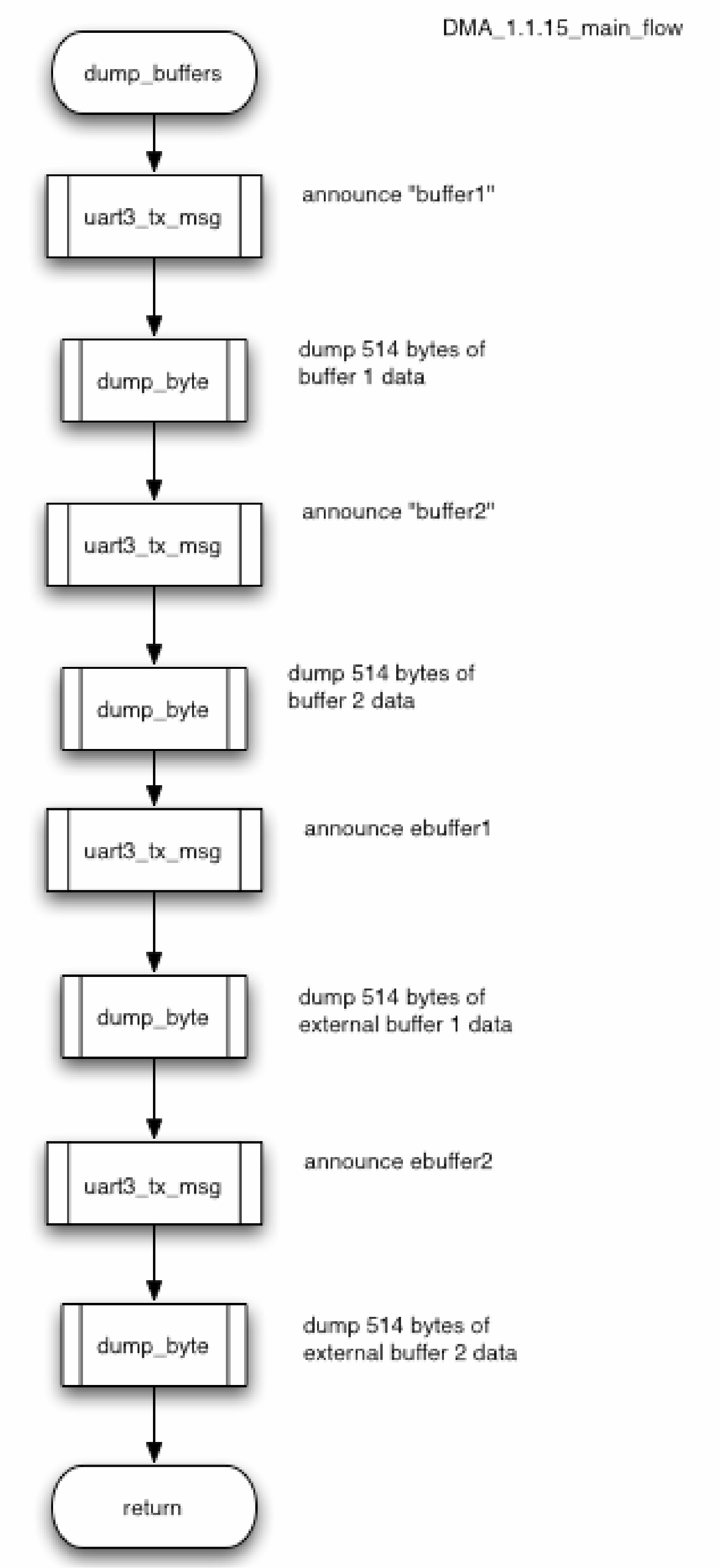


Figure 52. DMA_1.1.16_main_flow

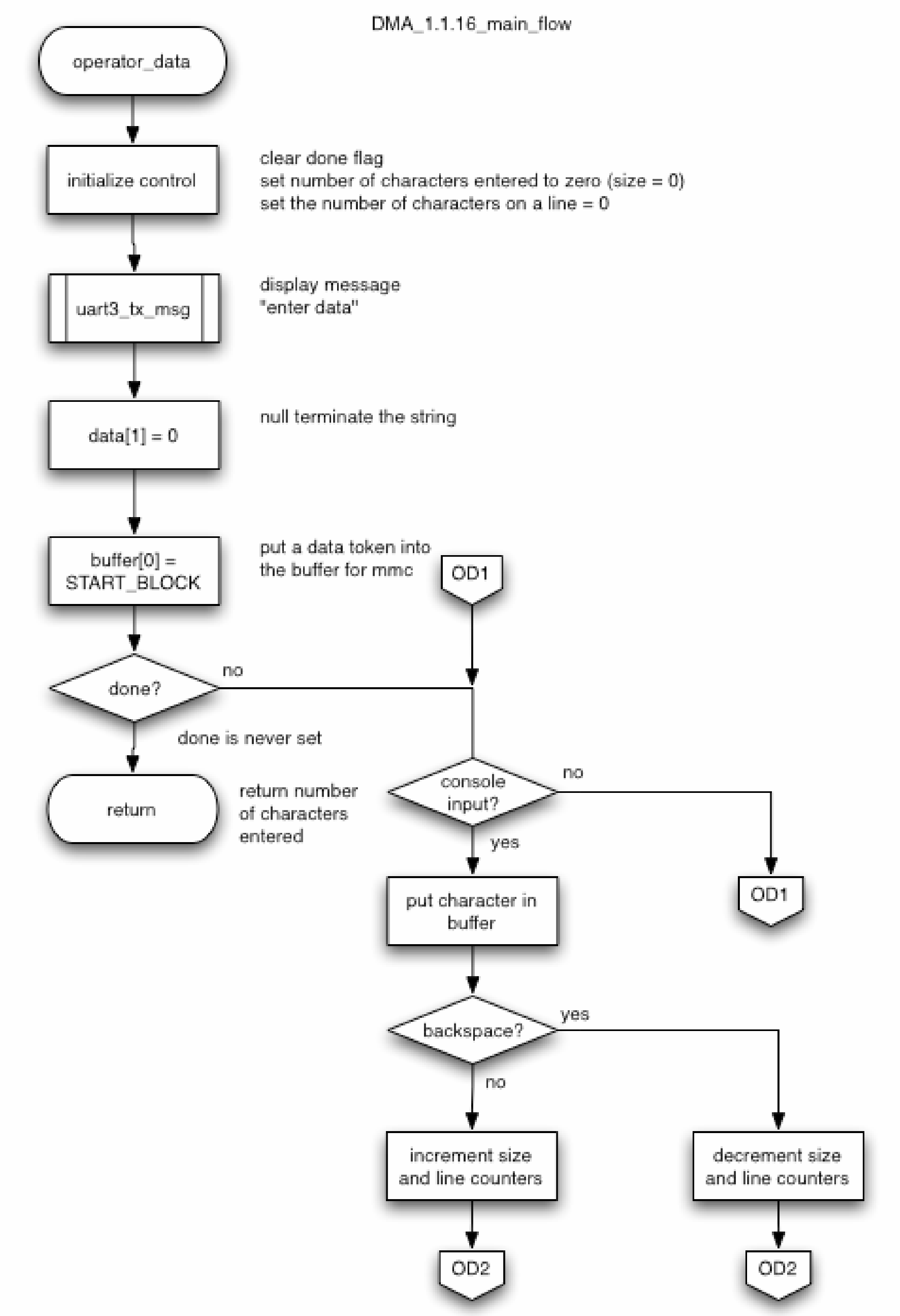


Figure 53. DMA_1.1.17_main_flow

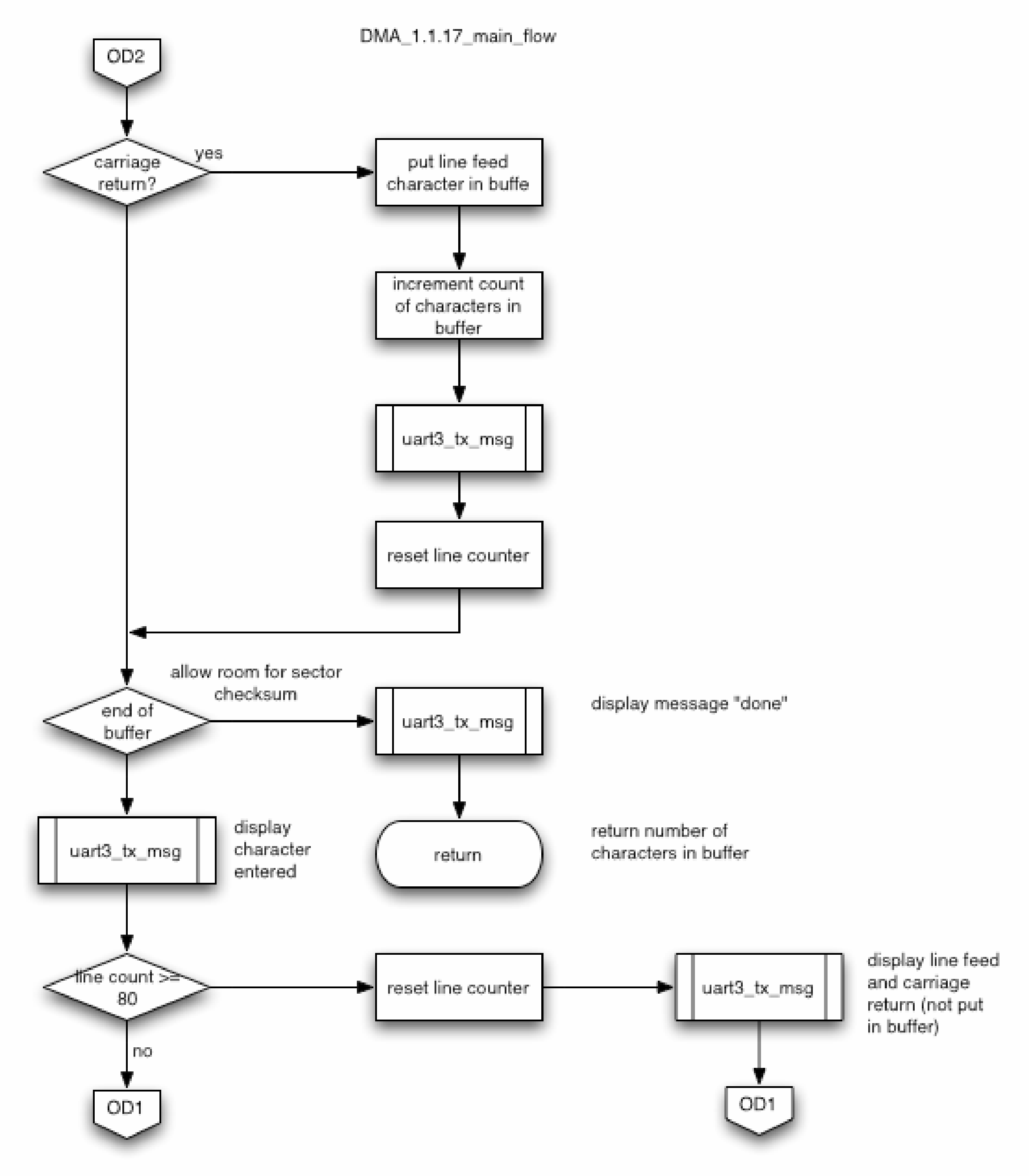

DMA_1.1.17_main_flow
OD2
carriage return?
yes
put line feed character in buffe
increment count of characters in buffer
uart3_tx_msg
reset line counter
allow room for sector checksum
end of buffer
uart3_tx_msg
display message "done"
uart3_tx_msg
display character entered
return
return number of characters in buffer
line count >= 80
reset line counter
uart3_tx_msg
display line feed and carriage return (not put in buffer)
no
OD1
OD1

Figure 54. DMA_1.1.18_main_flow

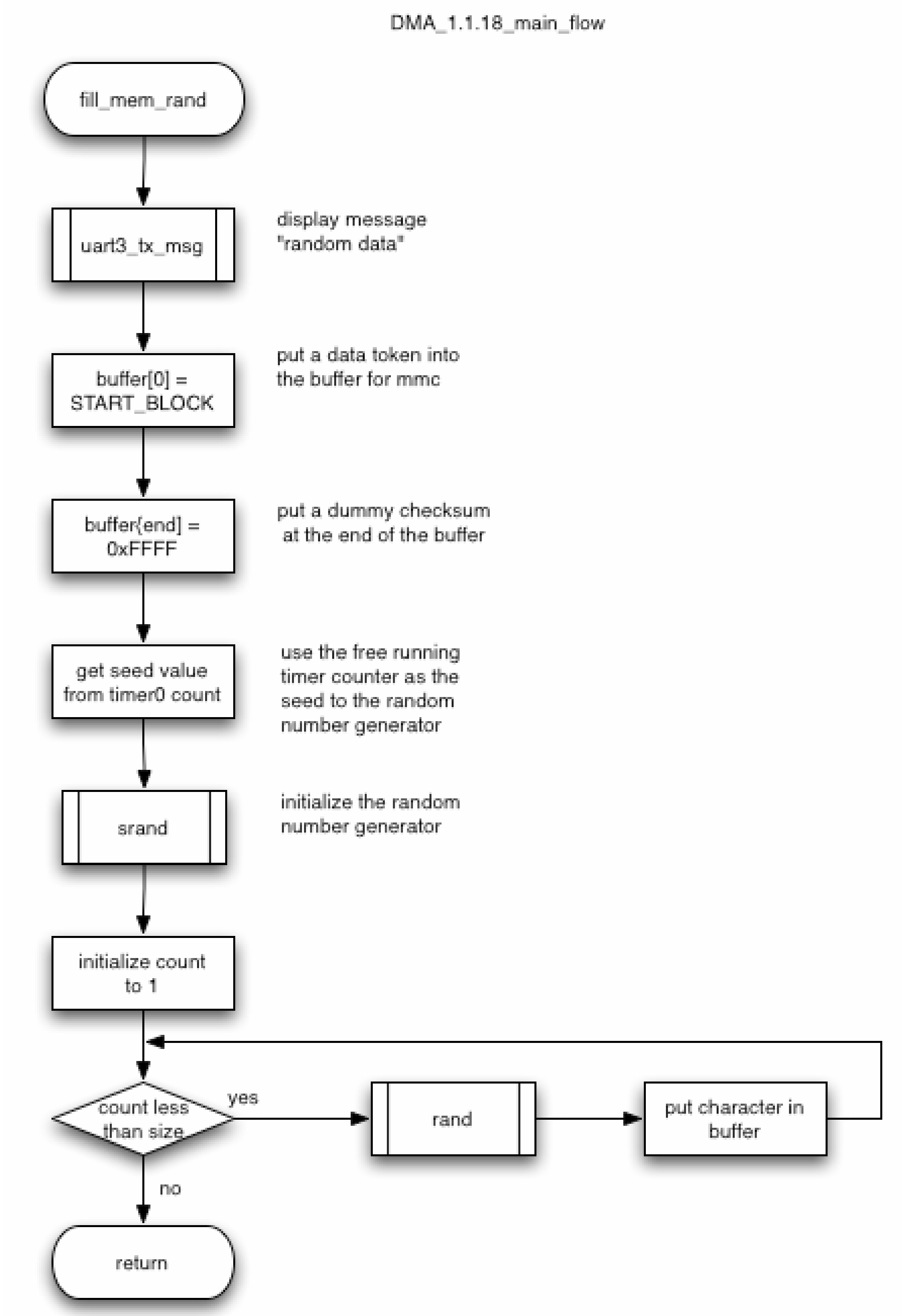
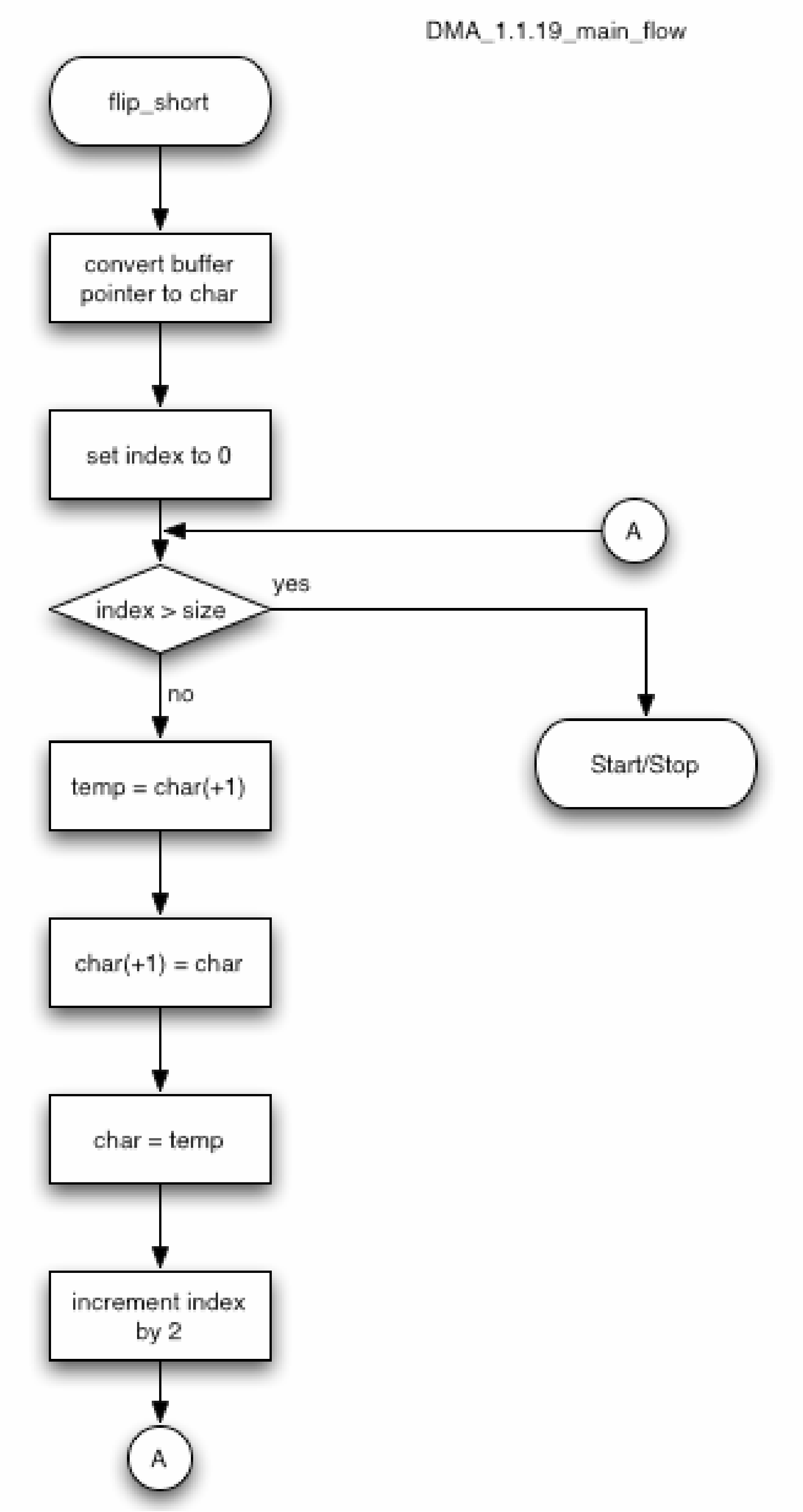

DMA_1.1.18_main_flow
fill_mem_rand
uart3_tx_msg
display message "random data"
buffer[0] = START_BLOCK
put a data token into the buffer for mmc
buffer{end] = 0xFFFF
put a dummy checksum at the end of the buffer
get seed value from timer0 count
use the free running timer counter as the seed to the random number generator
srand
initialize the random number generator
initialize count to 1
count less than size
yes
rand
put character in buffer
no
return

Figure 55. DMA_1.1.19_main_flow



8.3 SD Memory Flow

Figure 56. DMA_2.0.0_sdmemory_flow

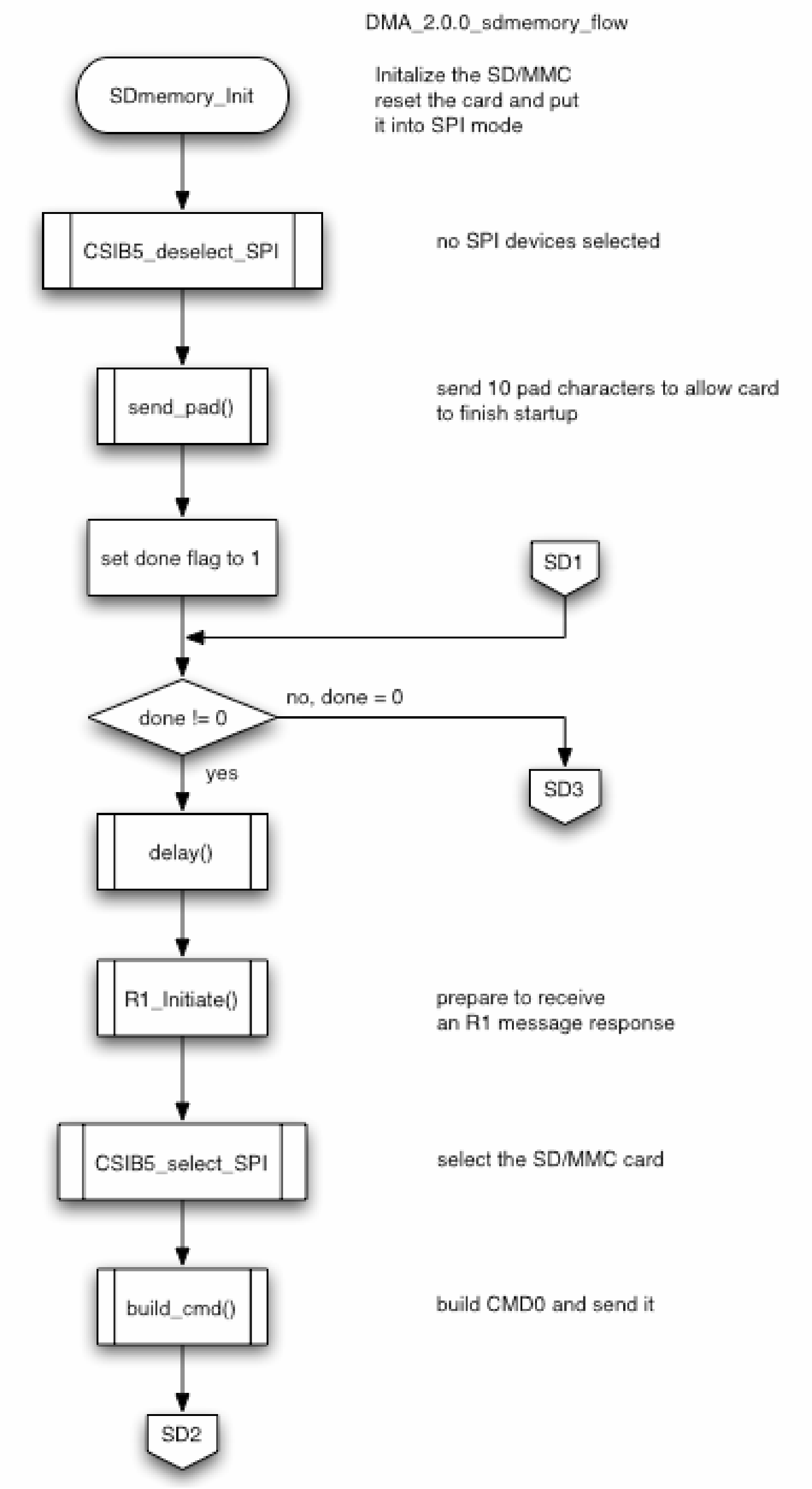


Figure 57. DMA_2.0.1_sdmemory_flow

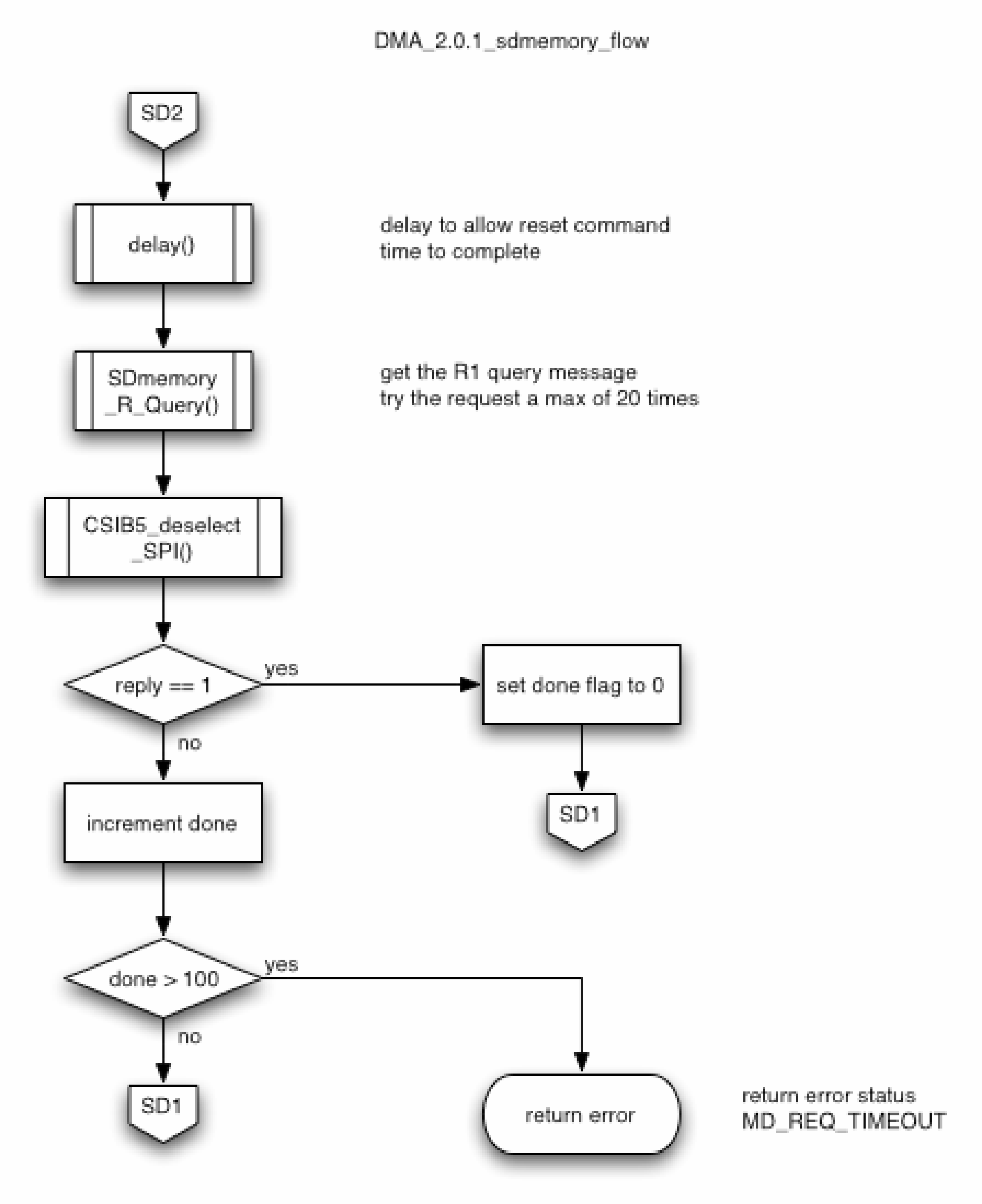


Figure 58.   DMA_2.0.2_sdmemory_flow

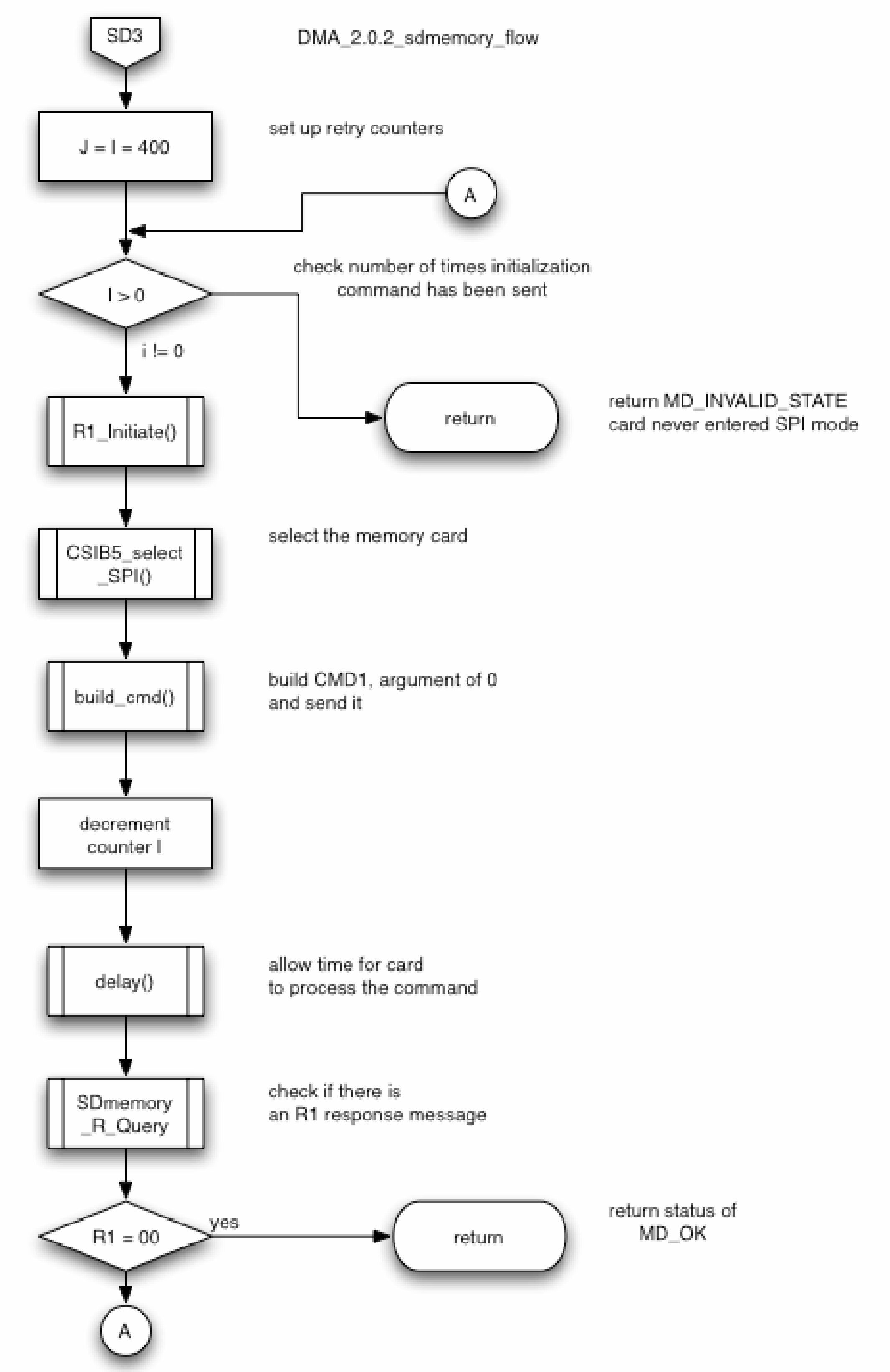

SD3
DMA_2.0.2_sdmemory_flow
J = I = 400
set up retry counters
A
I > 0
check number of times initialization command has been sent
i != 0
R1_Initiate()
return
return MD_INVALID_STATE card never entered SPI mode
CSIB5_select _SPI()
select the memory card
build_cmd()
build CMD1, argument of 0 and send it
decrement counter I
delay()
allow time for card to process the command
SDmemory _R_Query
check if there is an R1 response message
R1 = 00
yes
return
return status of MD_OK
A

Figure 59.   DMA_2.0.3_sdmemory_flow

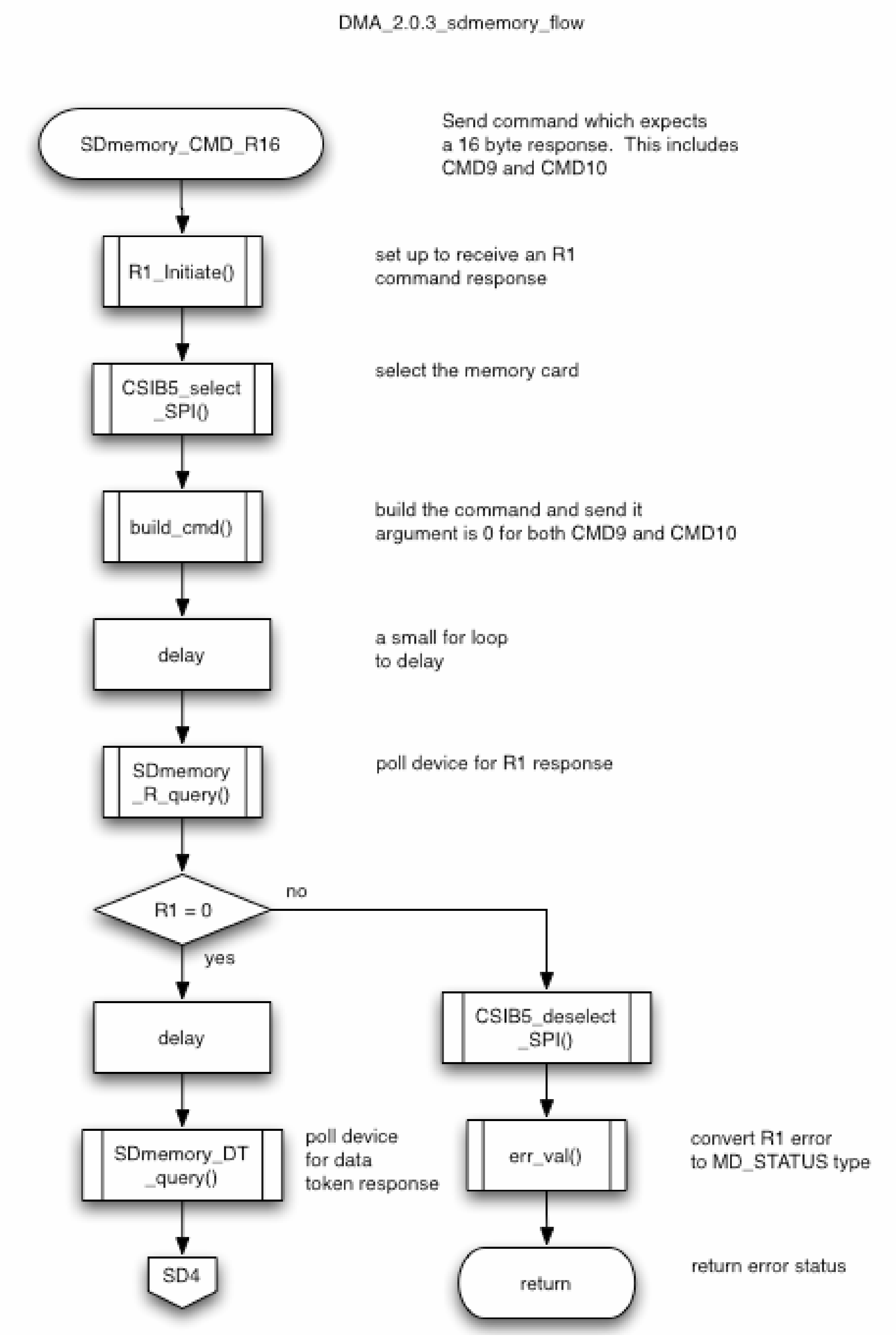


Figure 60. DMA_2.0.4_sdmemory_flow

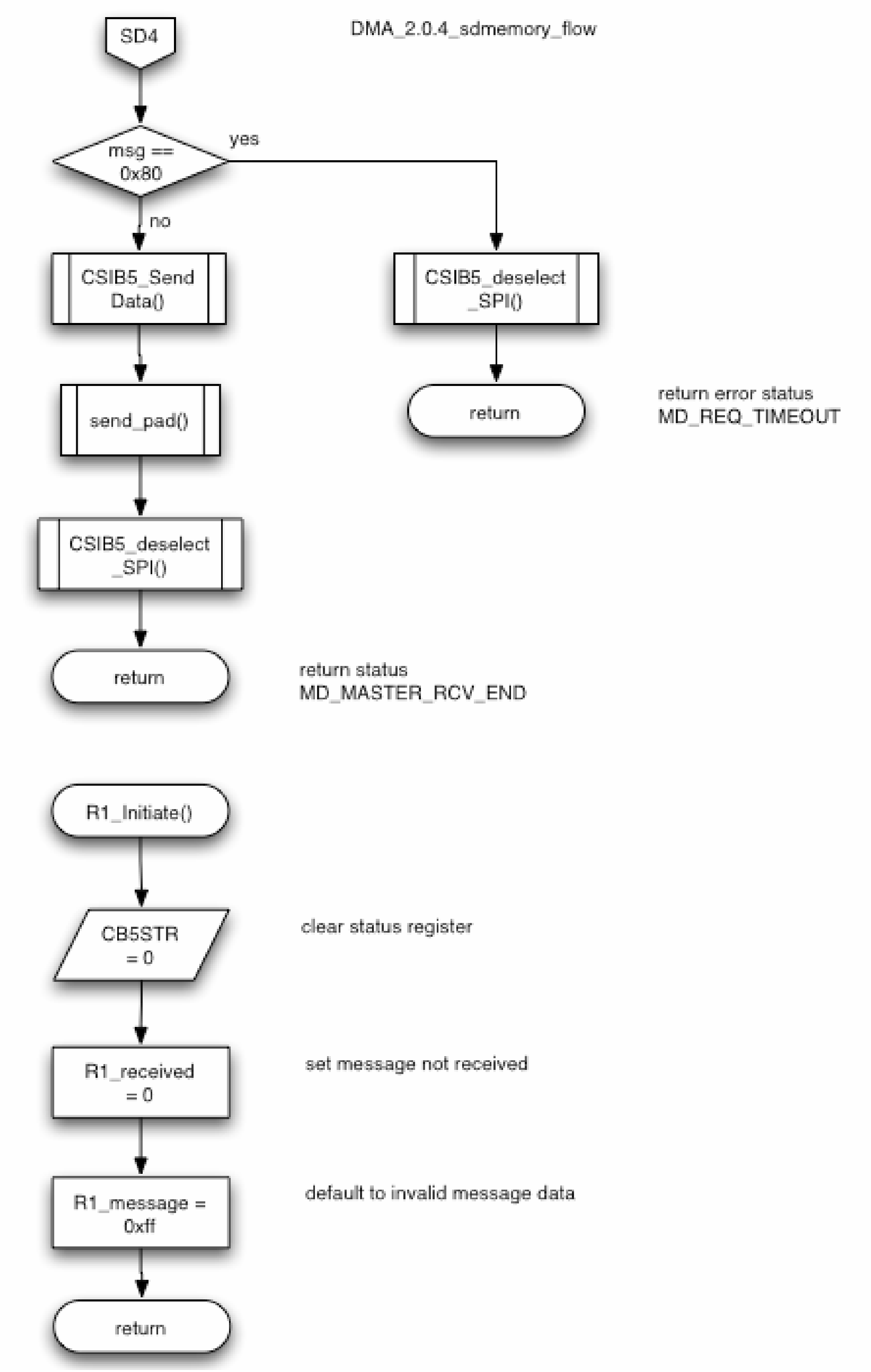


Figure 61. DMA_2.0.5_sdmemory_flow

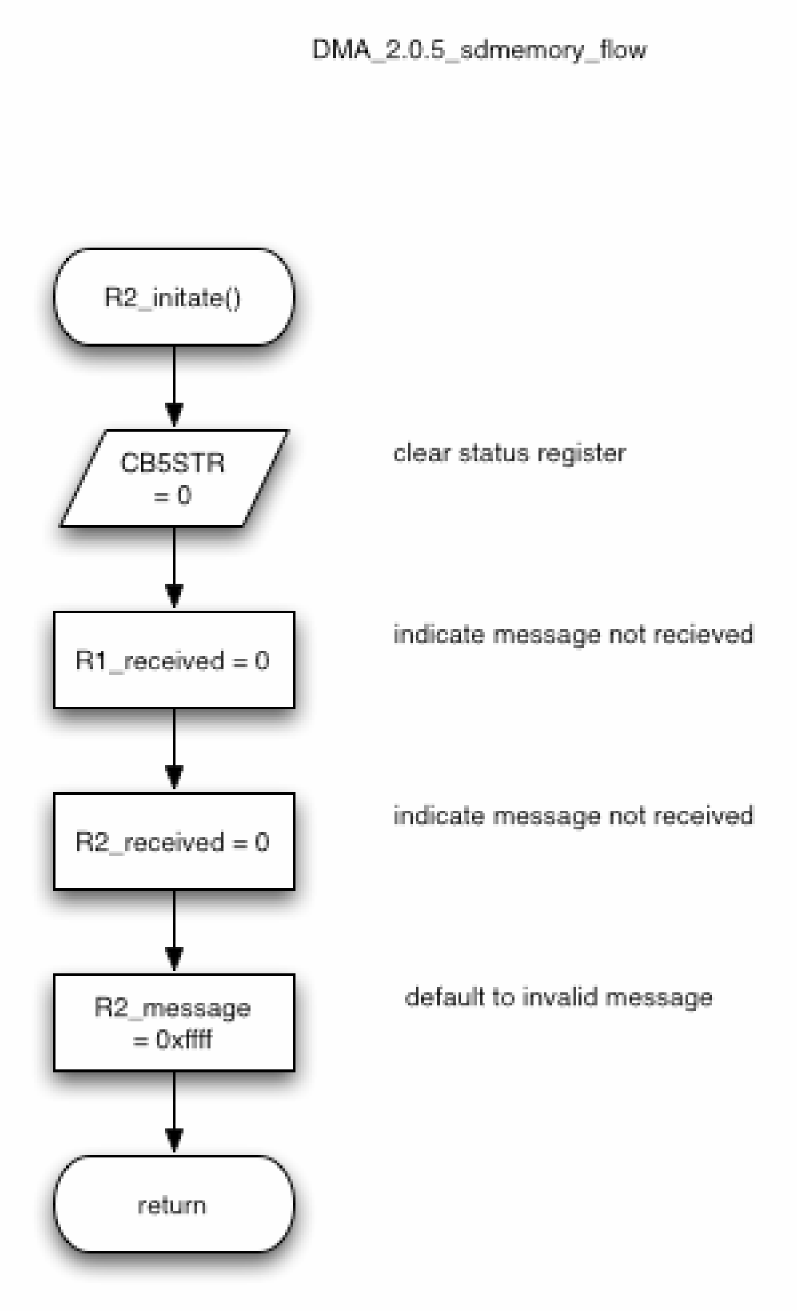


Figure 62. DMA_2.0.6_sdmemory_flow

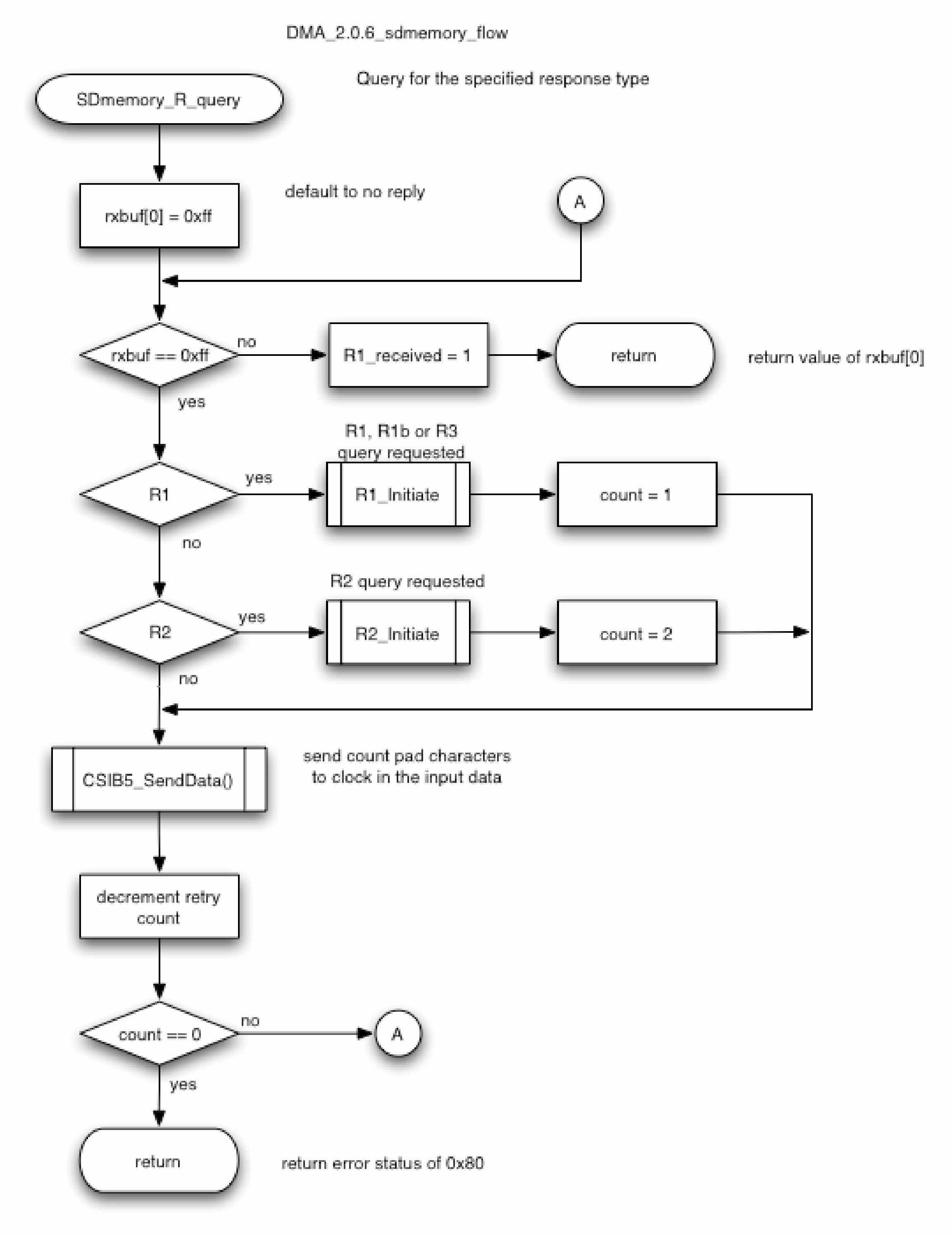


Figure 63. DMA_2.0.7_sdmemory_flow

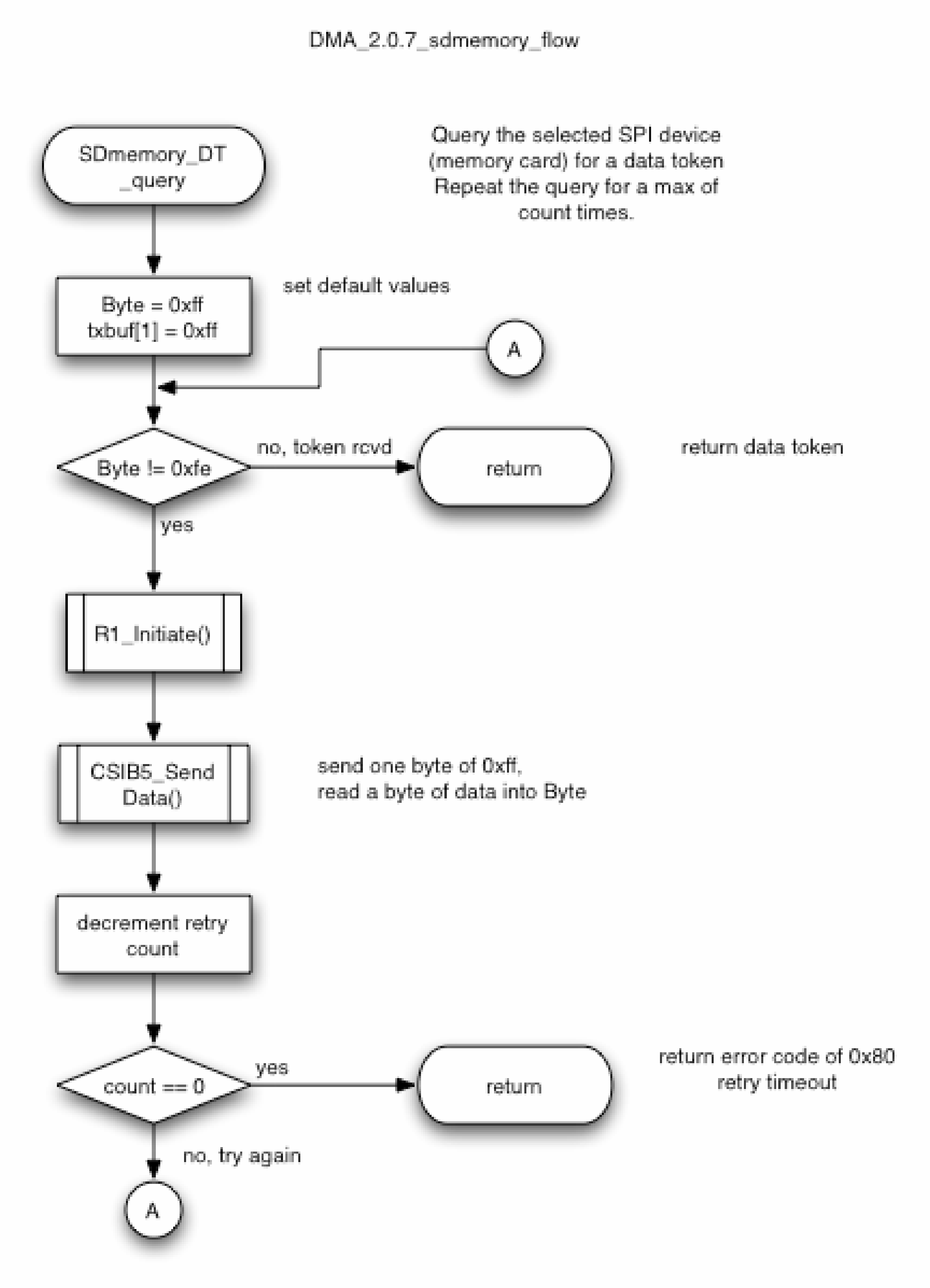


Figure 64.   DMA_2.0.8_sdmemory_flow

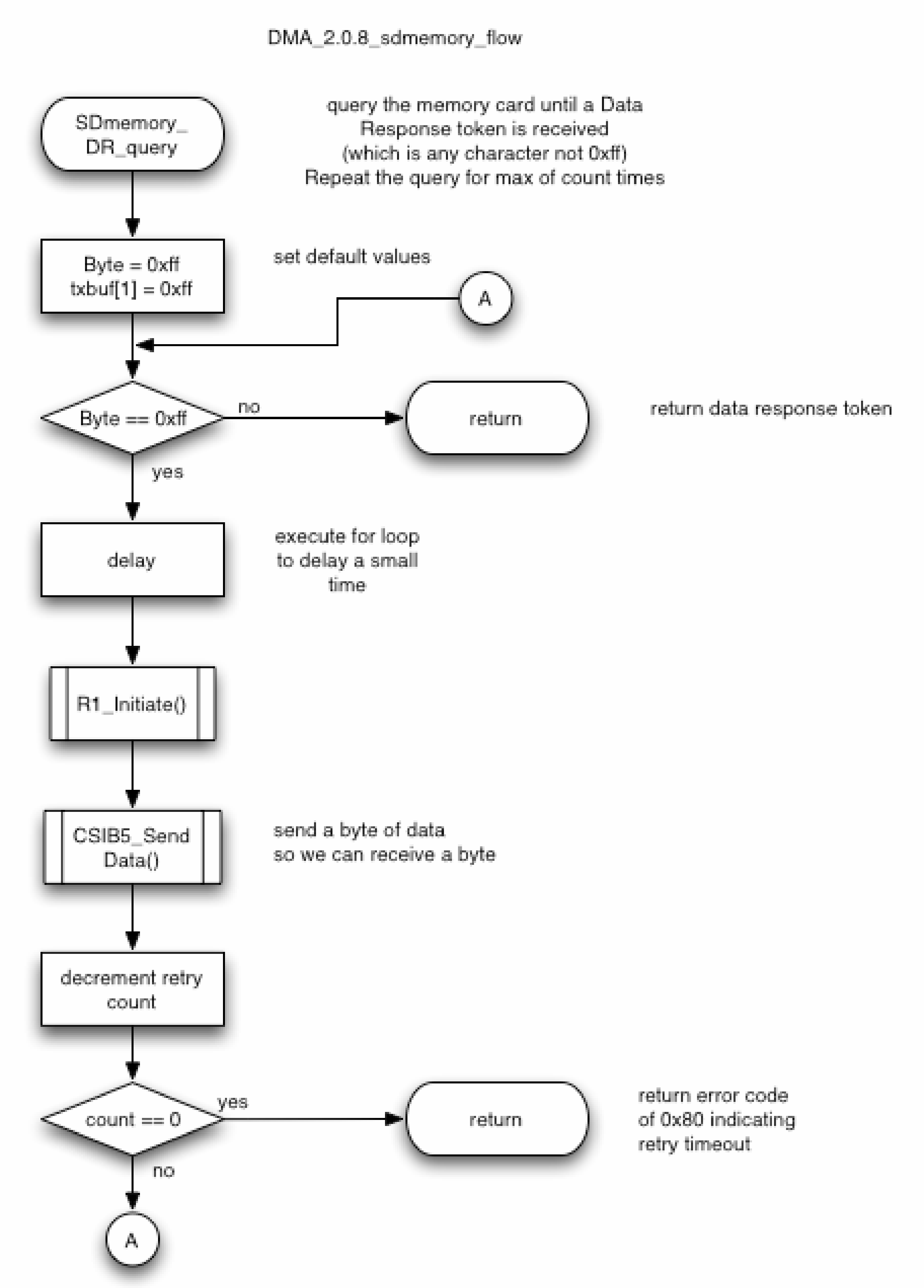

DMA_2.0.8_sdmemory_flow
SDmemory_ DR_query
query the memory card until a Data Response token is received (which is any character not 0xff) Repeat the query for max of count times
Byte = 0xff
txbuf[1] = 0xff
set default values
A
Byte == 0xff
no
return
return data response token
yes
delay
execute for loop to delay a small time
R1_Initiate()
CSIB5_Send Data()
send a byte of data so we can receive a byte
decrement retry count
count == 0
yes
return
return error code of 0x80 indicating retry timeout
no
A

Figure 65.   DMA_2.0.9_sdmemory_flow

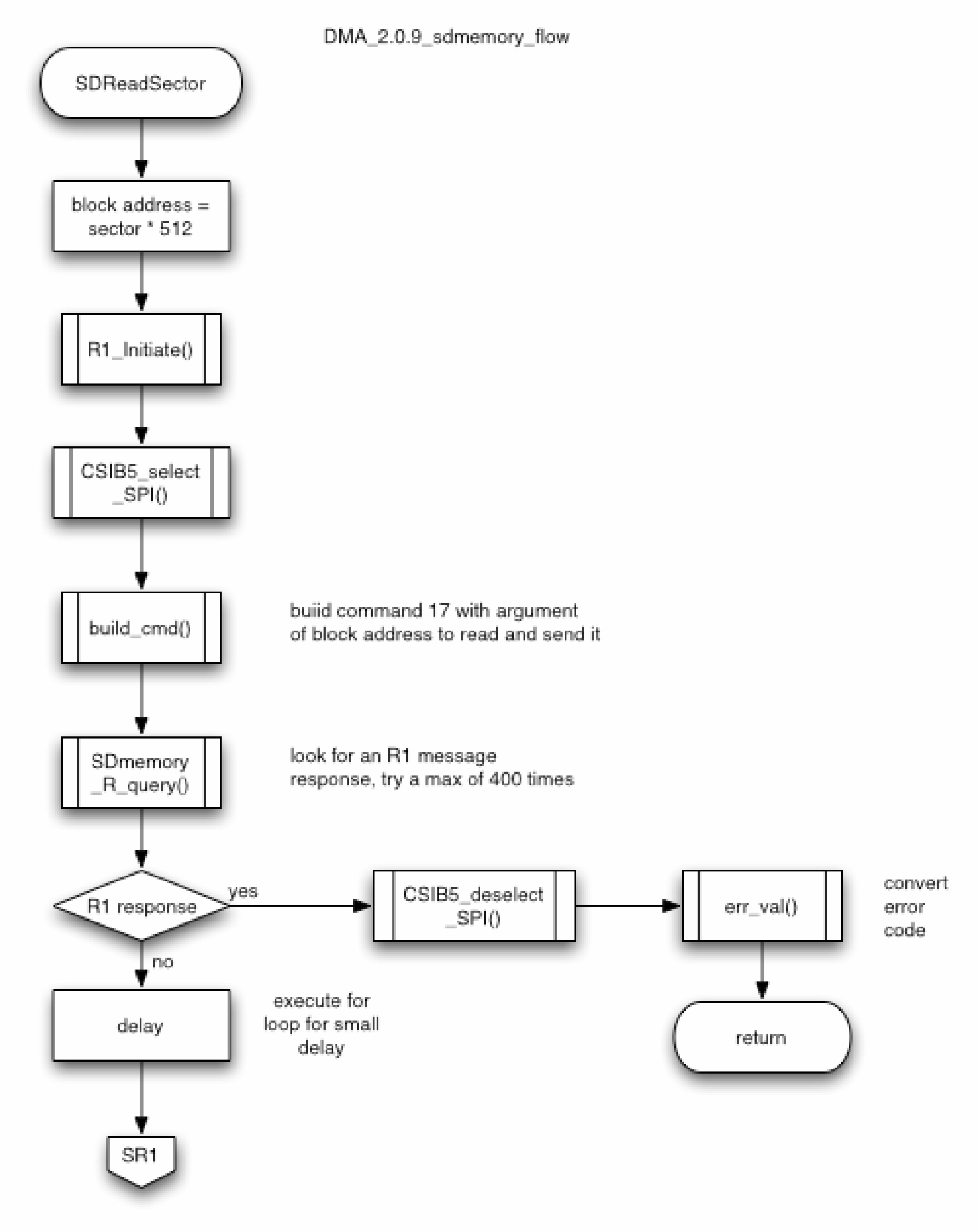

DMA_2.0.9_sdmemory_flow
SDReadSector
block address = sector * 512
R1_Initiate()
CSIB5_select _SPI()
build_cmd()
buiid command 17 with argument of block address to read and send it
SDmemory _R_query()
look for an R1 message response, try a max of 400 times
R1 response
yes
no
CSIB5_deselect _SPI()
err_val()
convert error code
delay
execute for loop for small delay
return
SR1

Figure 66. DMA_2.0.10_sdmemory_flow

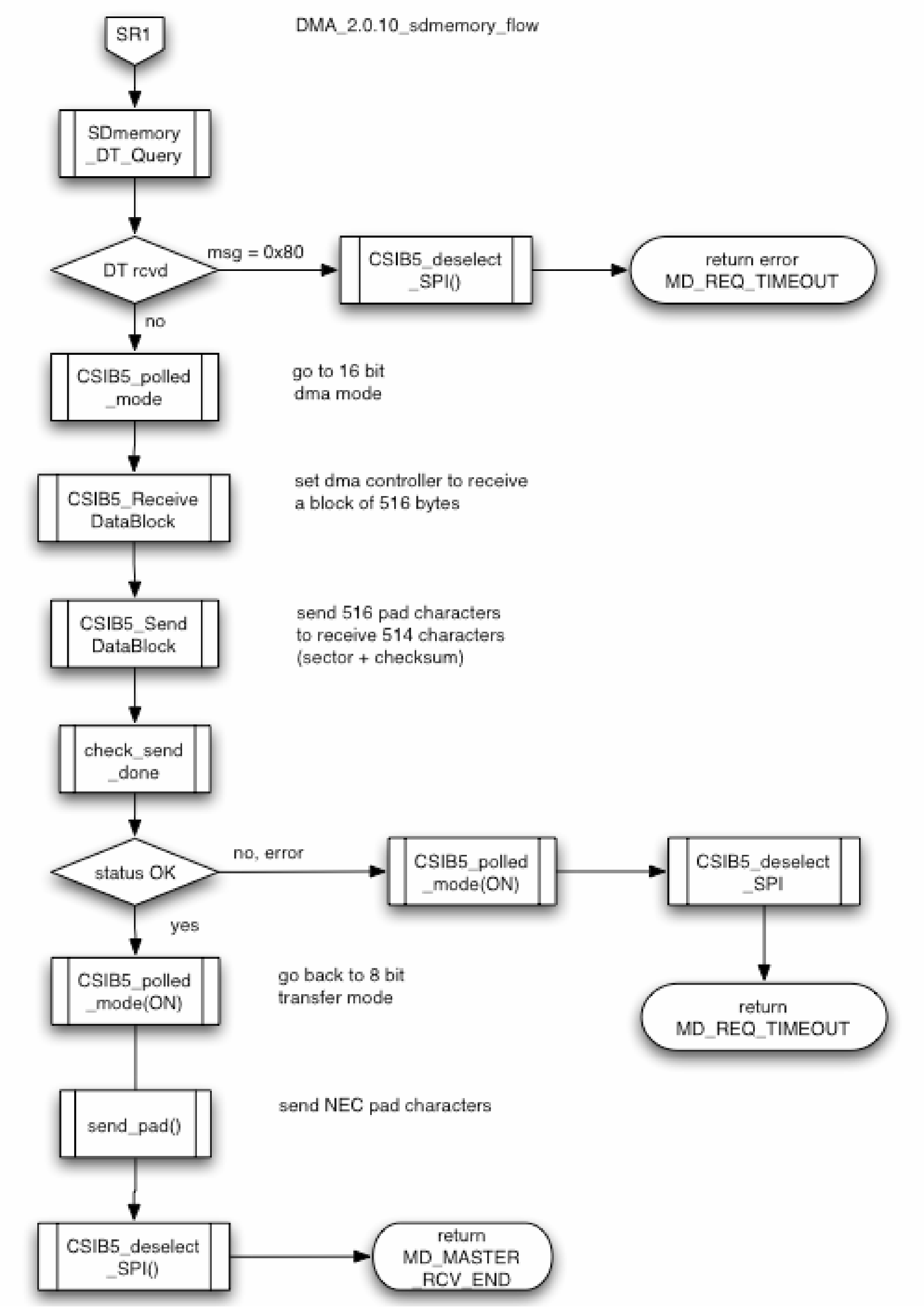


Figure 67. DMA_2.0.11_sdmemory_flow

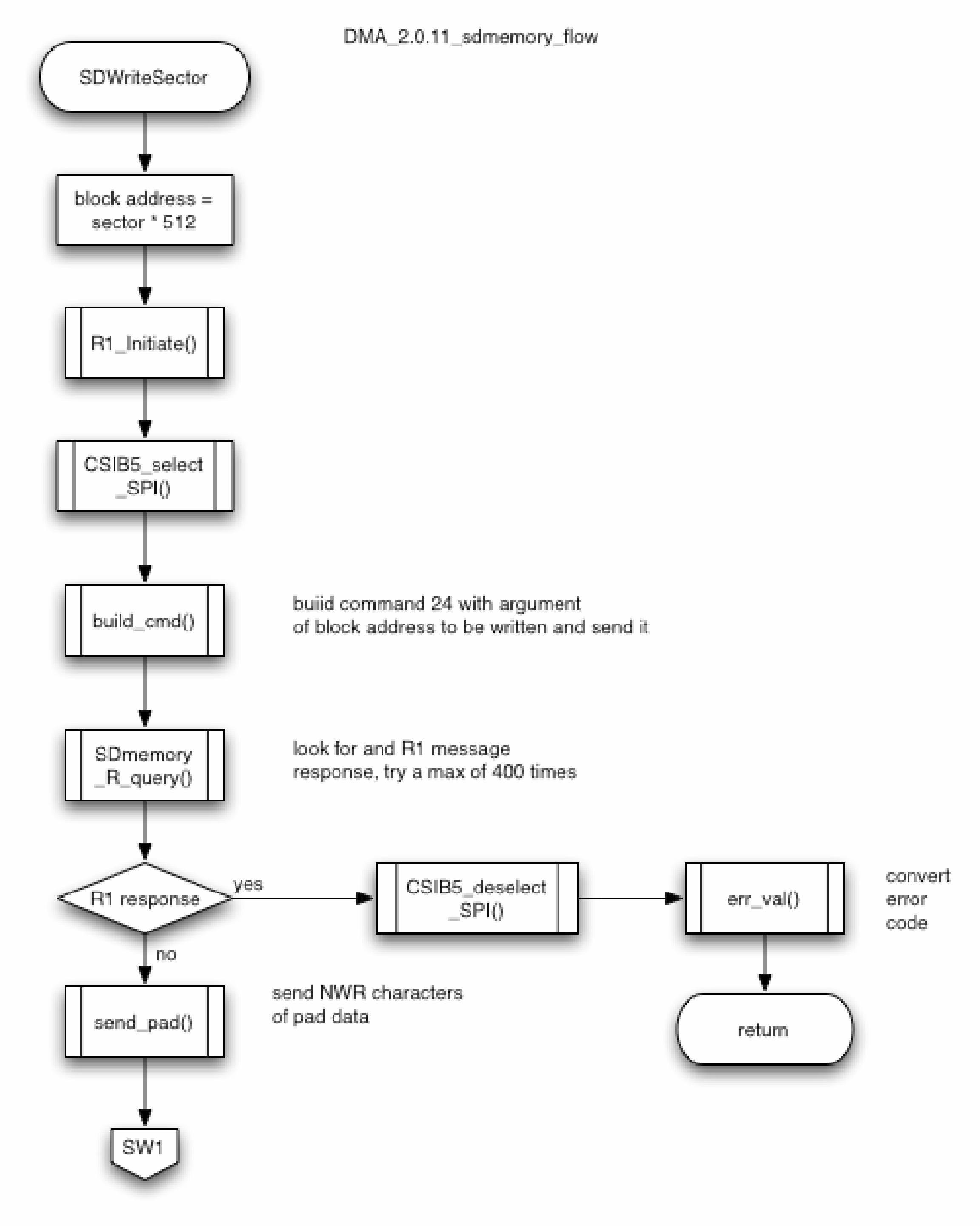

DMA_2.0.11_sdmemory_flow
SDWriteSector
block address = sector * 512
R1_Initiate()
CSIB5_select _SPI()
build_cmd()
buiid command 24 with argument of block address to be written and send it
SDmemory _R_query()
look for and R1 message response, try a max of 400 times
R1 response
yes
CSIB5_deselect _SPI()
err_val()
convert error code
return
no
send_pad()
send NWR characters of pad data
SW1

Figure 68. DMA_2.0.12_sdmemory_flow

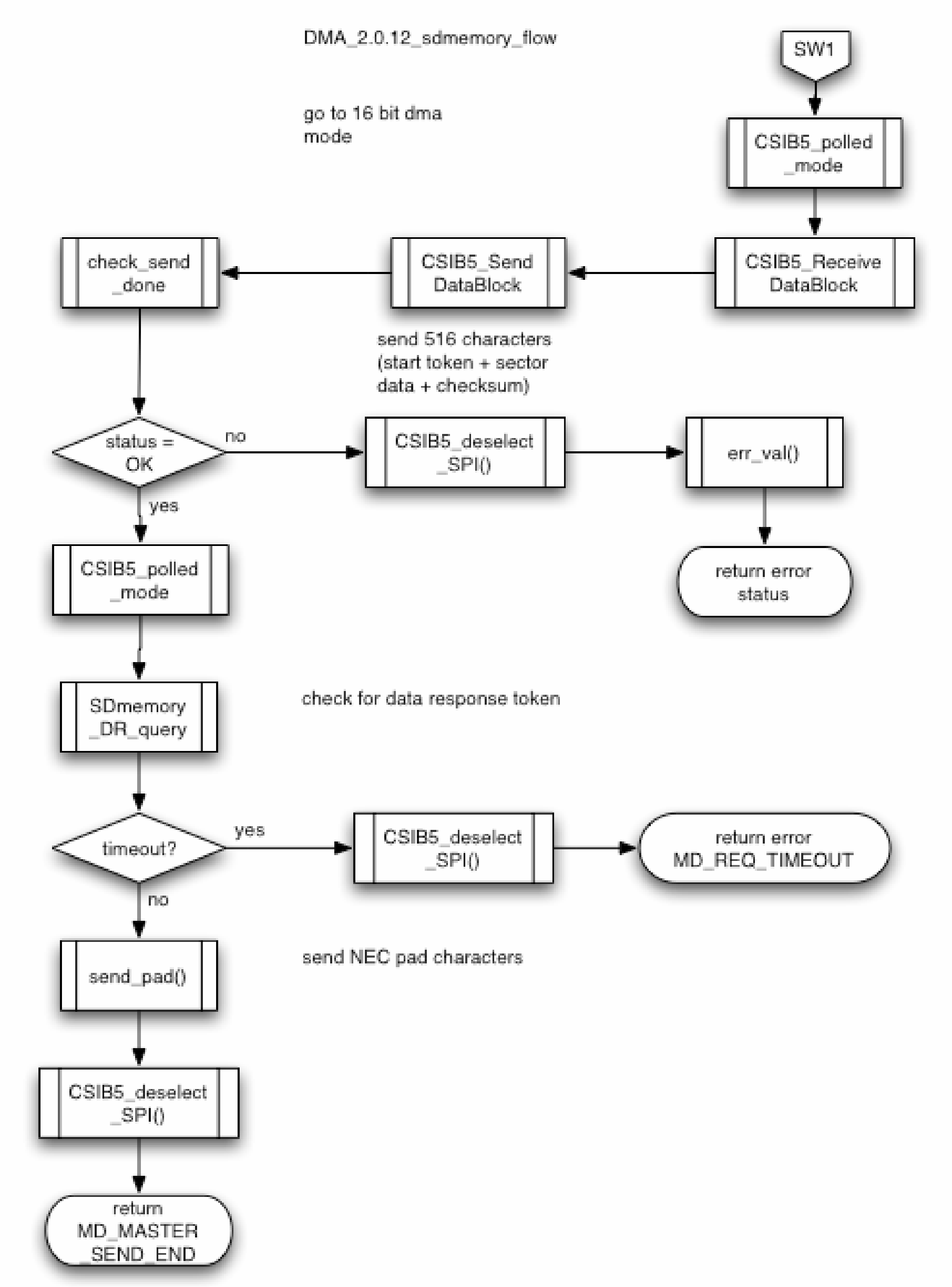


Figure 69. DMA_2.0.13_sdmemory_flow

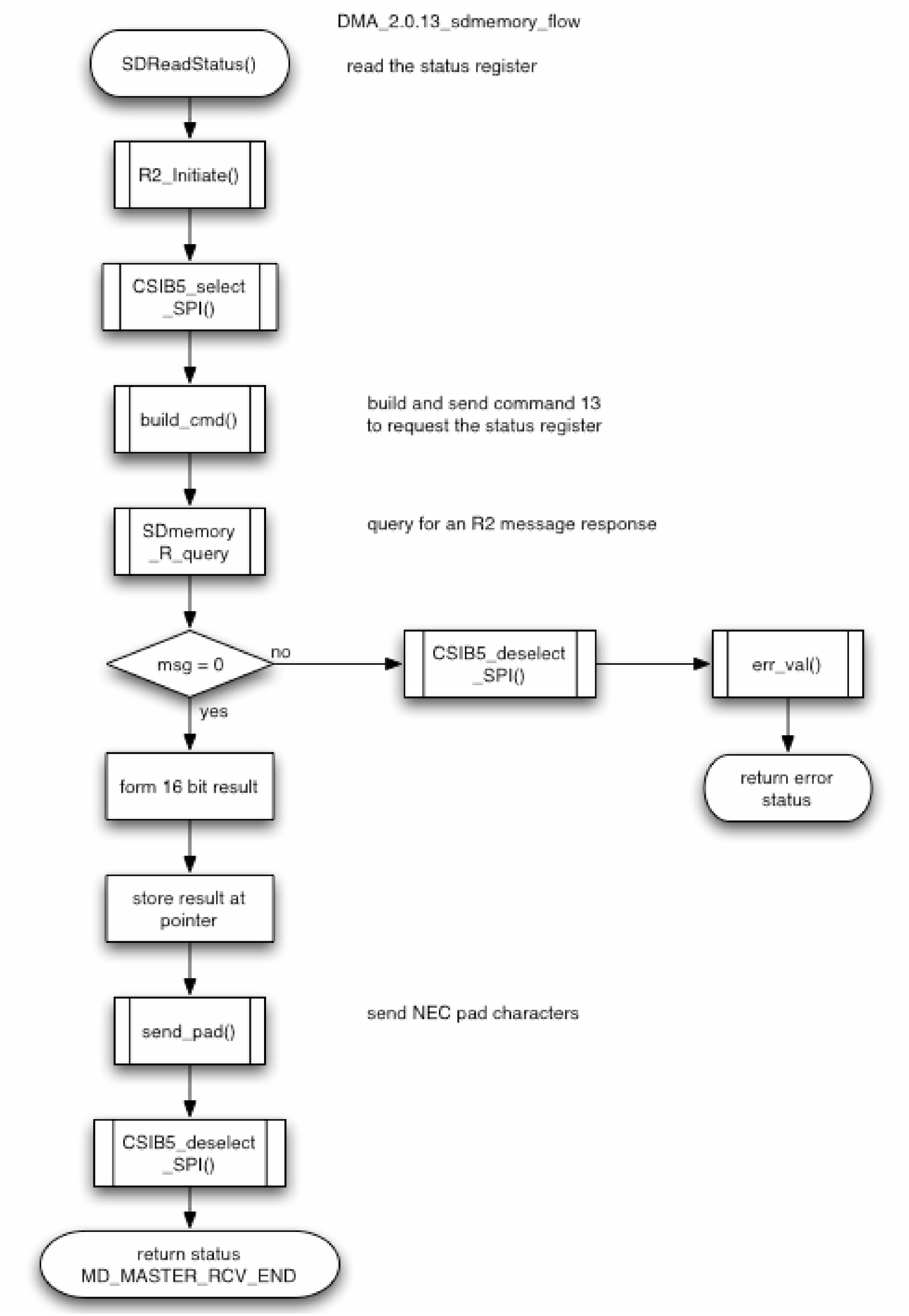


Figure 70. DMA_2.0.14_sdmemory_flow

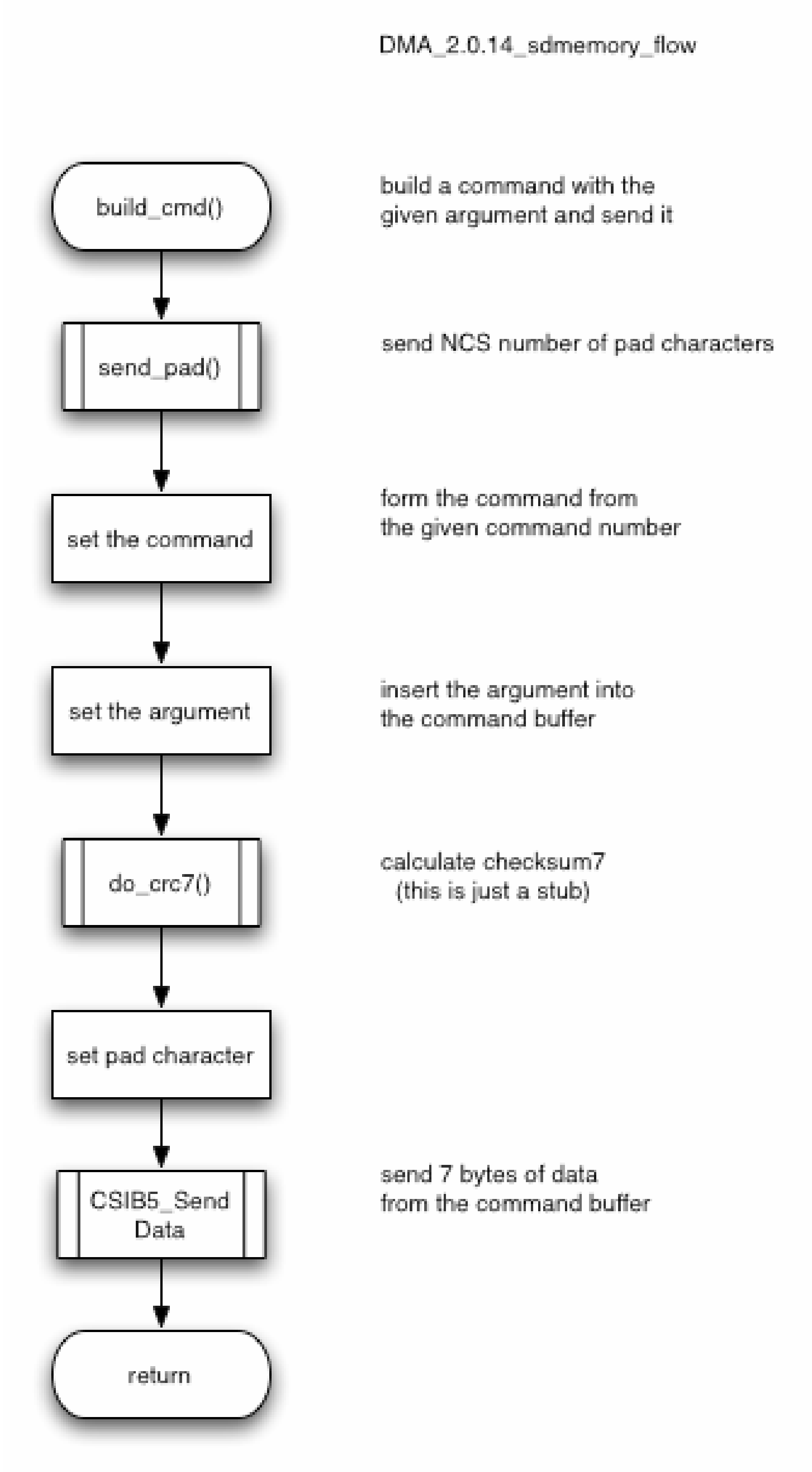

DMA_2.0.14_sdmemory_flow
build_cmd()
build a command with the given argument and send it
send_pad()
send NCS number of pad characters
set the command
form the command from the given command number
set the argument
insert the argument into the command buffer
do_crc7()
calculate checksum7 (this is just a stub)
set pad character
CSIB5_Send Data
send 7 bytes of data from the command buffer
return

Figure 71.   DMA_2.0.15_sdmemory_flow

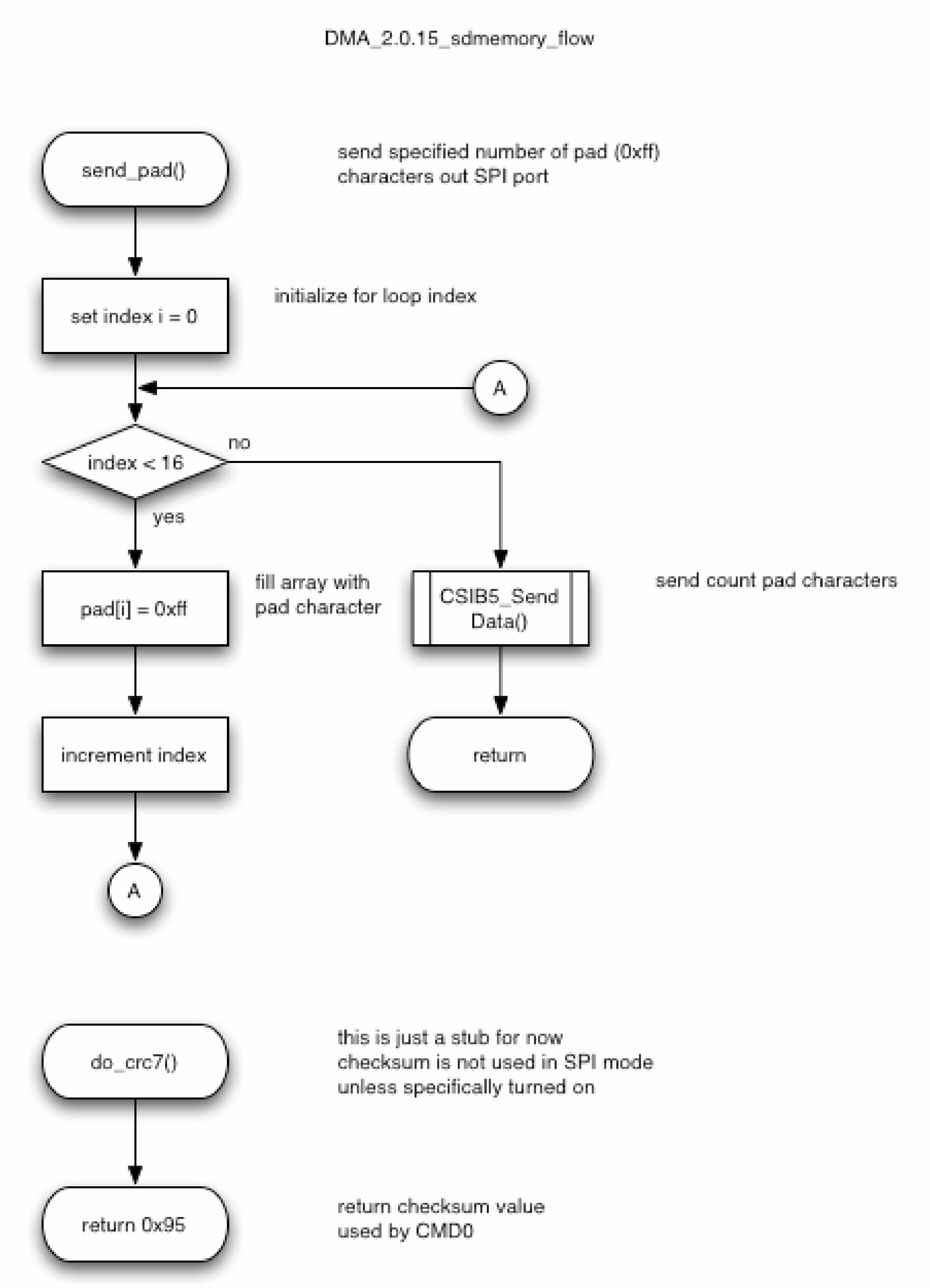


Figure 72. DMA_2.0.16_sdmemory_flow

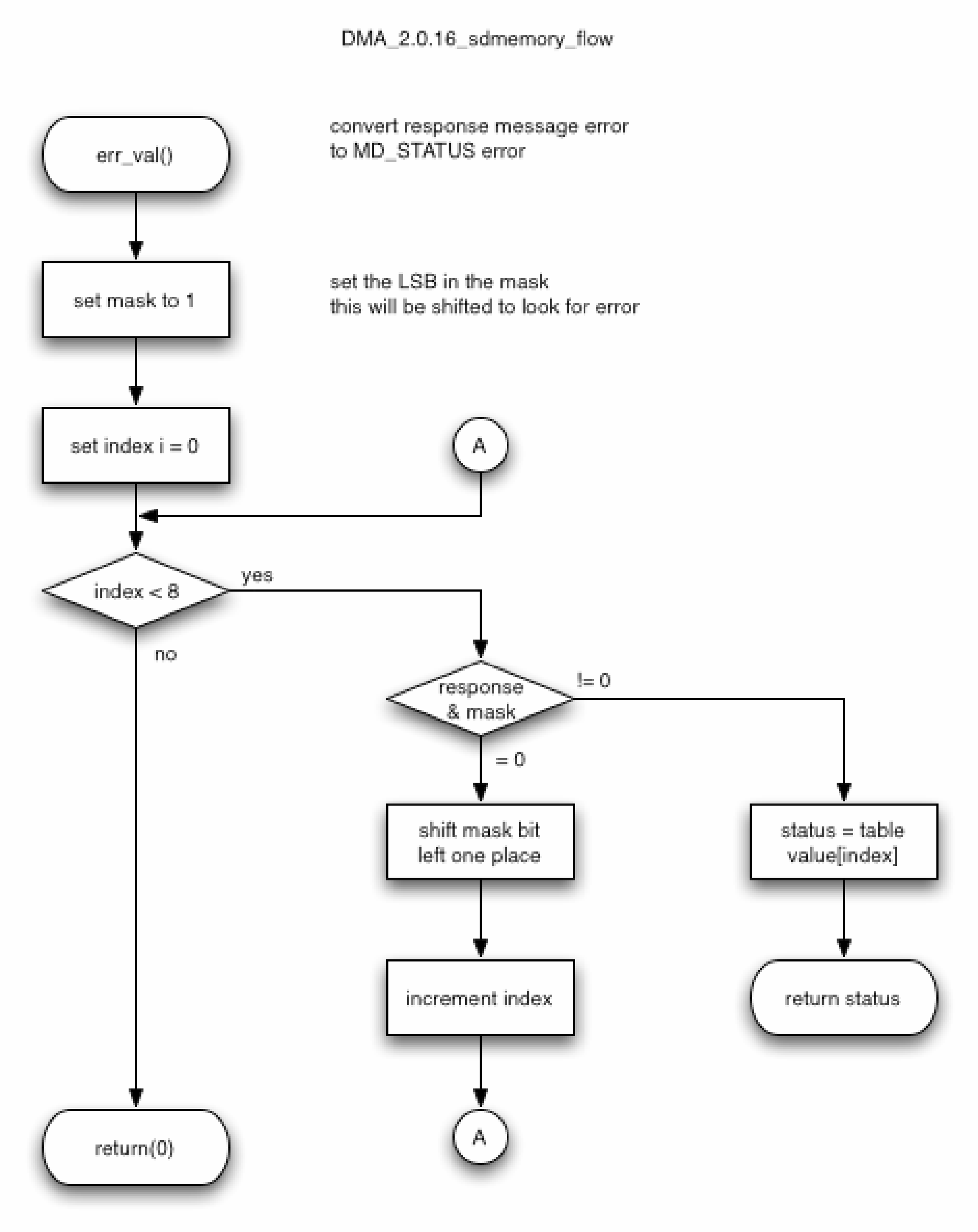


Figure 73. DMA_2.0.17_sdmemory_flow

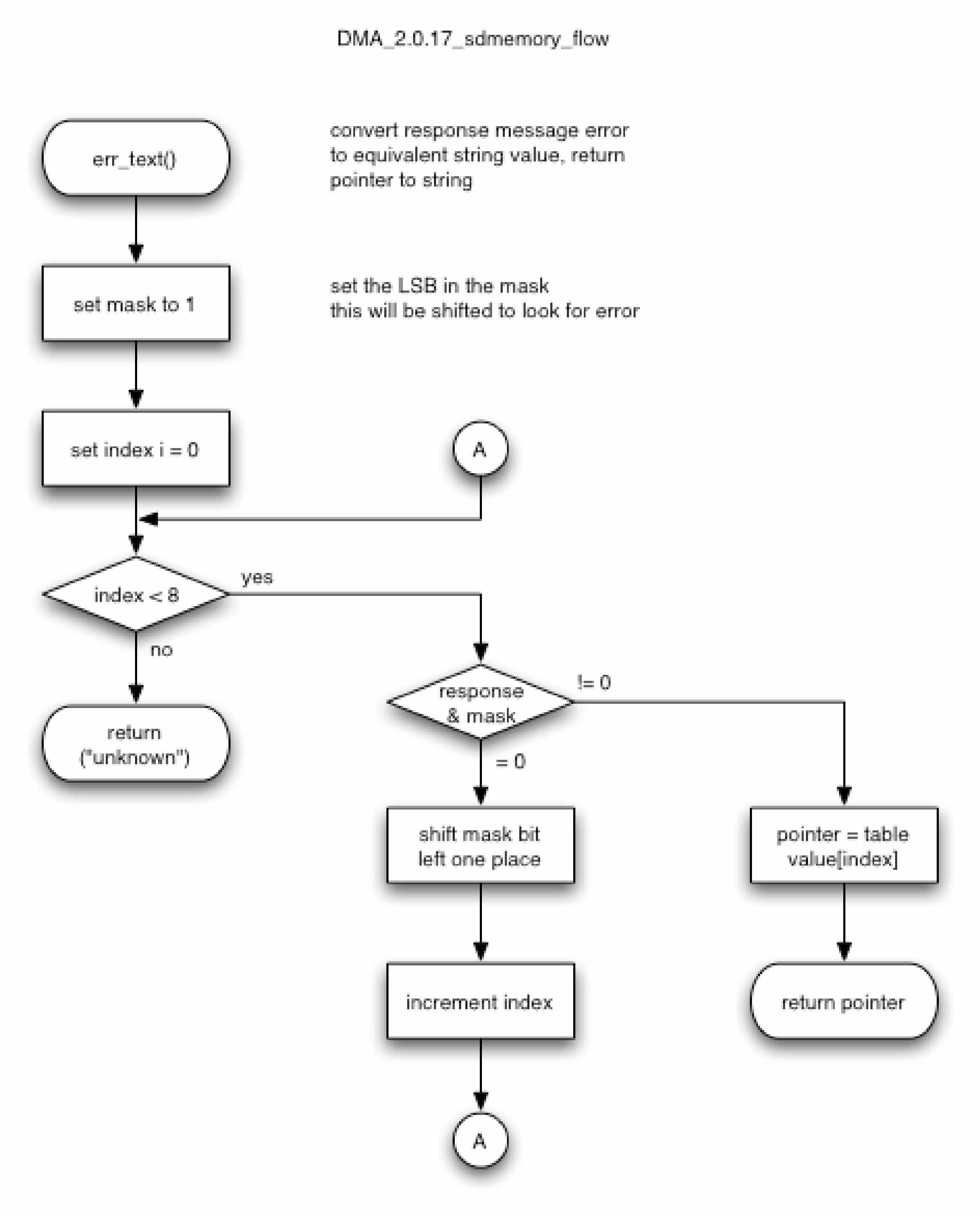

DMA_2.0.17_sdmemory_flow
err_text()
convert response message error to equivalent string value, return pointer to string
set mask to 1
set the LSB in the mask this will be shifted to look for error
set index i = 0
A
index < 8
yes
no
return ("unknown")
response & mask
!= 0
= 0
shift mask bit left one place
pointer = table value[index]
increment index
return pointer
A

Figure 74.   DMA_2.0.18_sdmemory_flow

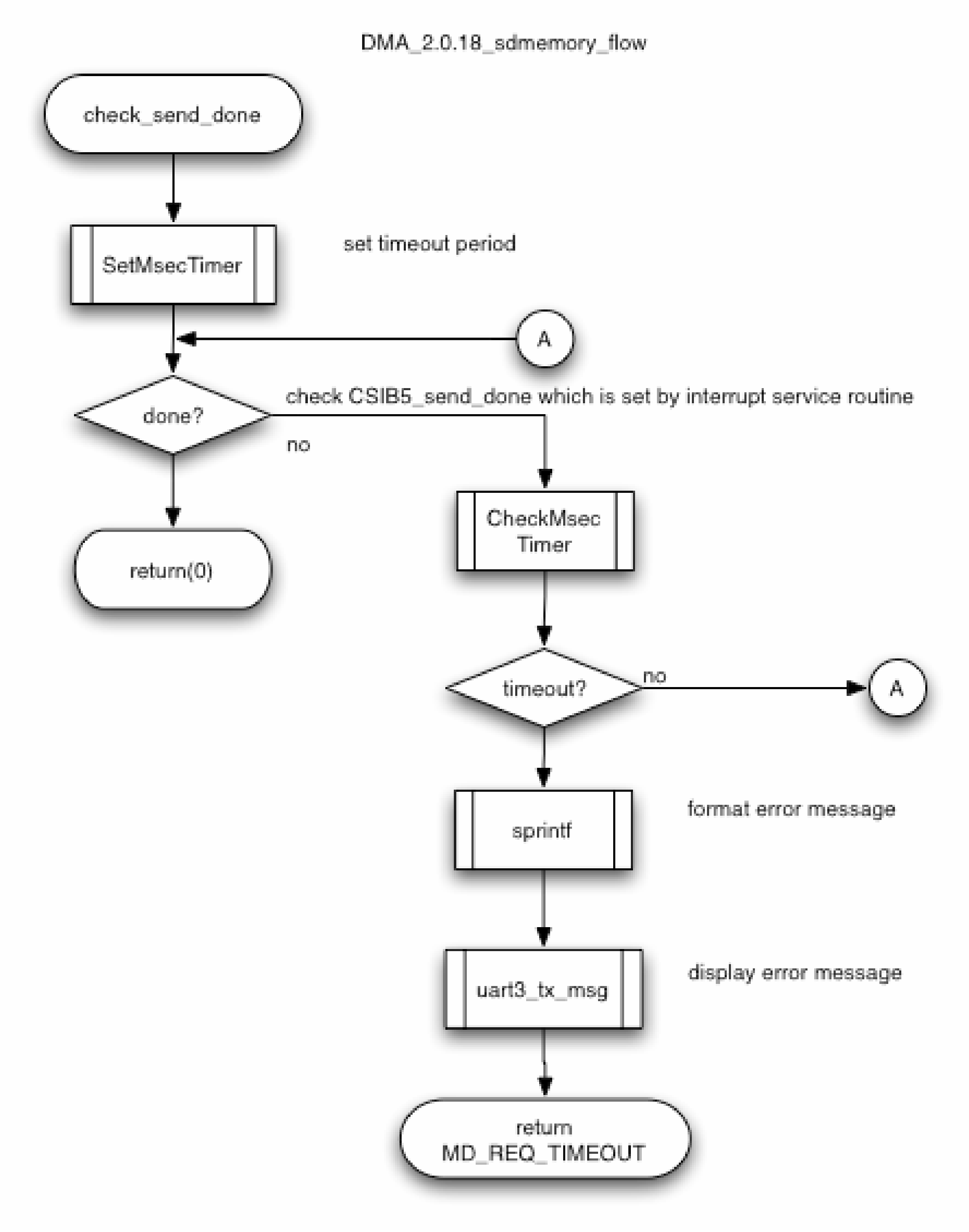
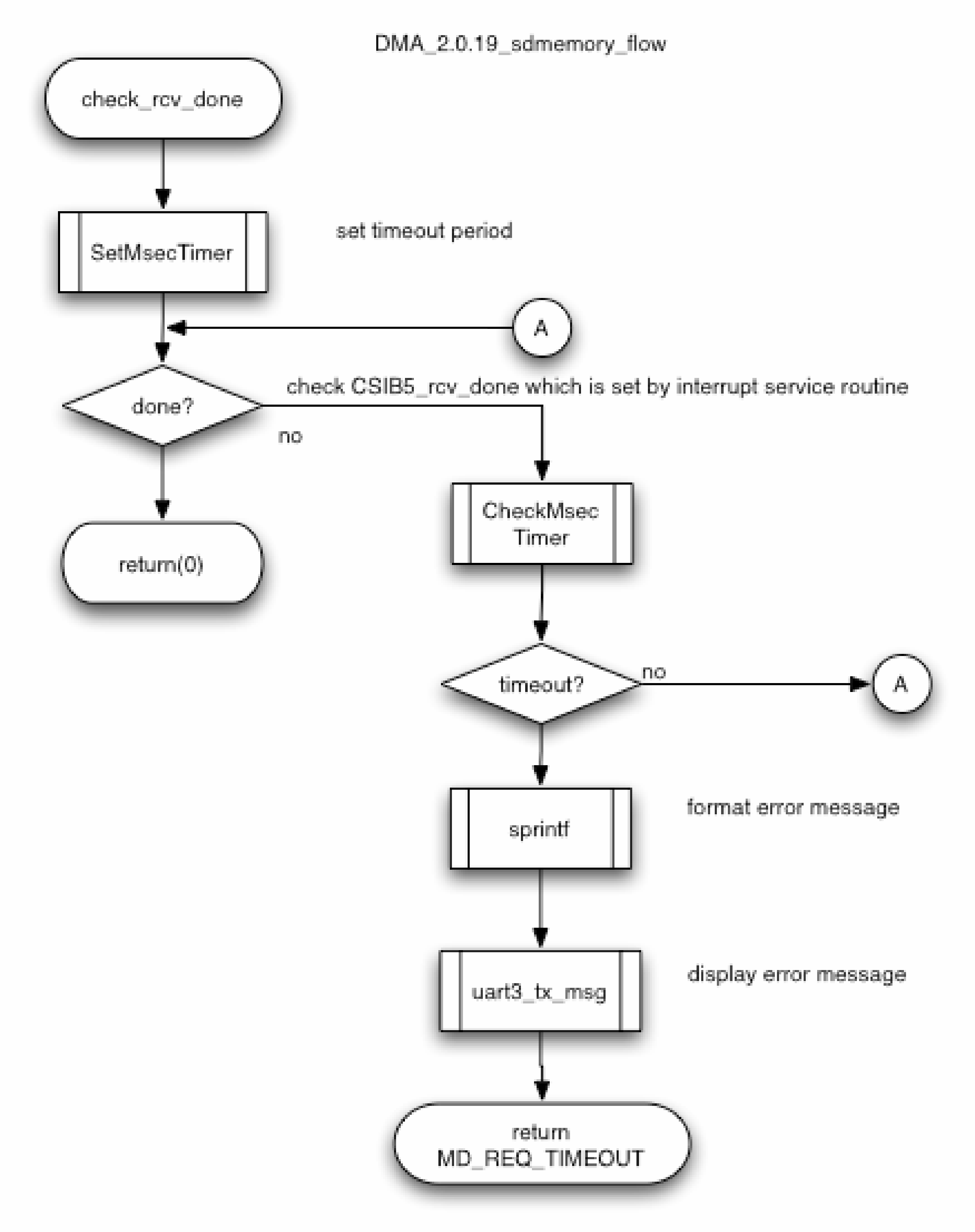


Figure 75. DMA_2.0.19_sdmemory_flow



8.4 Serial Interface

Figure 76. DMA_3.0.0_serial_interface

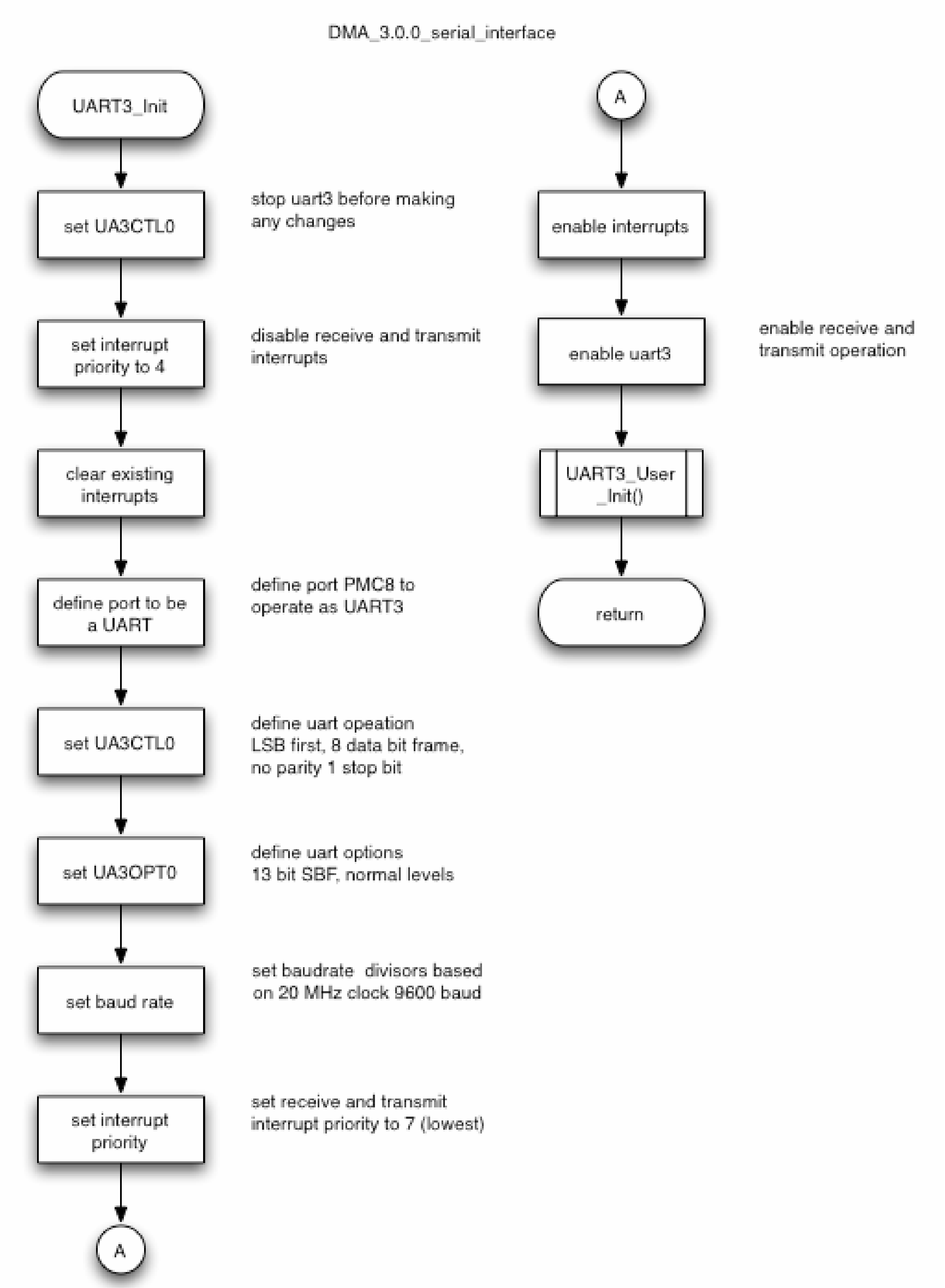

DMA_3.0.0_serial_interface
UART3_Init
set UA3CTL0
stop uart3 before making any changes
set interrupt priority to 4
disable receive and transmit interrupts
clear existing interrupts
define port to be a UART
define port PMC8 to operate as UART3
set UA3CTL0
define uart opeation LSB first, 8 data bit frame, no parity 1 stop bit
set UA3OPT0
define uart options 13 bit SBF, normal levels
set baud rate
set baudrate divisors based on 20 MHz clock 9600 baud
set interrupt priority
set receive and transmit interrupt priority to 7 (lowest)
A
A
enable interrupts
enable uart3
enable receive and transmit operation
UART3_User _Init()
return

Figure 77. DMA_3.0.1_serial_interface

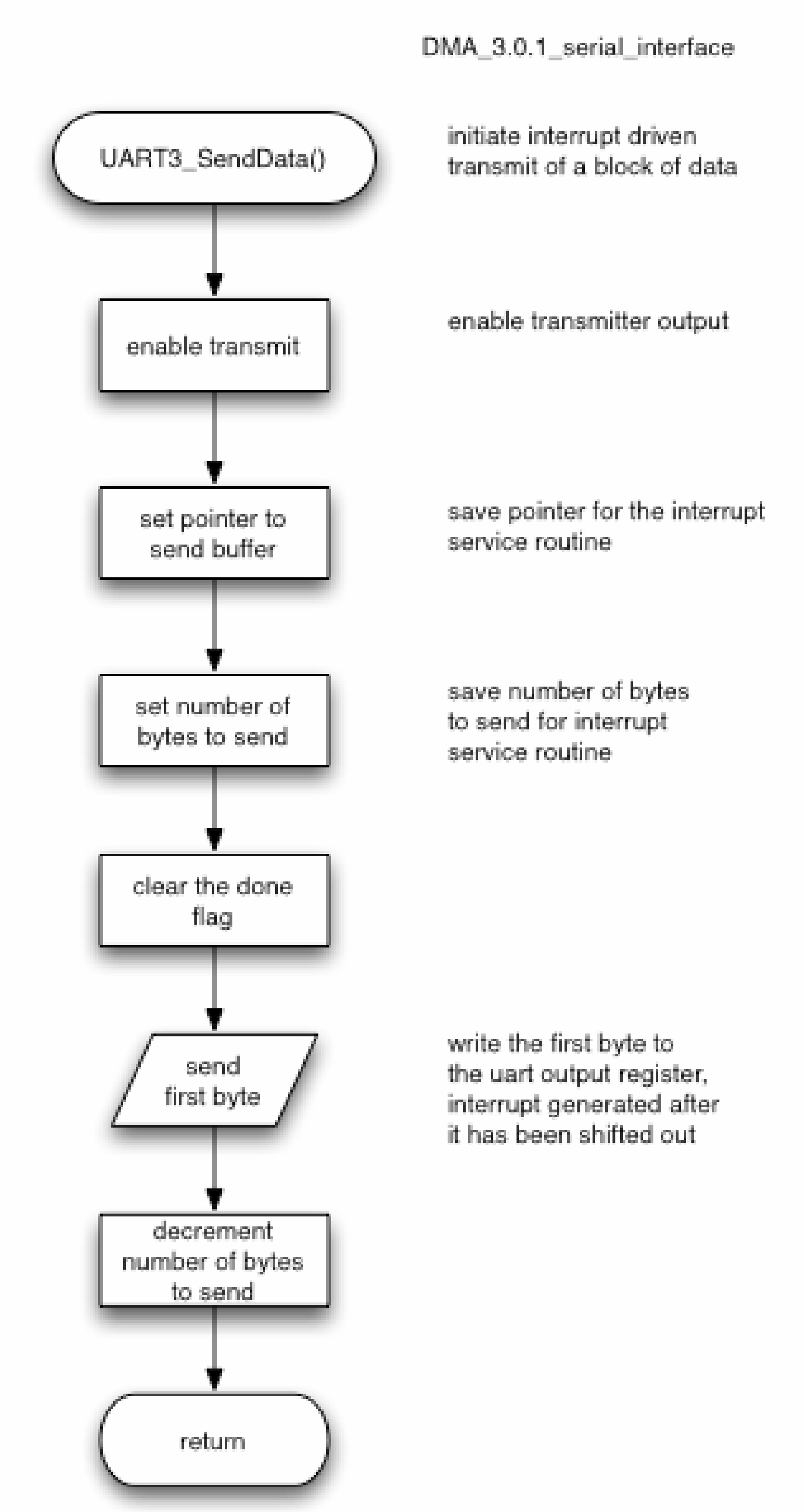

DMA_3.0.1_serial_interface
UART3_SendData()
initiate interrupt driven transmit of a block of data
enable transmit
enable transmitter output
set pointer to send buffer
save pointer for the interrupt service routine
set number of bytes to send
save number of bytes to send for interrupt service routine
clear the done flag
send first byte
write the first byte to the uart output register, interrupt generated after it has been shifted out
decrement number of bytes to send
return

Figure 78. DMA_3.0.2_serial_interface

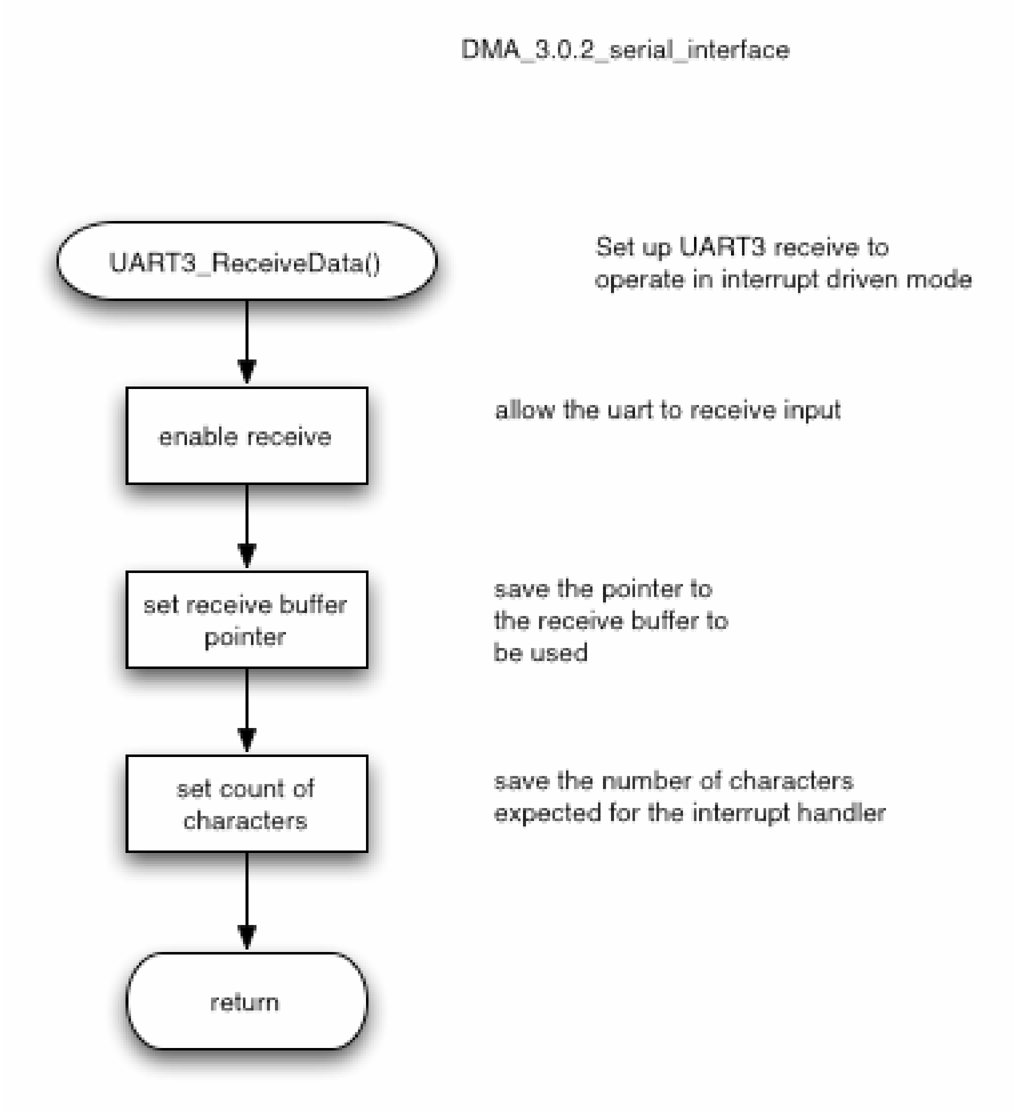

DMA_3.0.2_serial_interface
UART3_ReceiveData()
Set up UART3 receive to operate in interrupt driven mode
enable receive
allow the uart to receive input
set receive buffer pointer
save the pointer to the receive buffer to be used
set count of characters
save the number of characters expected for the interrupt handler
return

Figure 79. DMA_3.0.3_serial_interface

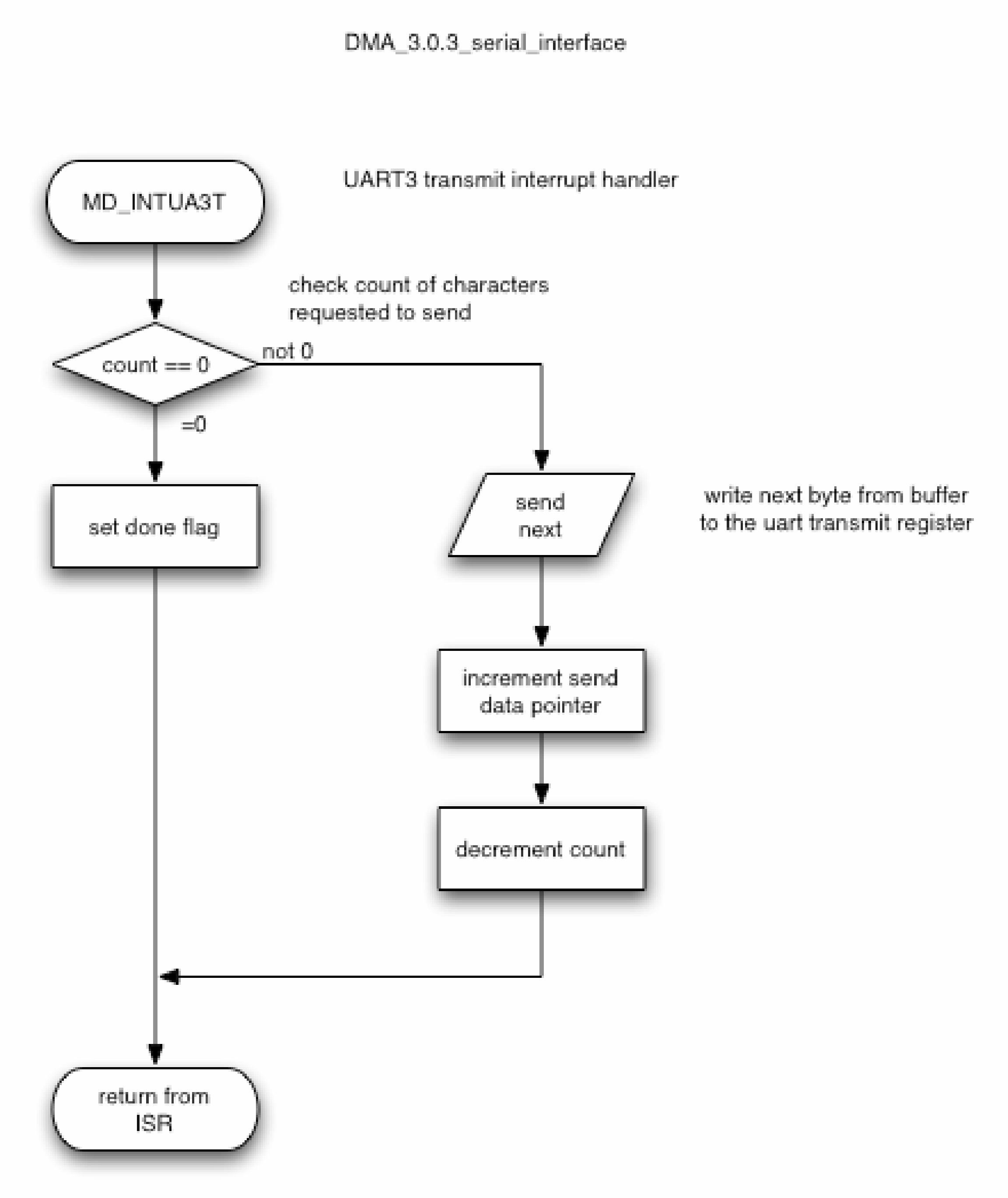

DMA_3.0.3_serial_interface
MD_INTUA3T
UART3 transmit interrupt handler
check count of characters
requested to send
count == 0
not 0
=0
set done flag
send
next
write next byte from buffer
to the uart transmit register
increment send
data pointer
decrement count
return from
ISR

Figure 80. DMA_3.0.4_serial_interface

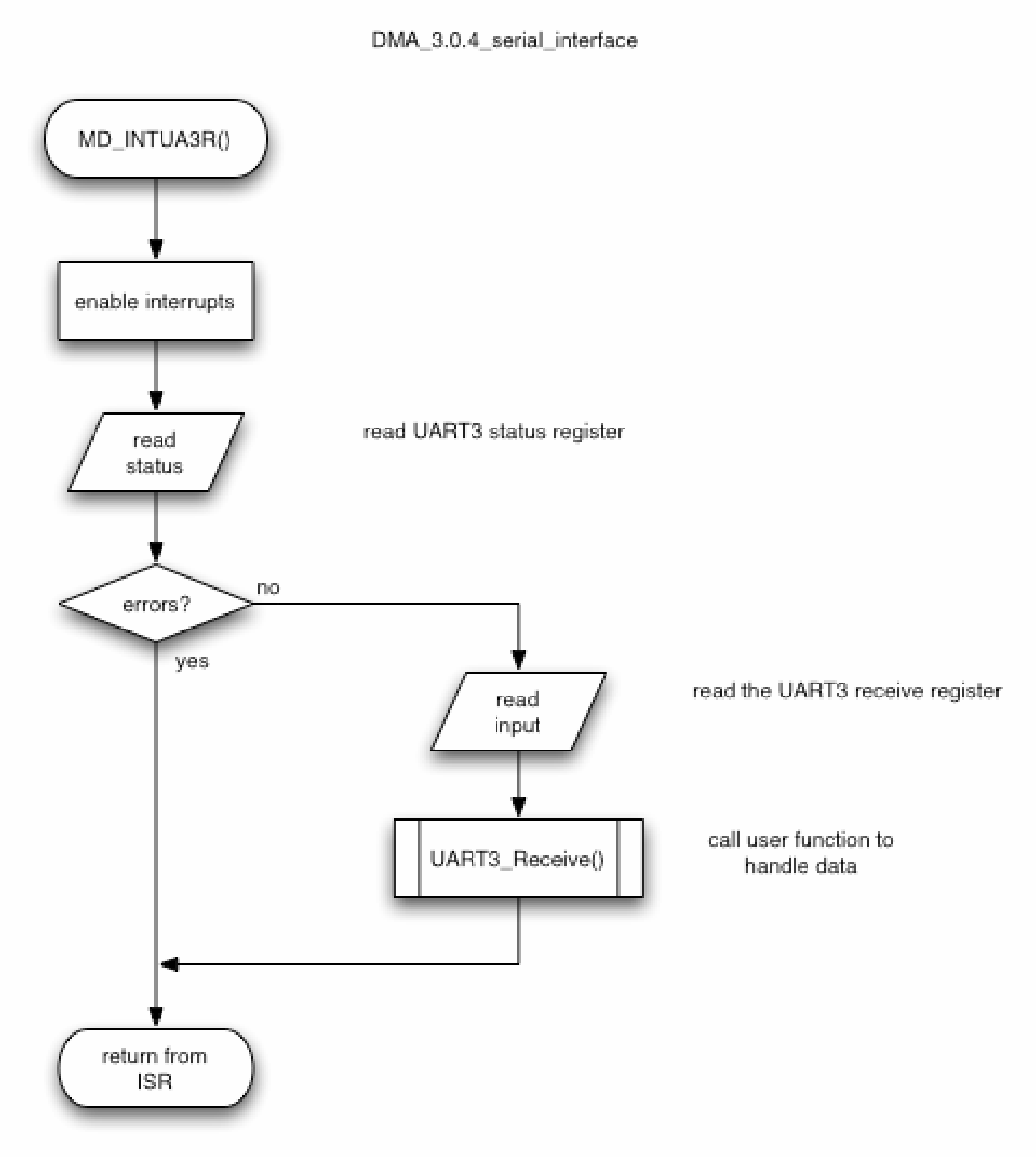

DMA_3.0.4_serial_interface
MD_INTUA3R()
enable interrupts
read status
read UART3 status register
errors?
no
yes
read input
read the UART3 receive register
UART3_Receive()
call user function to handle data
return from ISR

Figure 81. DMA_3.0.5_serial_interface

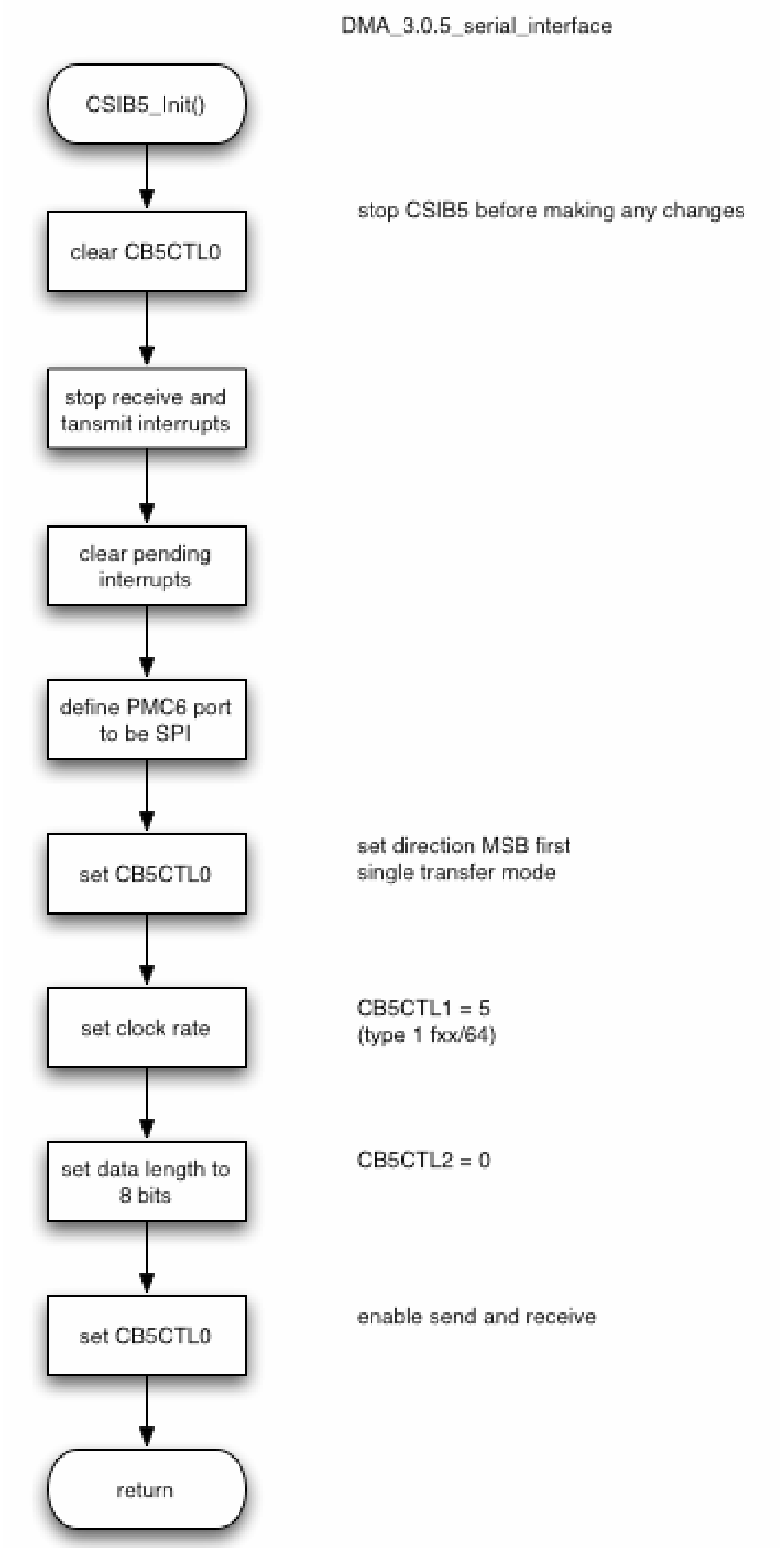


Figure 82. DMA_3.0.6_serial_interface

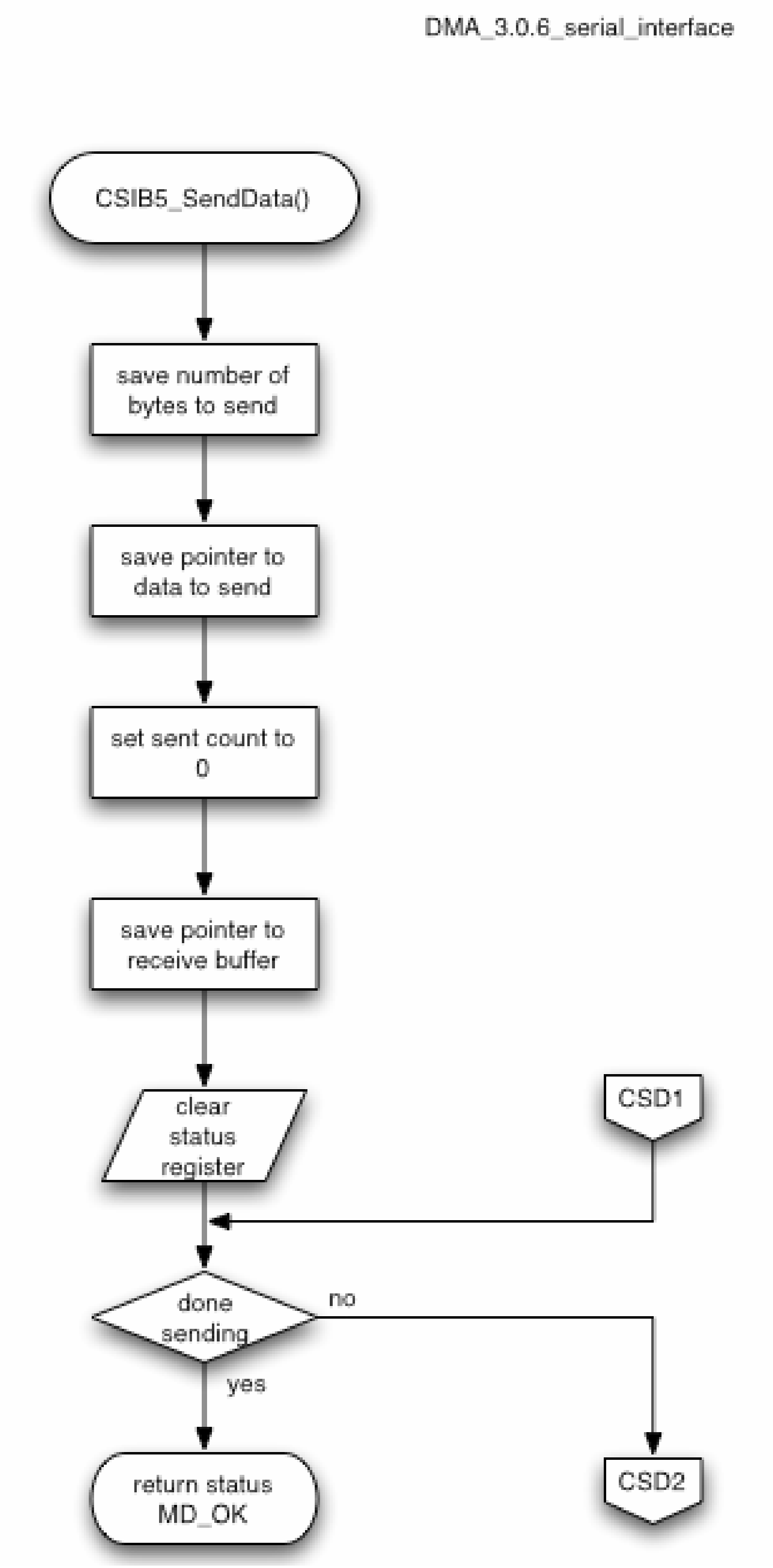

DMA_3.0.6_serial_interface
CSIB5_SendData()
save number of bytes to send
save pointer to data to send
set sent count to 0
save pointer to receive buffer
clear status register
CSD1
done sending
no
yes
return status MD_OK
CSD2

Figure 83. DMA_3.0.7_serial_interface

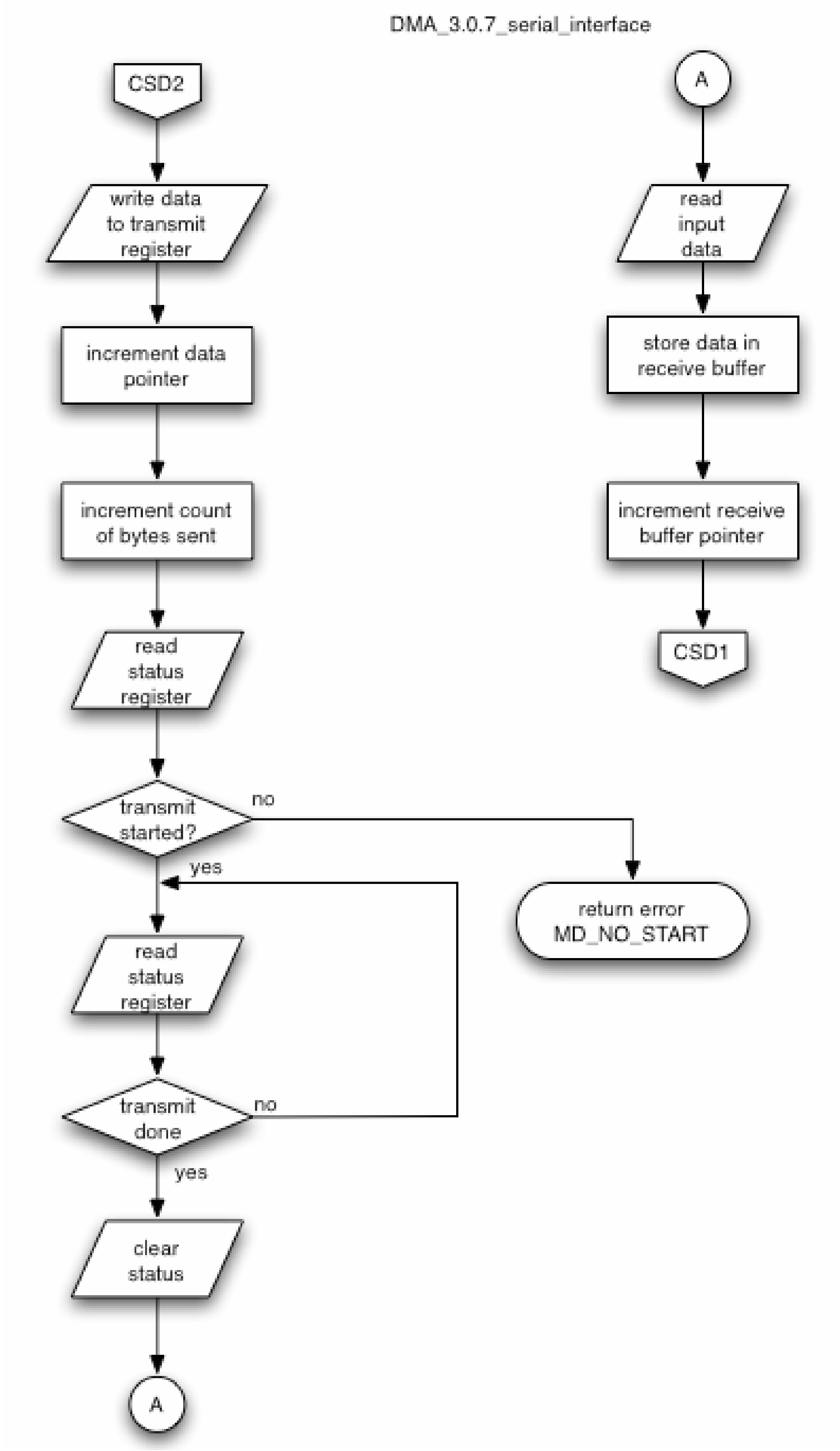


Figure 84. DMA_3.0.8_serial_interface

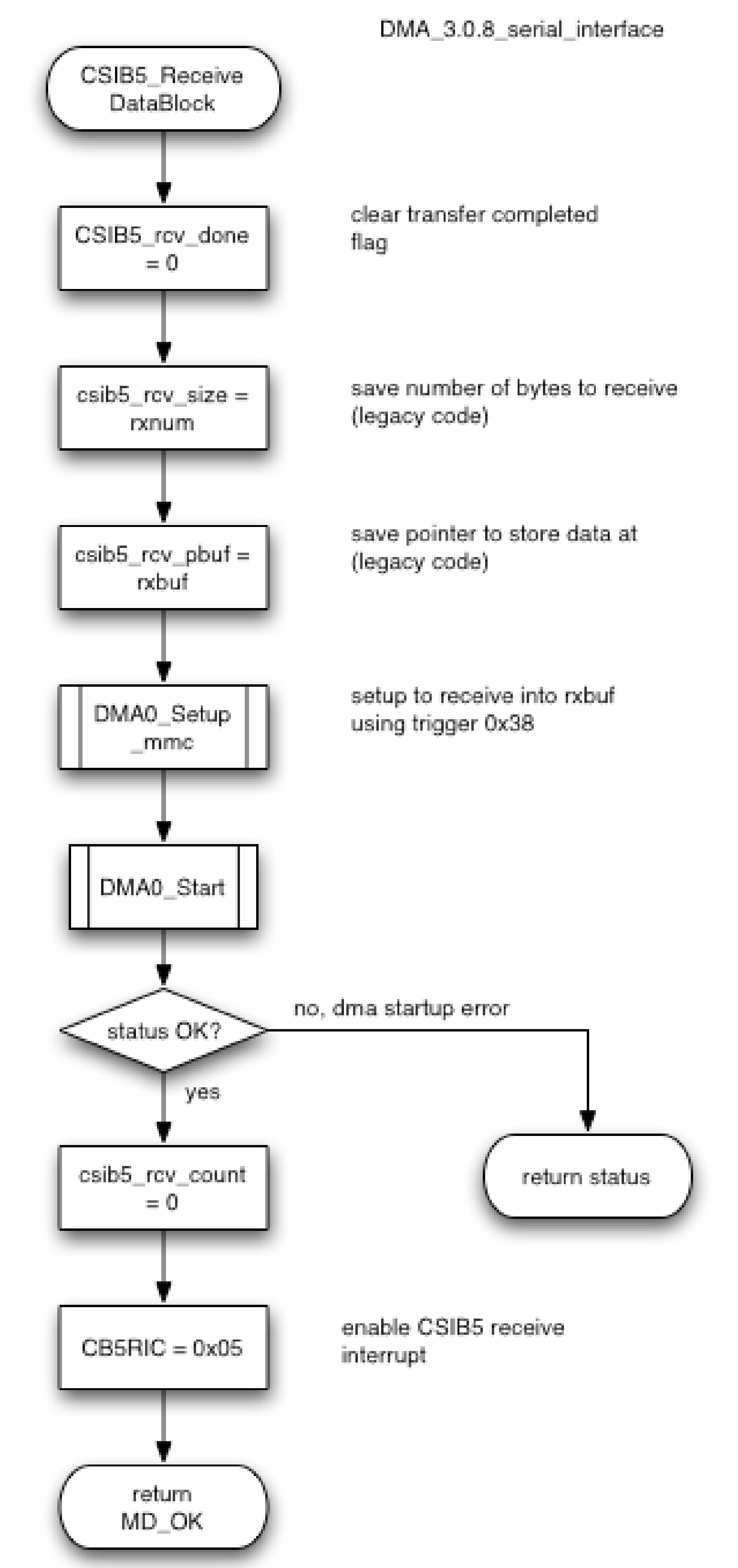

DMA_3.0.8_serial_interface
CSIB5_Receive DataBlock
CSIB5_rcv_done = 0
clear transfer completed flag
csib5_rcv_size = rxnum
save number of bytes to receive (legacy code)
csib5_rcv_pbuf = rxbuf
save pointer to store data at (legacy code)
DMA0_Setup _mmc
setup to receive into rxbuf using trigger 0x38
DMA0_Start
status OK?
no, dma startup error
yes
csib5_rcv_count = 0
return status
CB5RIC = 0x05
enable CSIB5 receive interrupt
return MD_OK

Figure 85. DMA_3.0.9_serial_interface

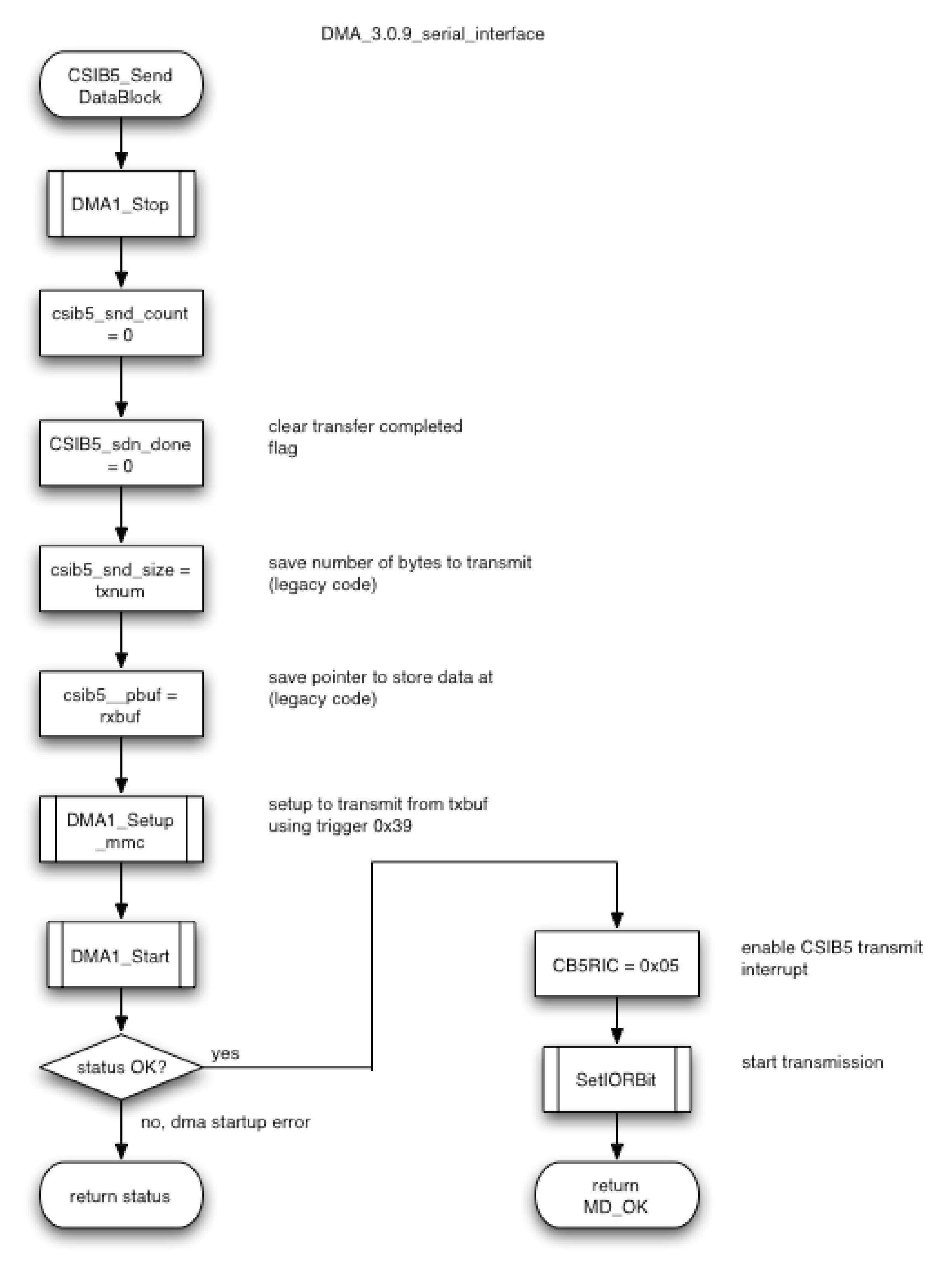


Figure 86. DMA_3.0.10_serial_interface

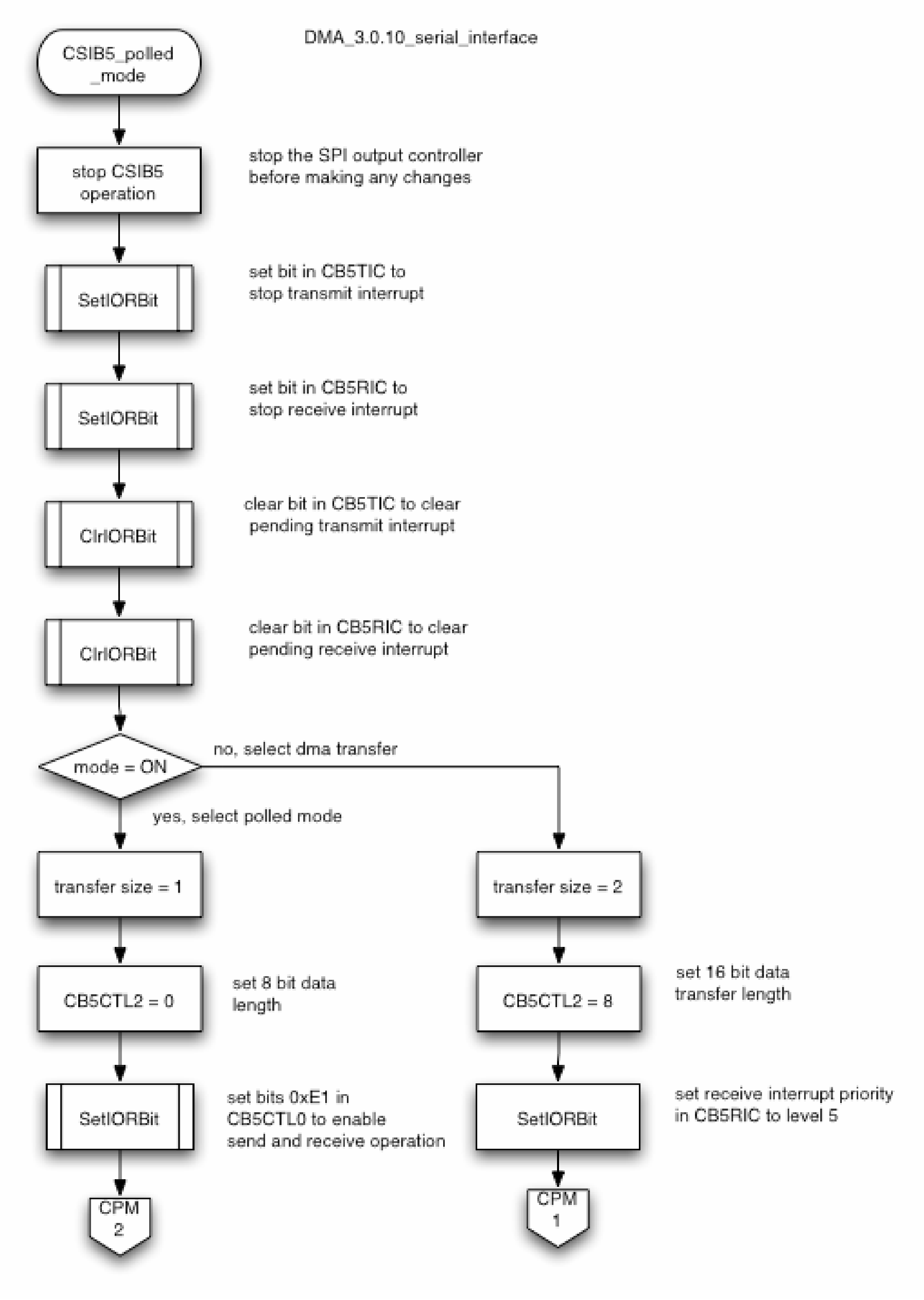

DMA_3.0.10_serial_interface
CSIB5_polled _mode
stop CSIB5 operation
stop the SPI output controller before making any changes
SetIORBit
set bit in CB5TIC to stop transmit interrupt
SetIORBit
set bit in CB5RIC to stop receive interrupt
ClrIORBit
clear bit in CB5TIC to clear pending transmit interrupt
ClrIORBit
clear bit in CB5RIC to clear pending receive interrupt
mode = ON
no, select dma transfer
yes, select polled mode
transfer size = 1
transfer size = 2
CB5CTL2 = 0
set 8 bit data length
CB5CTL2 = 8
set 16 bit data transfer length
SetIORBit
set bits 0xE1 in CB5CTL0 to enable send and receive operation
SetIORBit
set receive interrupt priority in CB5RIC to level 5
CPM 2
CPM 1

Figure 87. DMA_3.0.11_serial_interface

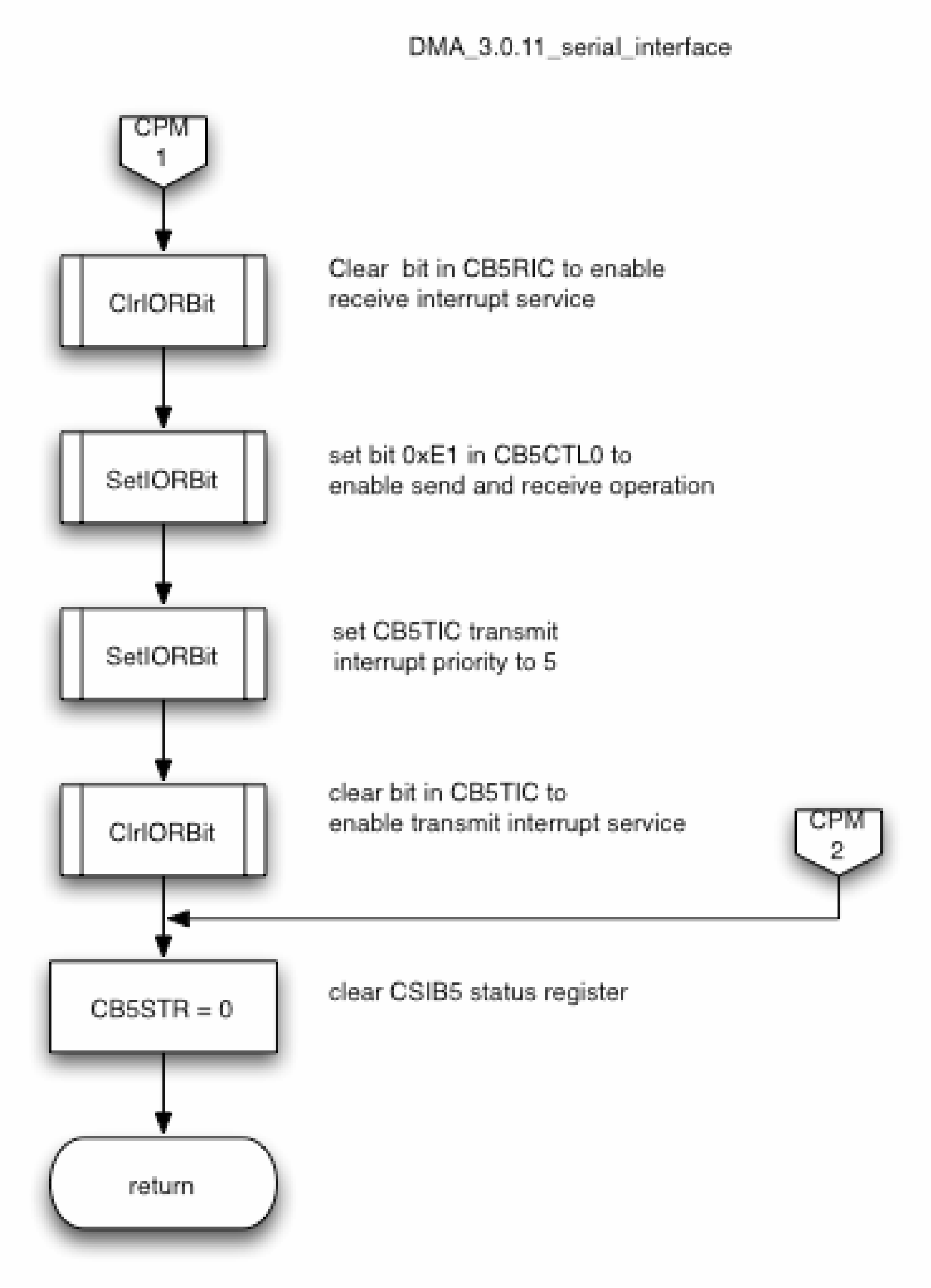

DMA_3.0.11_serial_interface
CPM 1
ClrIORBit
Clear bit in CB5RIC to enable receive interrupt service
SetIORBit
set bit 0xE1 in CB5CTL0 to enable send and receive operation
SetIORBit
set CB5TIC transmit interrupt priority to 5
ClrIORBit
clear bit in CB5TIC to enable transmit interrupt service
CPM 2
CB5STR = 0
clear CSIB5 status register
return

Figure 88. DMA_3.0.12_serial_interface

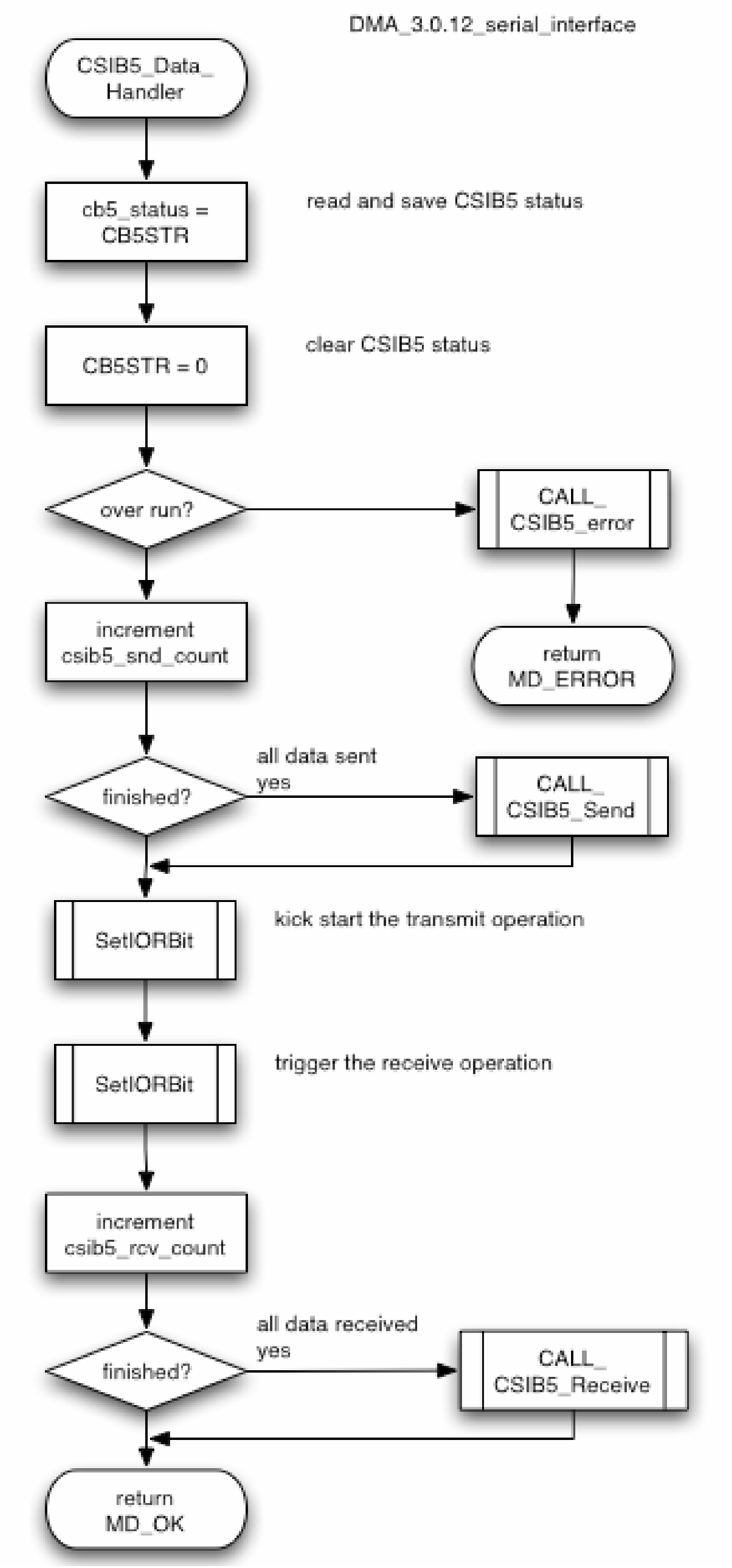
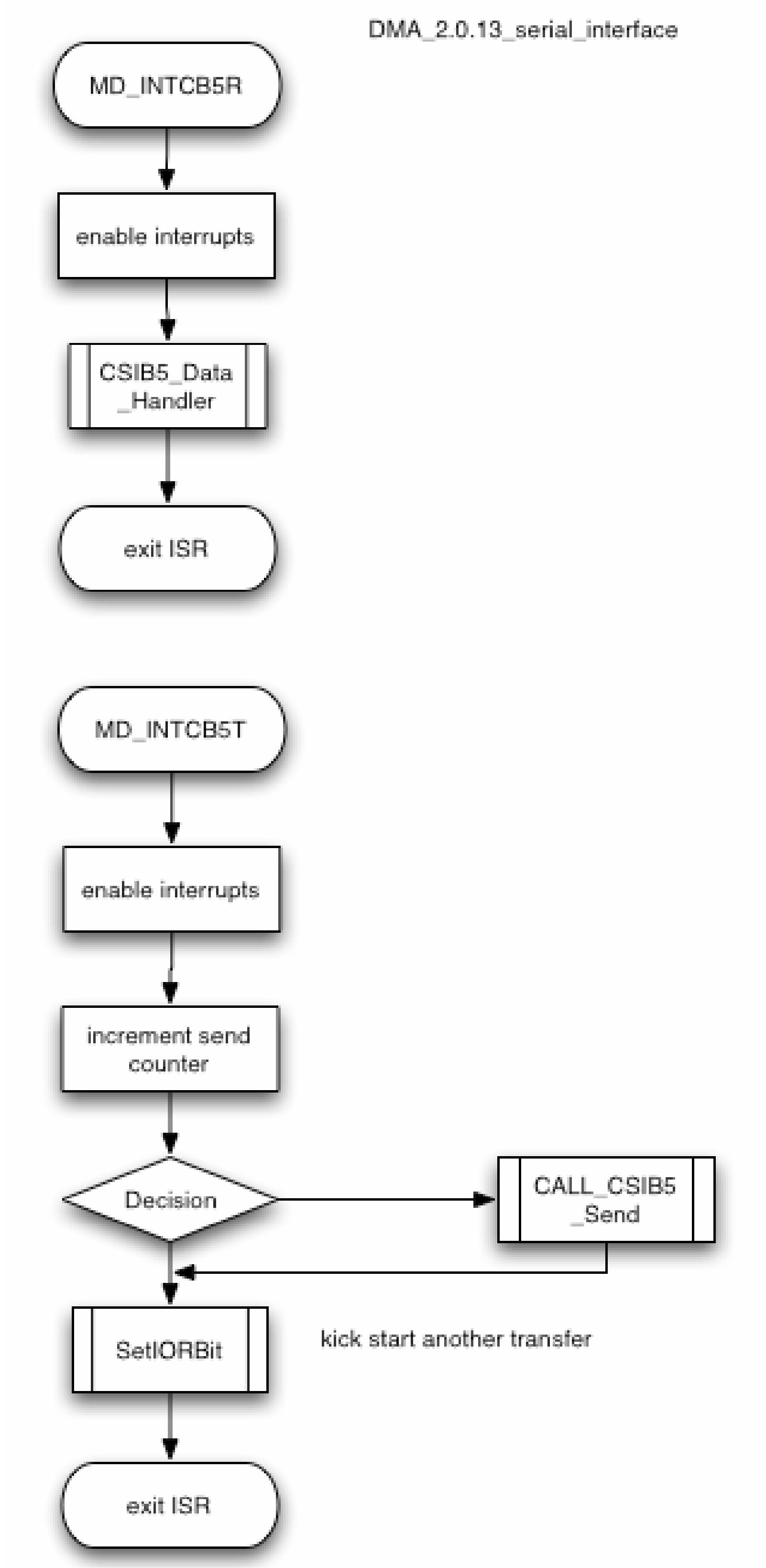


Figure 89. DMA_2.0.13_serial_interface



8.5 Serial Interface_User

Figure 90. DMA_3.1.0_serial_interface_user

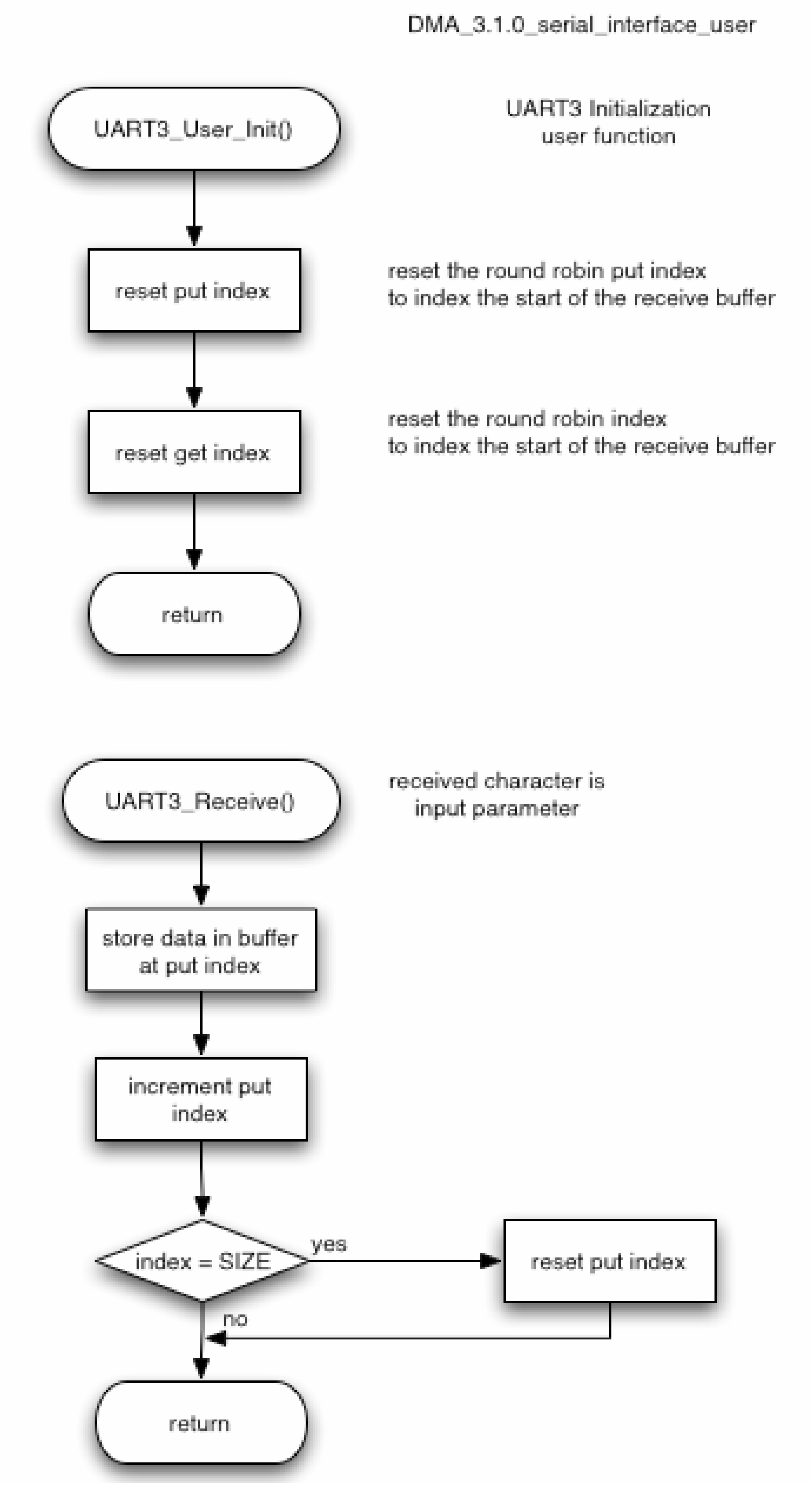


Figure 91. DMA_3.1.1_serial_interface_user

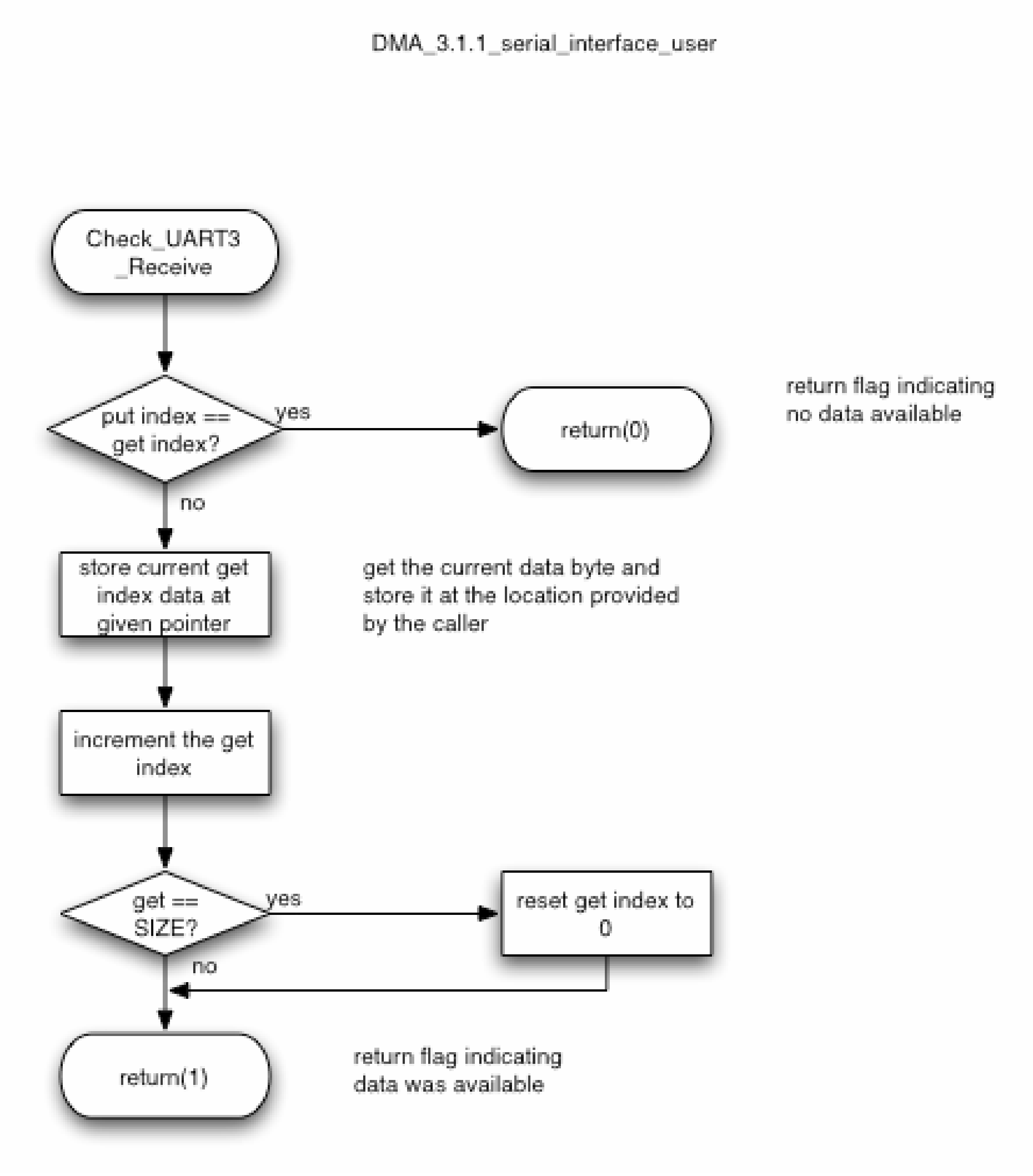

DMA_3.1.1_serial_interface_user
Check_UART3_Receive
put index == get index?
yes
return(0)
return flag indicating no data available
no
store current get index data at given pointer
get the current data byte and store it at the location provided by the caller
increment the get index
get == SIZE?
yes
reset get index to 0
no
return(1)
return flag indicating data was available

Figure 92. DMA_3.1.2_serial_interface_user

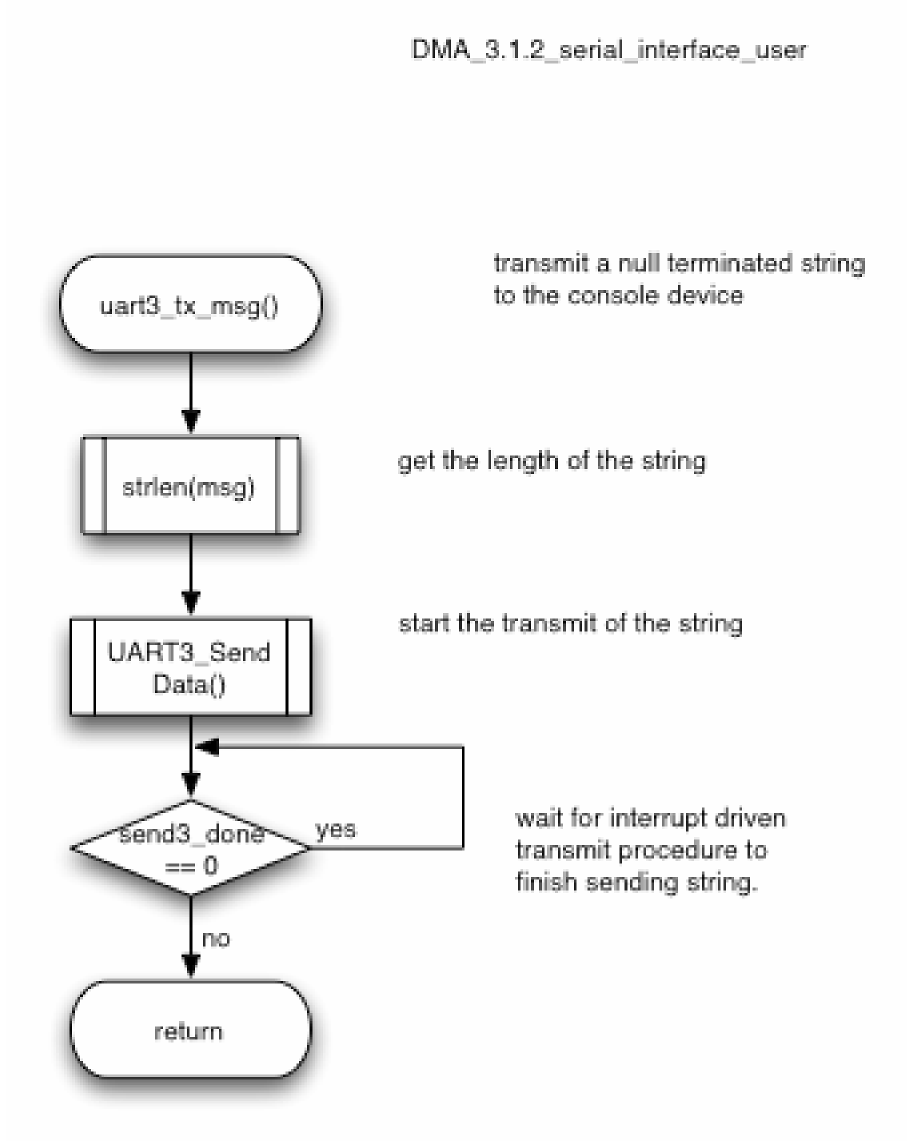

DMA_3.1.2_serial_interface_user
uart3_tx_msg()
transmit a null terminated string to the console device
strlen(msg)
get the length of the string
UART3_Send Data()
start the transmit of the string
send3_done == 0
yes
wait for interrupt driven transmit procedure to finish sending string.
no
return

Figure 93.   DMA_3.1.3_serial_interface_user

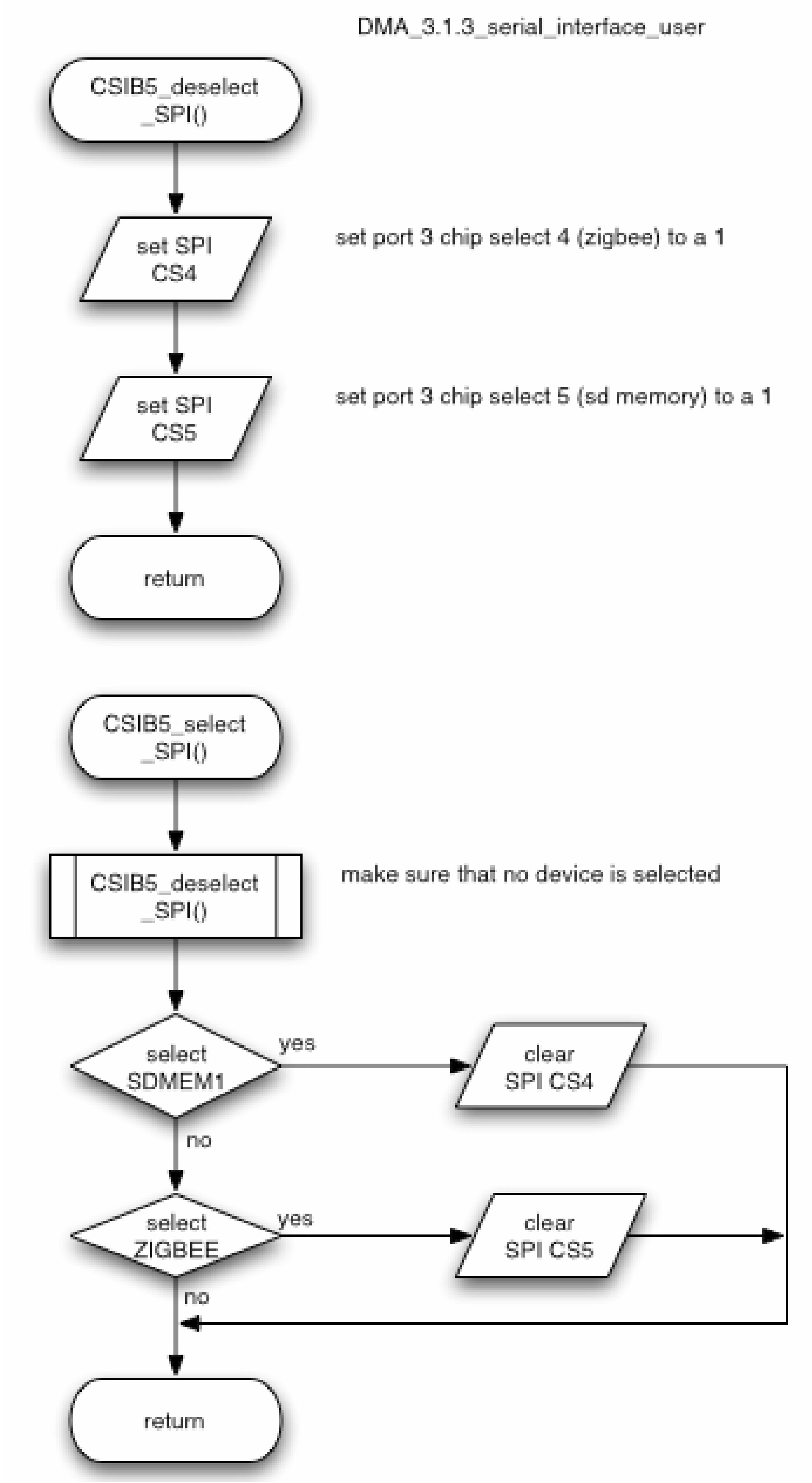
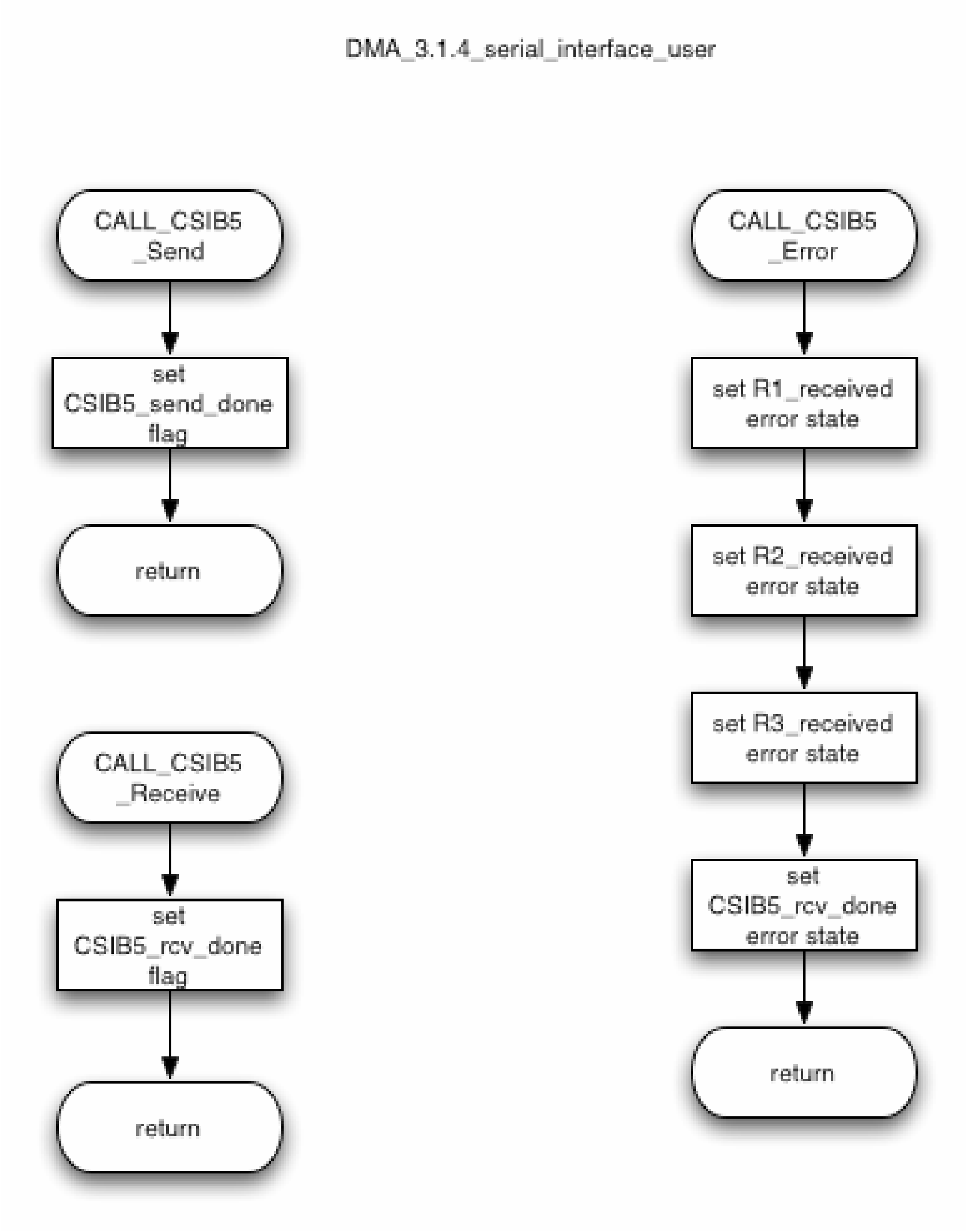
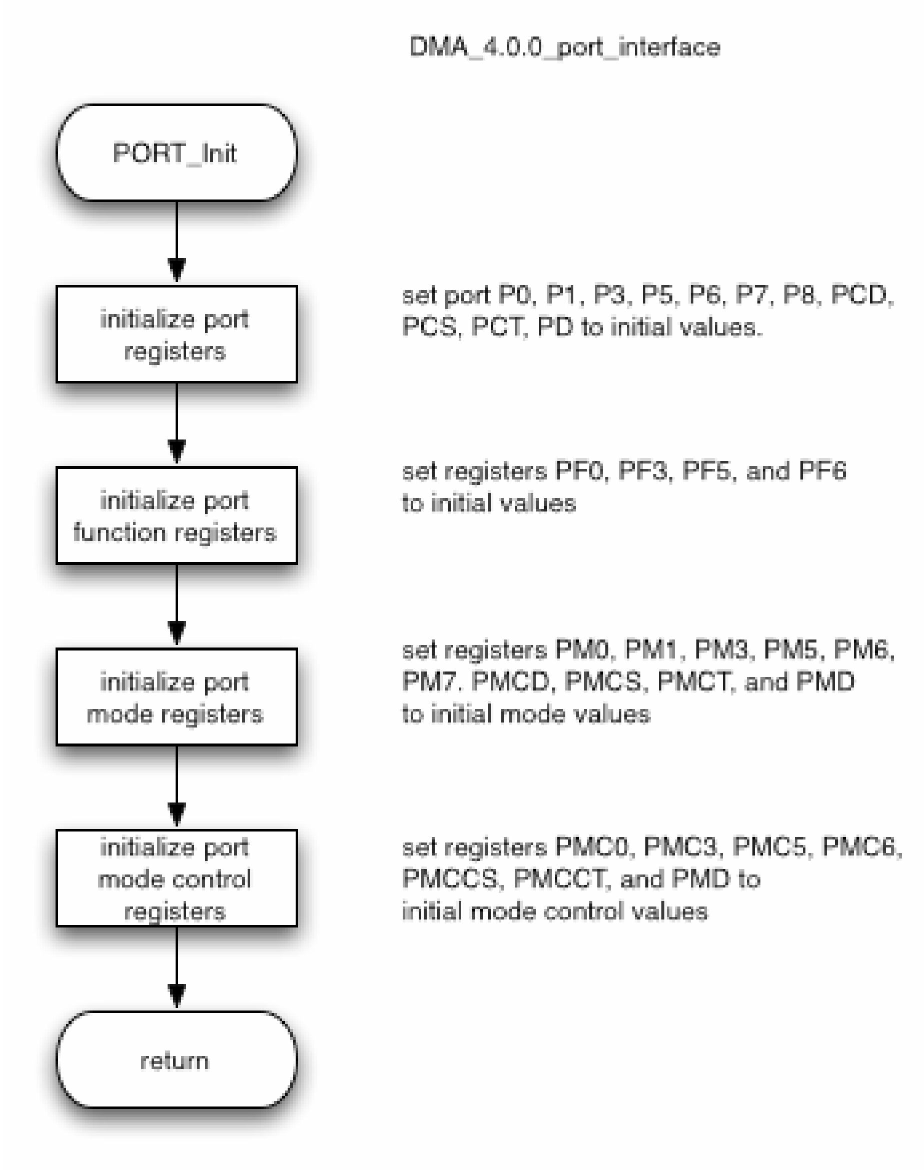


Figure 94. DMA_3.1.4_serial_interface_user


DMA_3.1.4_serial_interface_user
CALL_CSIB5 _Send
set CSIB5_send_done flag
return
CALL_CSIB5 _Receive
set CSIB5_rcv_done flag
return
CALL_CSIB5 _Error
set R1_received error state
set R2_received error state
set R3_received error state
set CSIB5_rcv_done error state
return

8.6 Port Interface

Figure 95. DMA_4.0.0_port_interface



DMA_4.0.0_port_interface

PORT_Init
initialize port registers
initialize port function registers
initialize port mode registers
initialize port mode control registers
return

set port P0, P1, P3, P5, P6, P7, P8, PCD, PCS, PCT, PD to initial values.

set registers PF0, PF3, PF5, and PF6 to initial values

set registers PM0, PM1, PM3, PM5, PM6, PM7. PMCD, PMCS, PMCT, and PMD to initial mode values

set registers PMC0, PMC3, PMC5, PMC6, PMCCS, PMCCT, and PMD to initial mode control values

8.7 Port Bus Interface

Figure 96. DMA_4.1.0_port_bus_interface

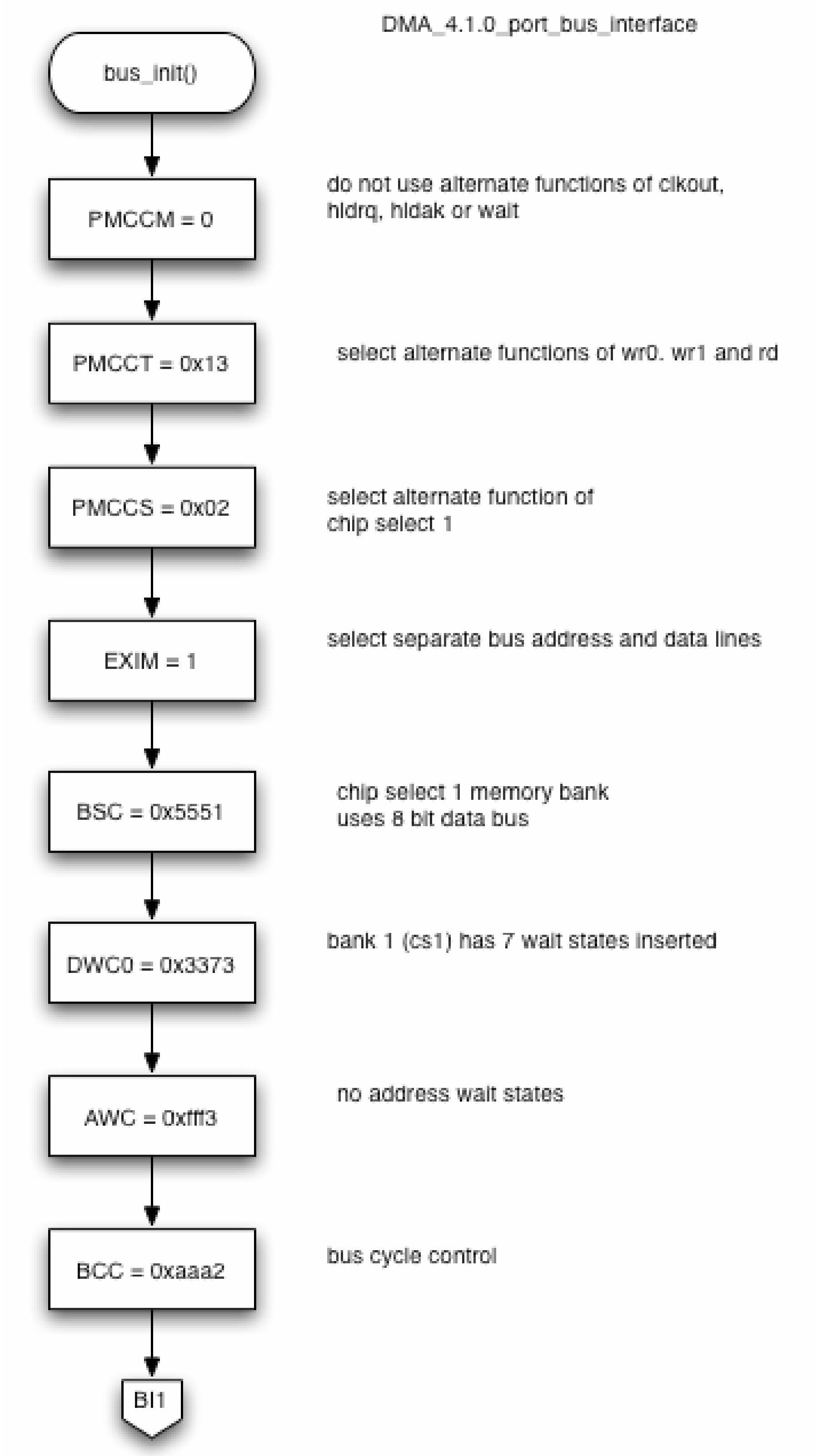
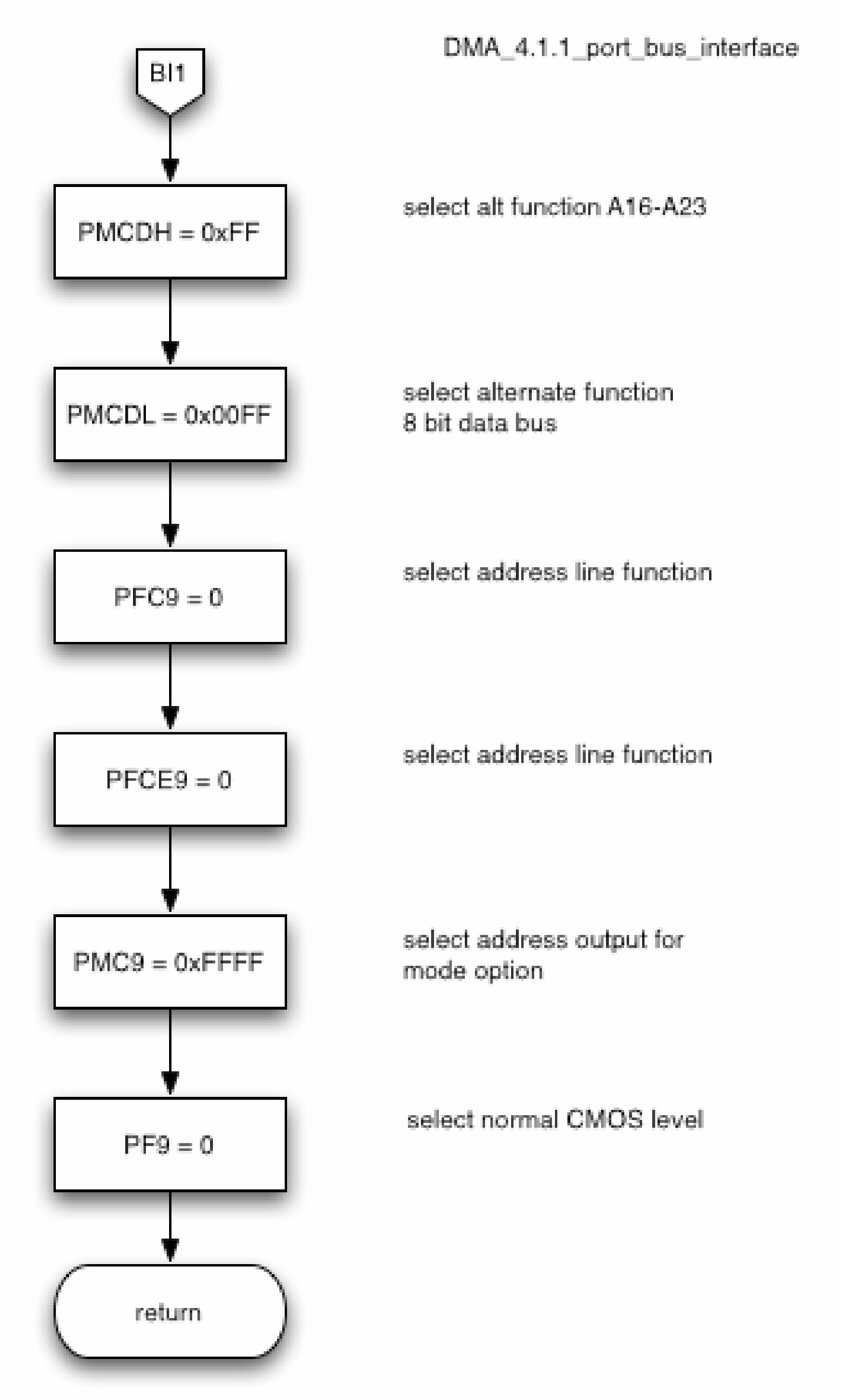

DMA_4.1.0_port_bus_interface
bus_init()
PMCCM = 0
do not use alternate functions of clkout, hldrq, hldak or wait
PMCCT = 0x13
select alternate functions of wr0. wr1 and rd
PMCCS = 0x02
select alternate function of chip select 1
EXIM = 1
select separate bus address and data lines
BSC = 0x5551
chip select 1 memory bank uses 8 bit data bus
DWC0 = 0x3373
bank 1 (cs1) has 7 wait states inserted
AWC = 0xfff3
no address wait states
BCC = 0xaaa2
bus cycle control
BI1

Figure 97. DMA_4.1.1_port_bus_interface


DMA_4.1.1_port_bus_interface
BI1
PMCDH = 0xFF
select alt function A16-A23
PMCDL = 0x00FF
select alternate function
8 bit data bus
PFC9 = 0
select address line function
PFCE9 = 0
select address line function
PMC9 = 0xFFFF
select address output for
mode option
PF9 = 0
select normal CMOS level
return

8.8 Timer Interface

Figure 98. DMA_5.0.0_timer_interface

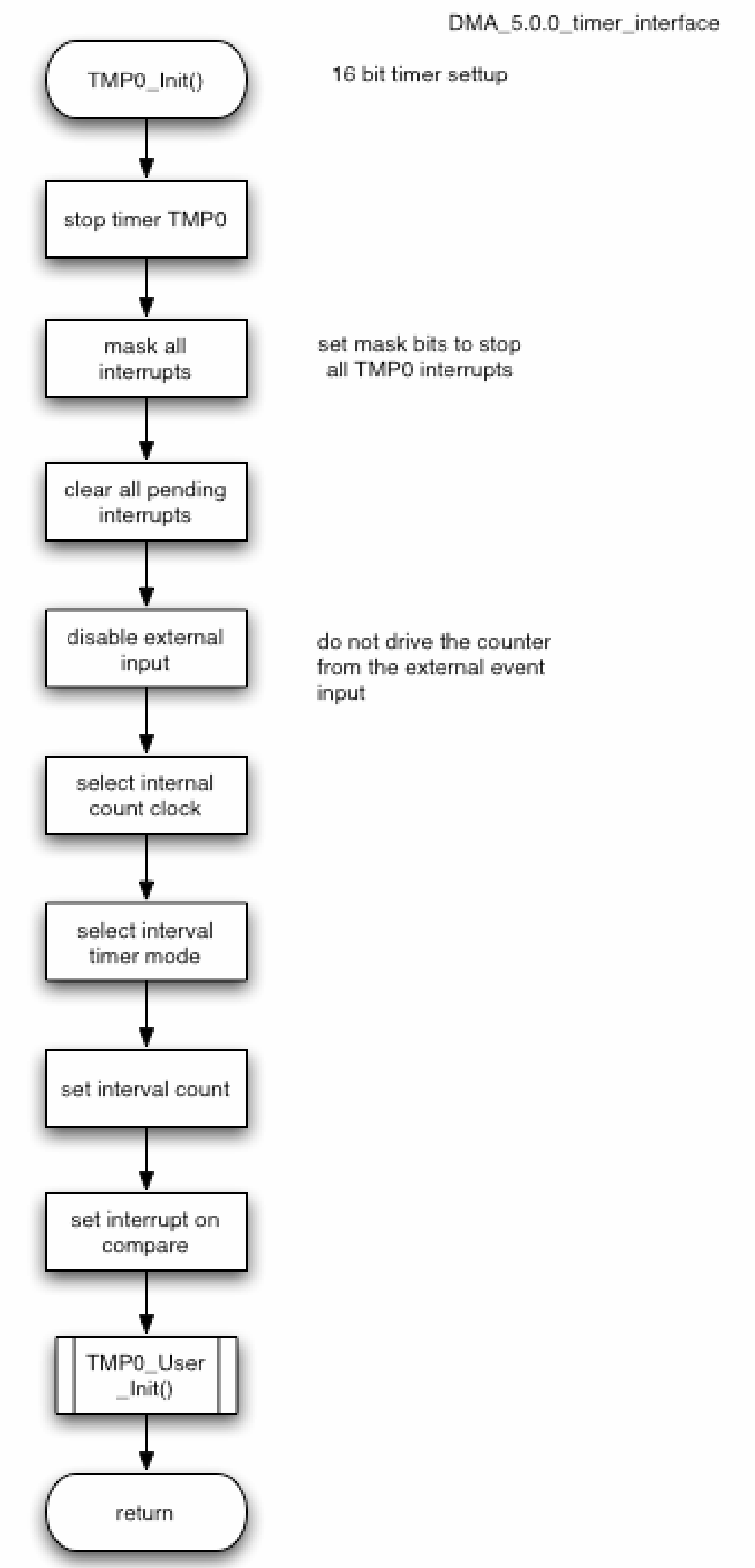

DMA_5.0.0_timer_interface
TMP0_Init()
16 bit timer settup
stop timer TMP0
mask all interrupts
set mask bits to stop all TMP0 interrupts
clear all pending interrupts
disable external input
do not drive the counter from the external event input
select internal count clock
select interval timer mode
set interval count
set interrupt on compare
TMP0_User _Init()
return

Figure 99. DMA_5.0.1_timer_interface

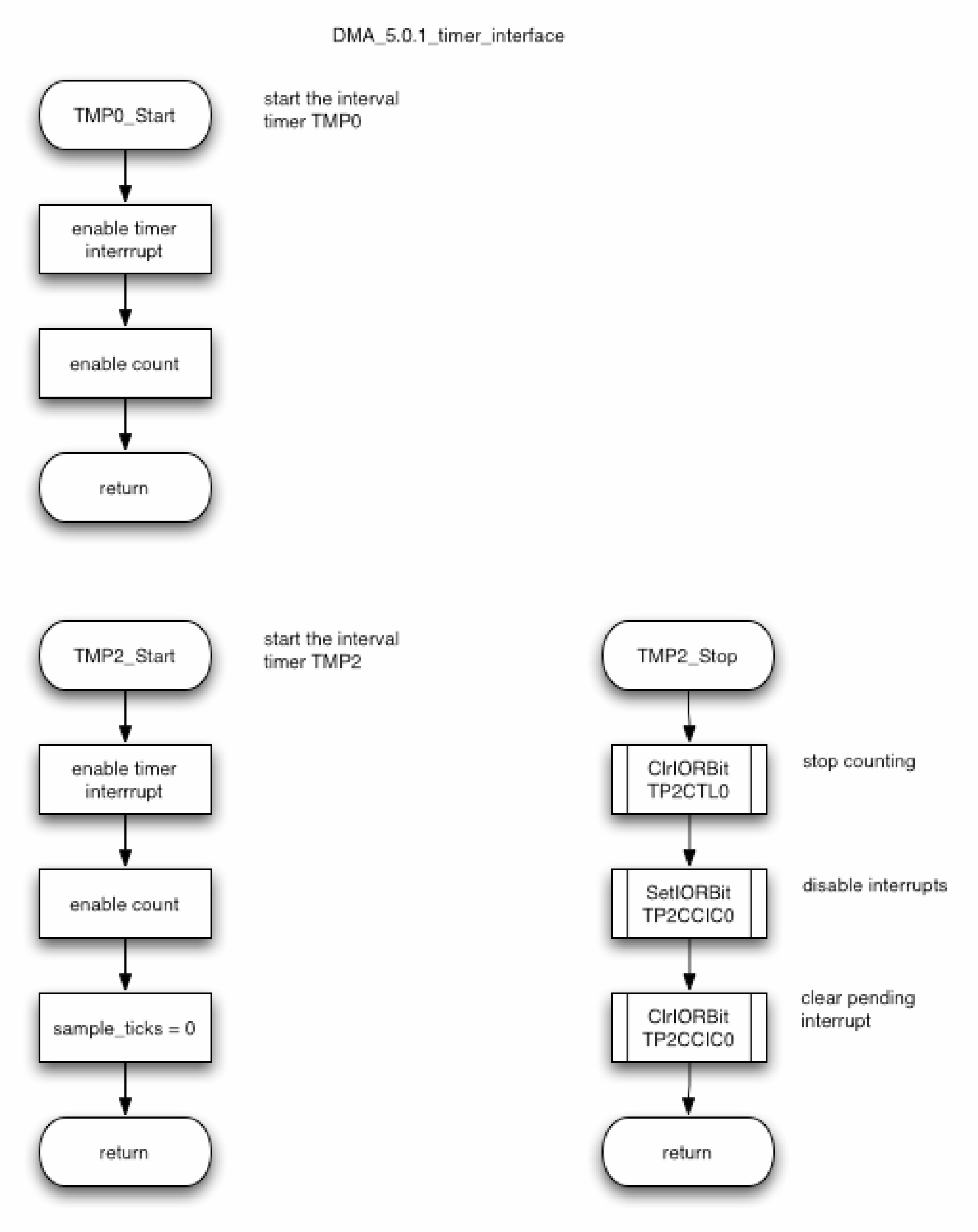
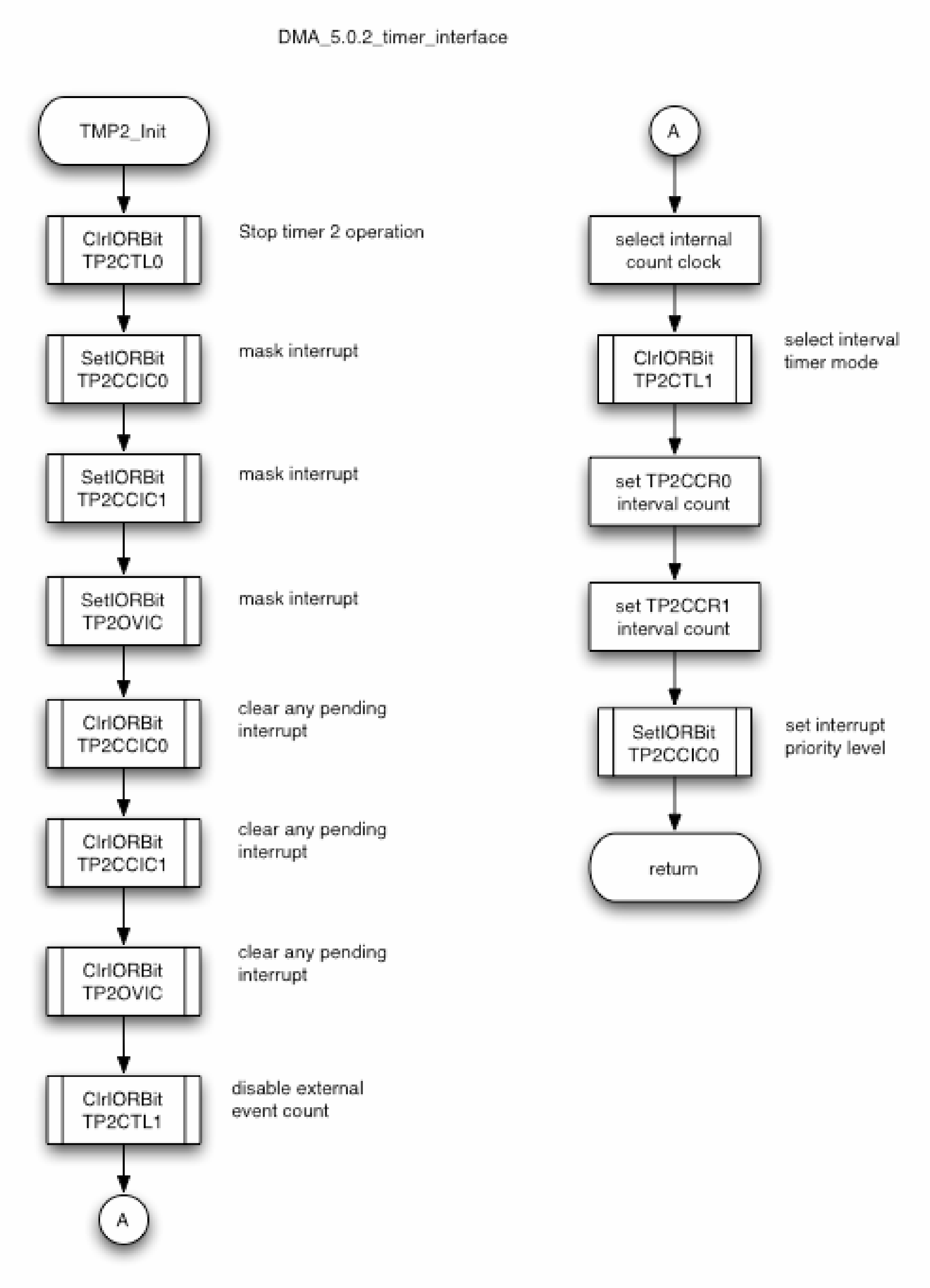


Figure 100. DMA_5.0.2_timer_interface


DMA_5.0.2_timer_interface
TMP2_Init
ClrIORBit TP2CTL0
Stop timer 2 operation
SetIORBit TP2CCIC0
mask interrupt
SetIORBit TP2CCIC1
mask interrupt
SetIORBit TP2OVIC
mask interrupt
ClrIORBit TP2CCIC0
clear any pending interrupt
ClrIORBit TP2CCIC1
clear any pending interrupt
ClrIORBit TP2OVIC
clear any pending interrupt
ClrIORBit TP2CTL1
disable external event count
A
A
select internal count clock
ClrIORBit TP2CTL1
select interval timer mode
set TP2CCR0 interval count
set TP2CCR1 interval count
SetIORBit TP2CCIC0
set interrupt priority level
return

8.9 Timer Interface_User

Figure 101. DMA_5.1.0_timer_interface_user

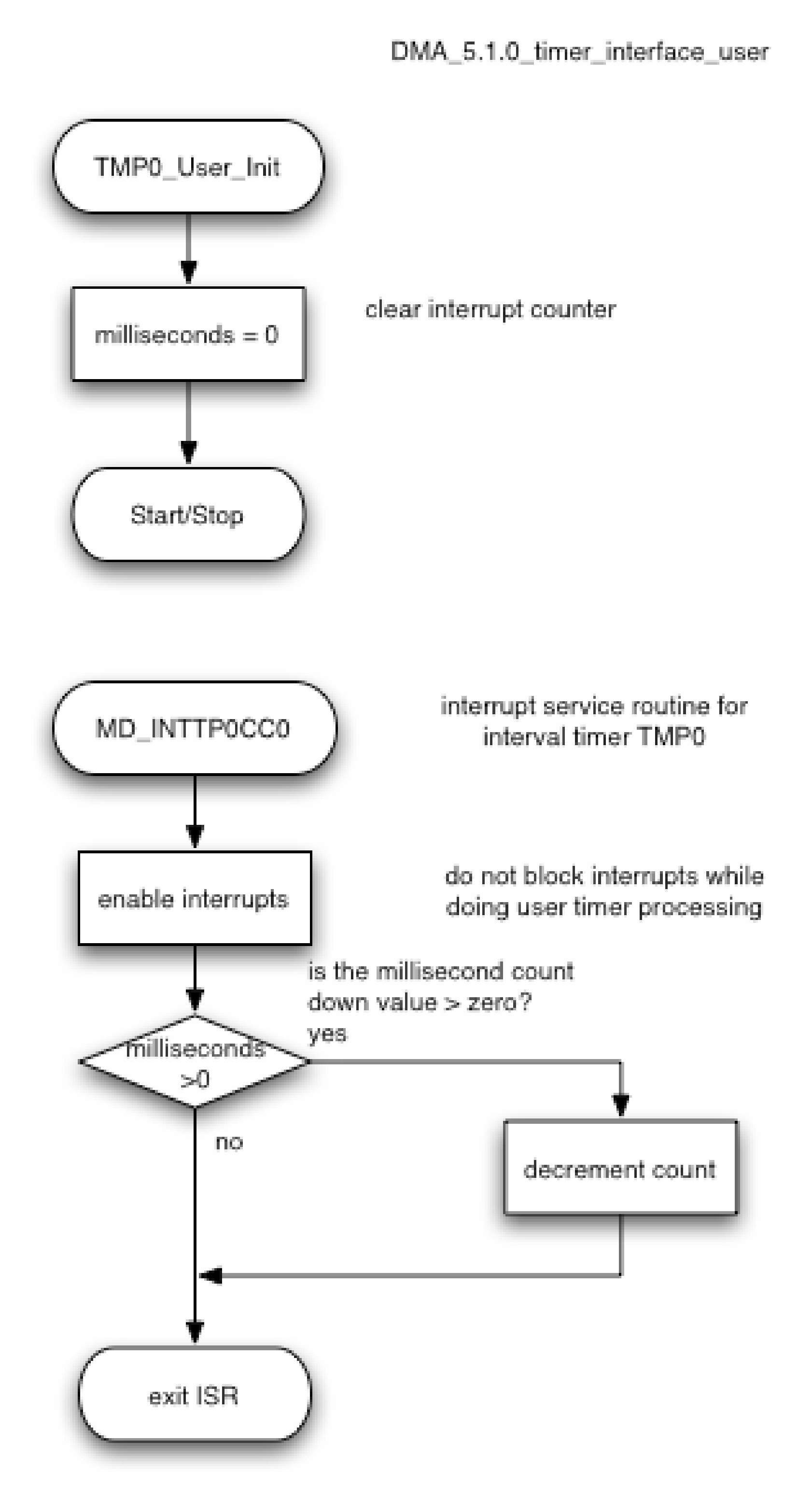

DMA_5.1.0_timer_interface_user
TMP0_User_Init
milliseconds = 0
clear interrupt counter
Start/Stop
MD_INTTP0CC0
interrupt service routine for interval timer TMP0
enable interrupts
do not block interrupts while doing user timer processing
is the millisecond count down value > zero?
yes
milliseconds >0
no
decrement count
exit ISR

Figure 102. DMA_5.1.1_timer_interface_user

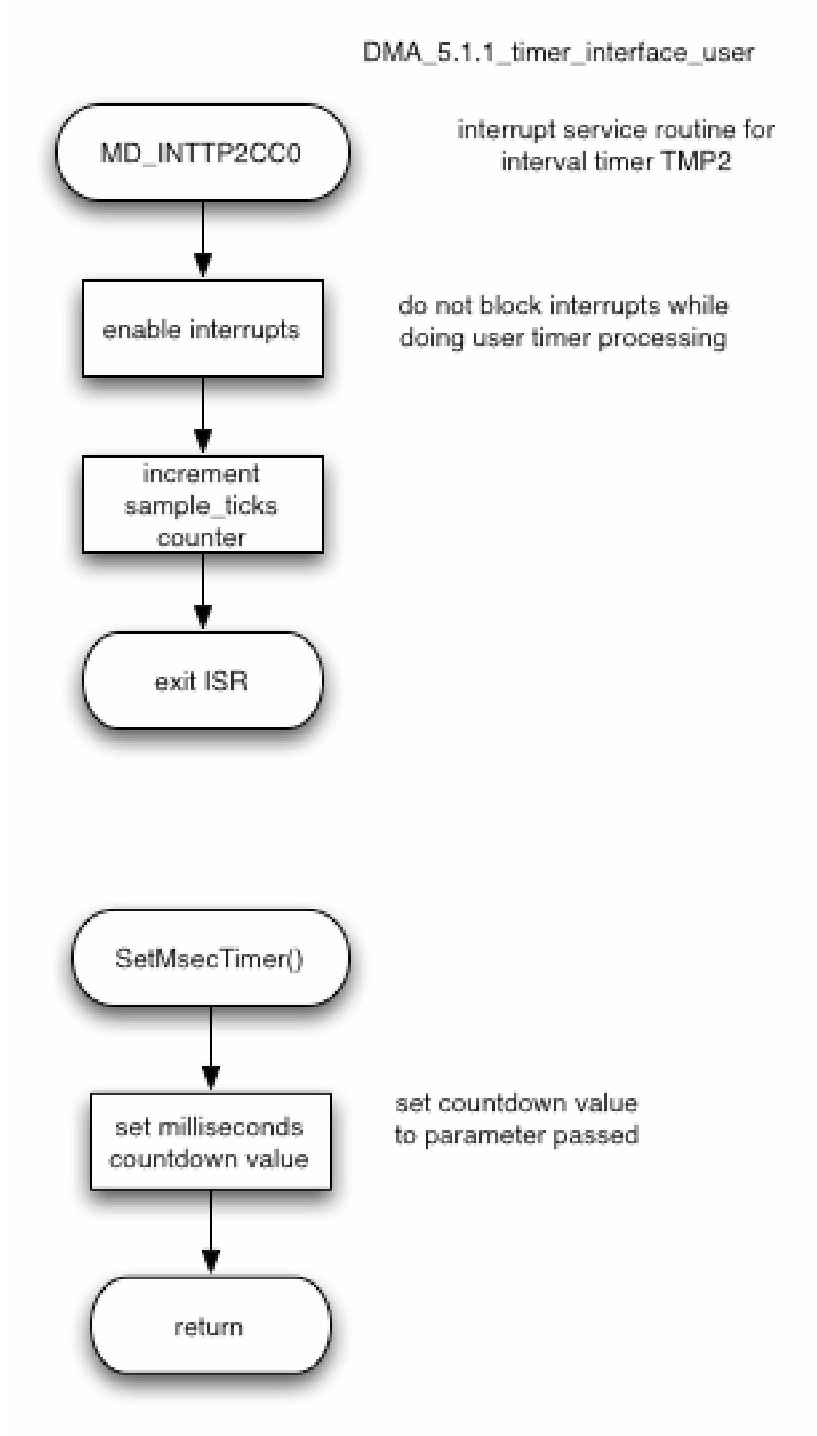
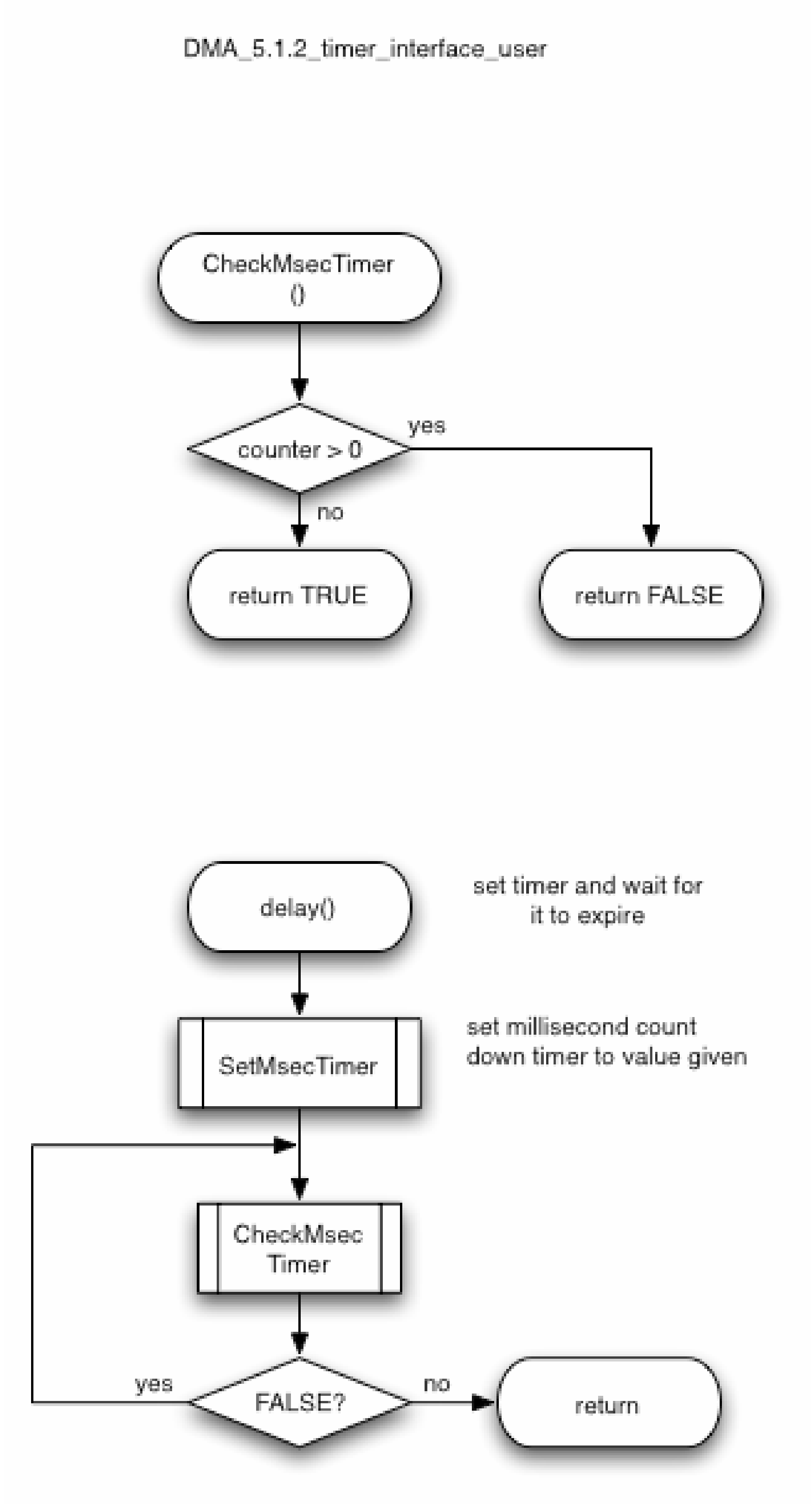

DMA_5.1.1_timer_interface_user
MD_INTTP2CC0
interrupt service routine for interval timer TMP2
enable interrupts
do not block interrupts while doing user timer processing
increment sample_ticks counter
exit ISR
SetMsecTimer()
set milliseconds countdown value
set countdown value to parameter passed
return

Figure 103. DMA_5.1.2_timer_interface_user


DMA_5.1.2_timer_interface_user
CheckMsecTimer ()
counter > 0
yes
no
return TRUE
return FALSE
delay()
set timer and wait for it to expire
SetMsecTimer
set millisecond count down timer to value given
CheckMsec Timer
yes
FALSE?
no
return

8.10 DMA Interface

Figure 104. DMA_6.0.0_dma_interface

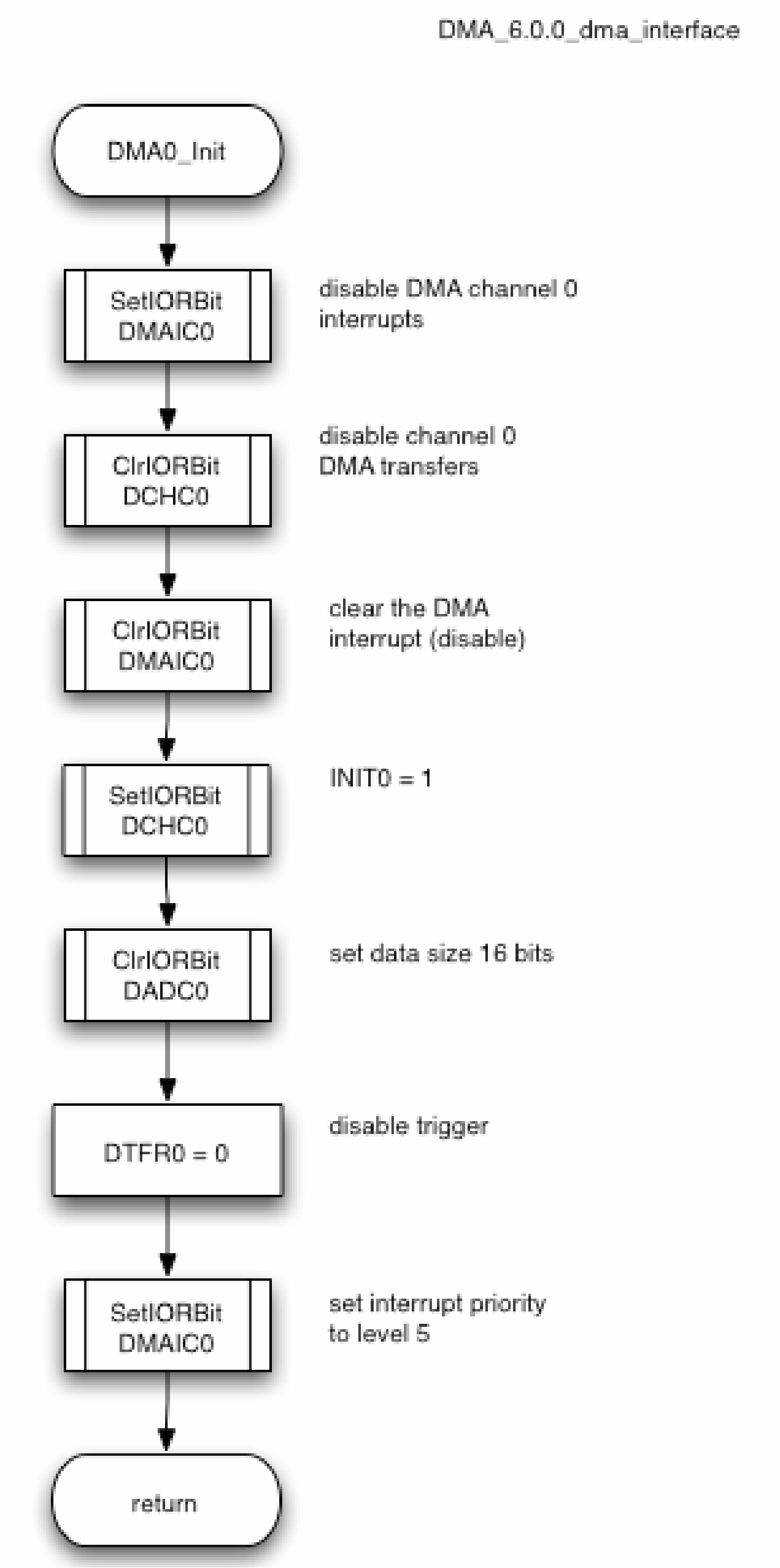


Figure 105. DMA_6.0.1_dma_interface

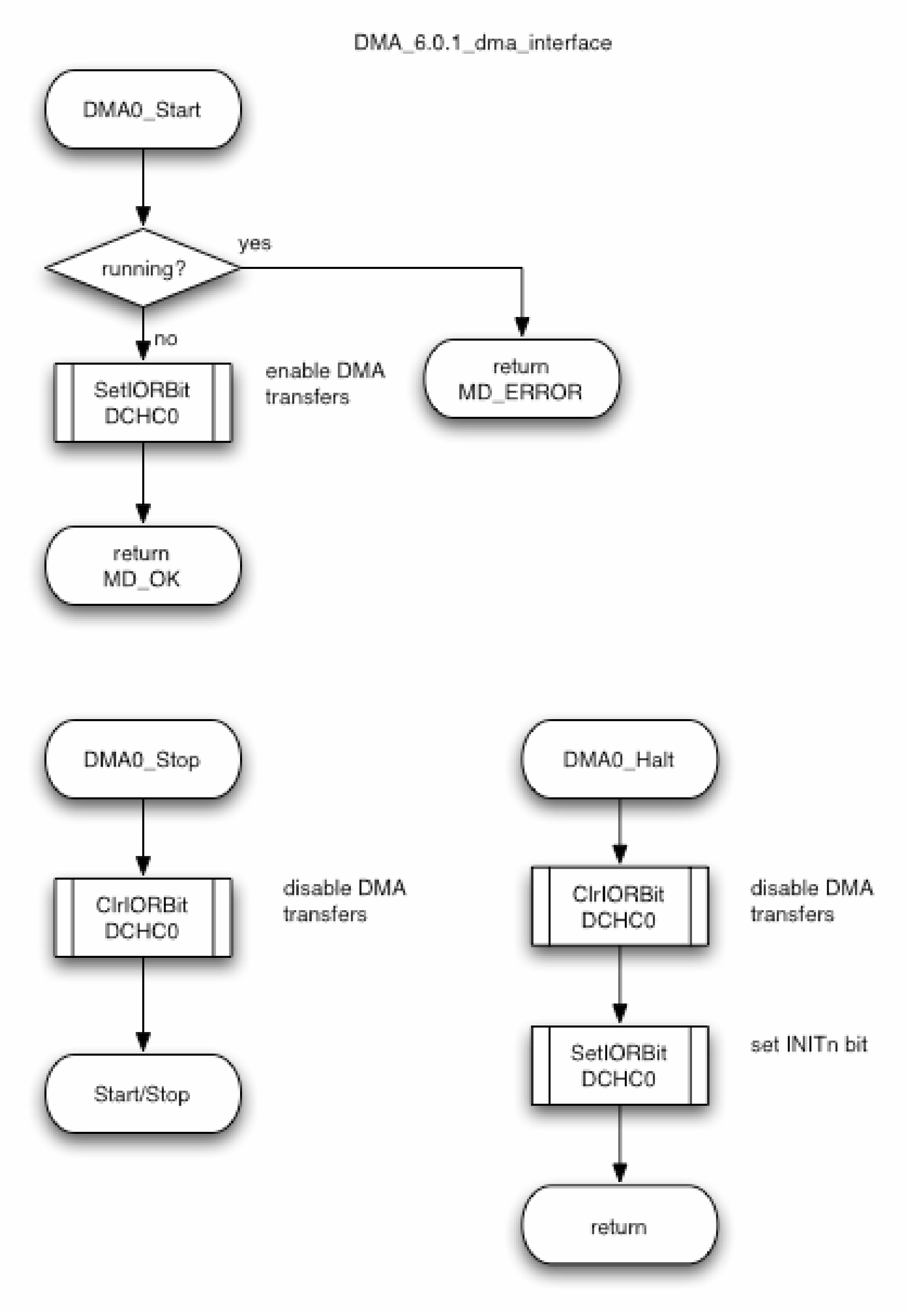


Figure 106. DMA_6.0.2_dma_interface

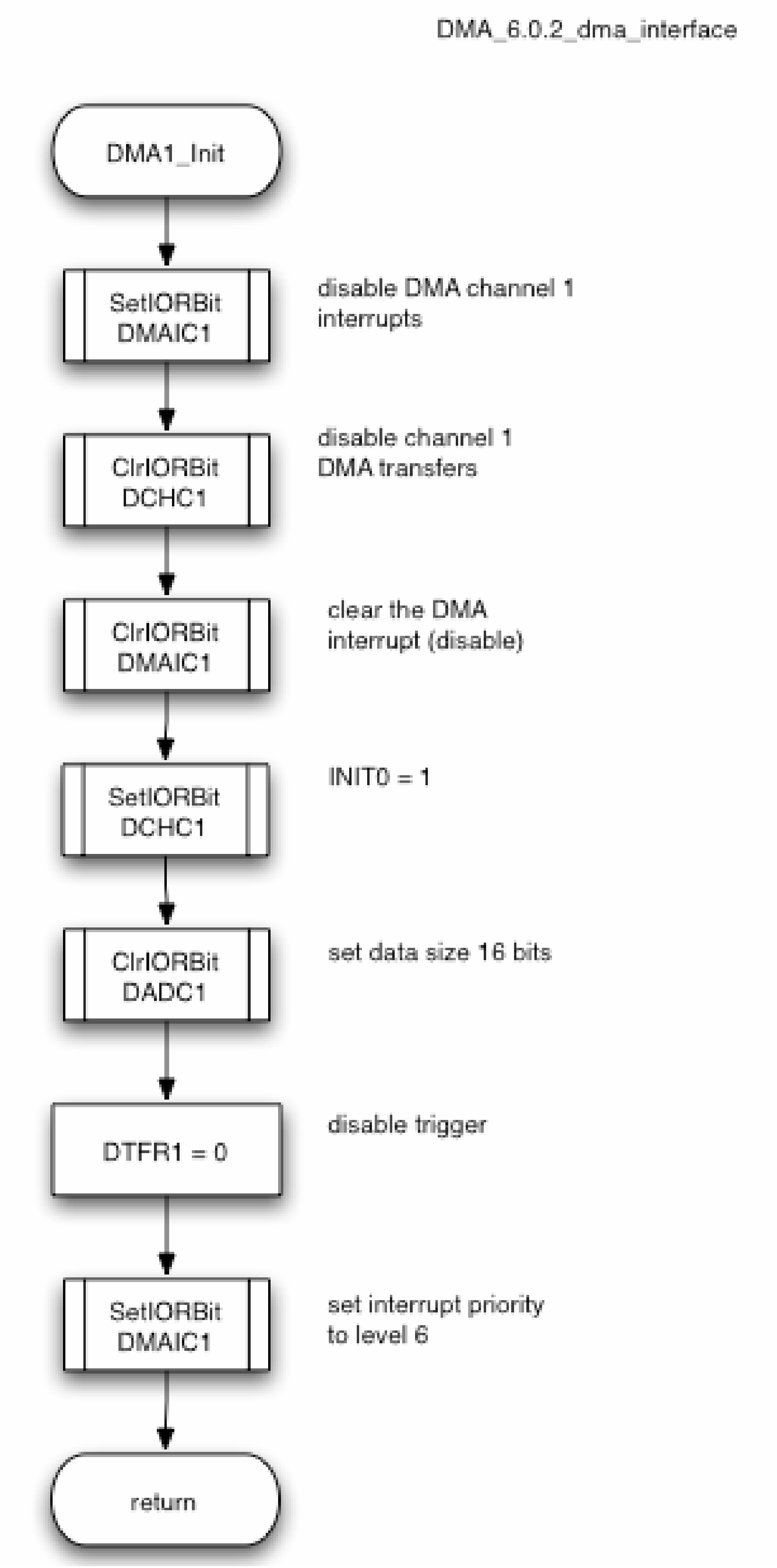
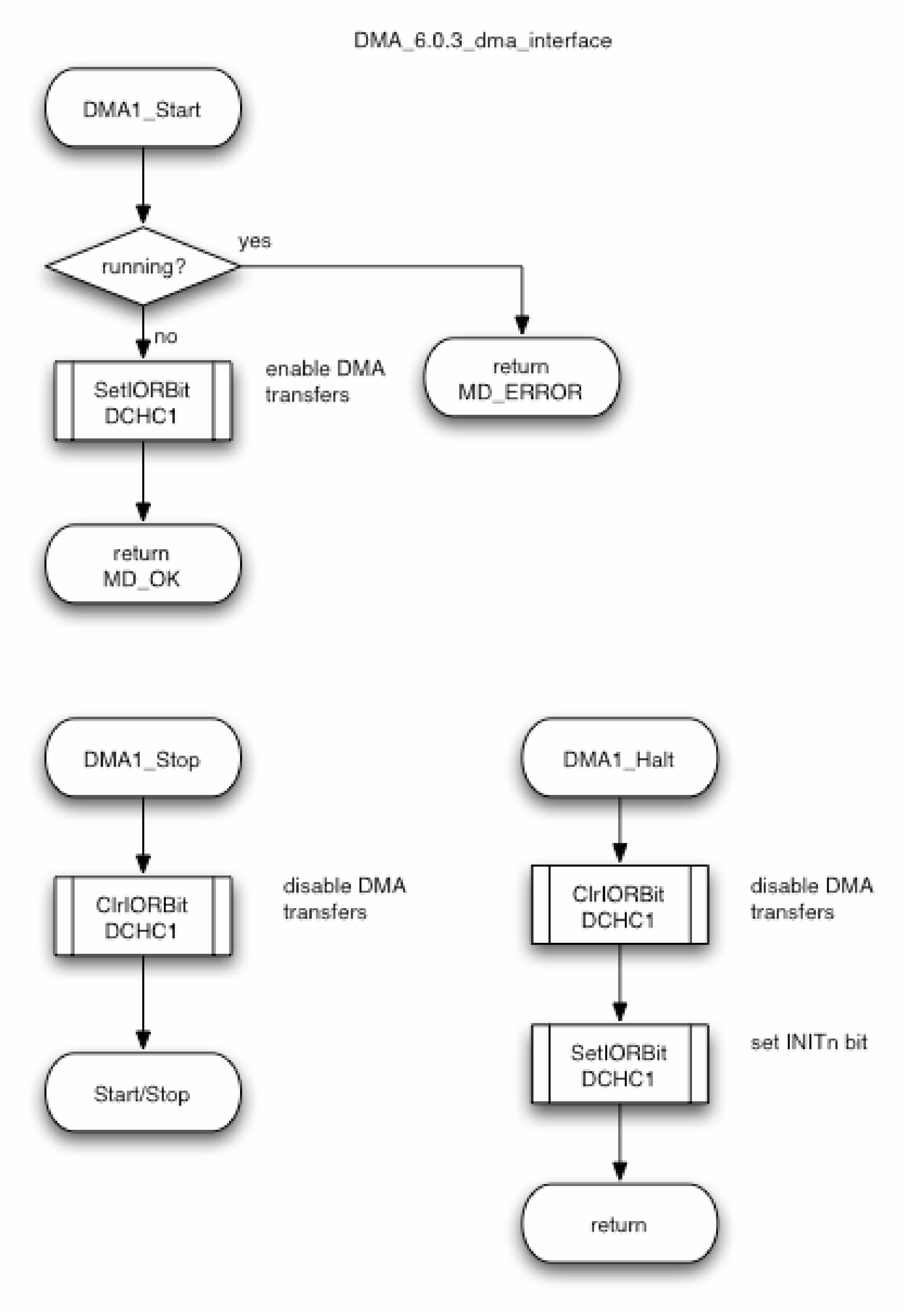


Figure 107. DMA_6.0.3_dma_interface



8.11 DMA Interface_User

Figure 108. DMA_6.1.0_dma_interface_user

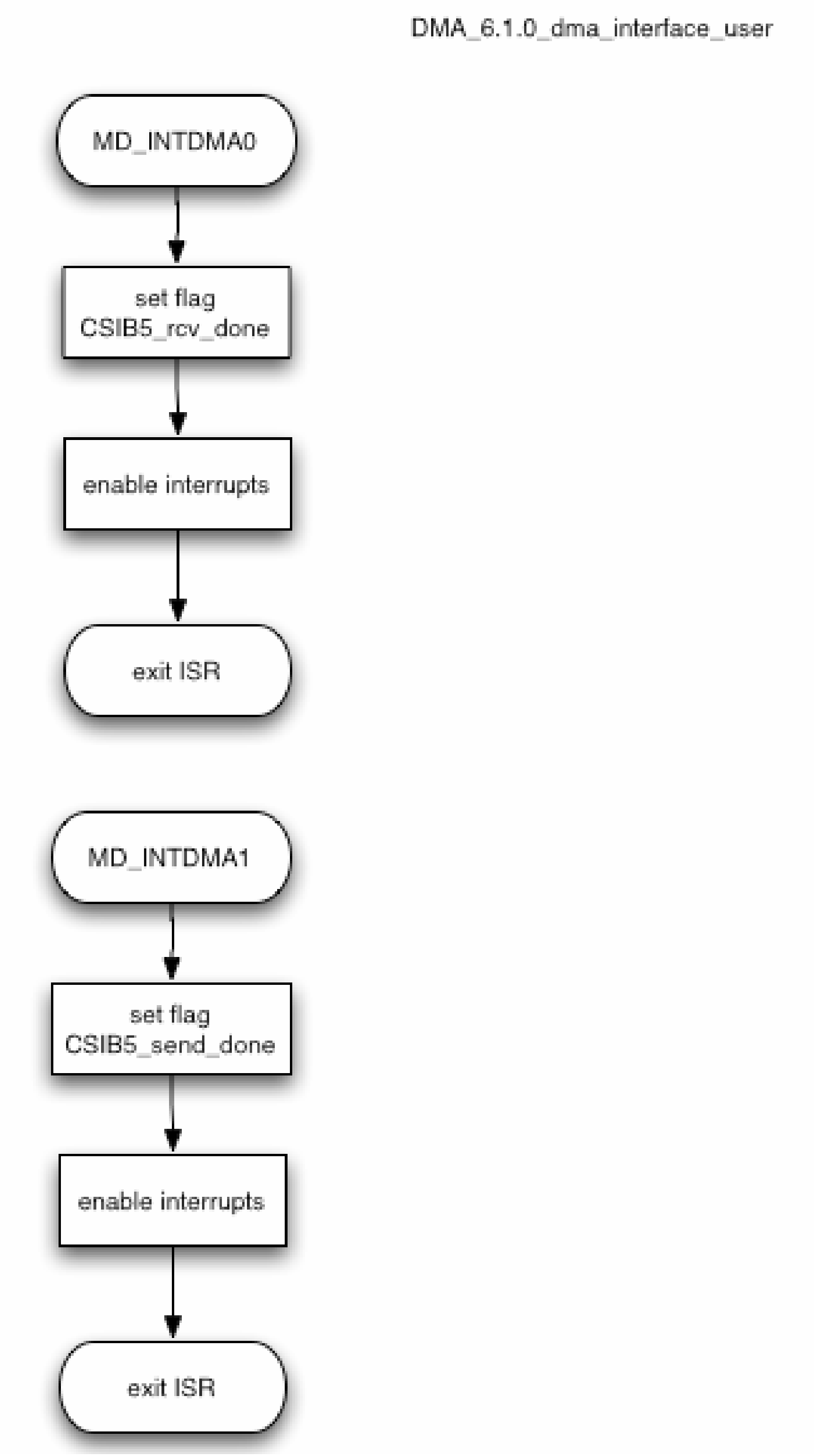


Figure 109. DMA_6.1.1_dma_interface_user

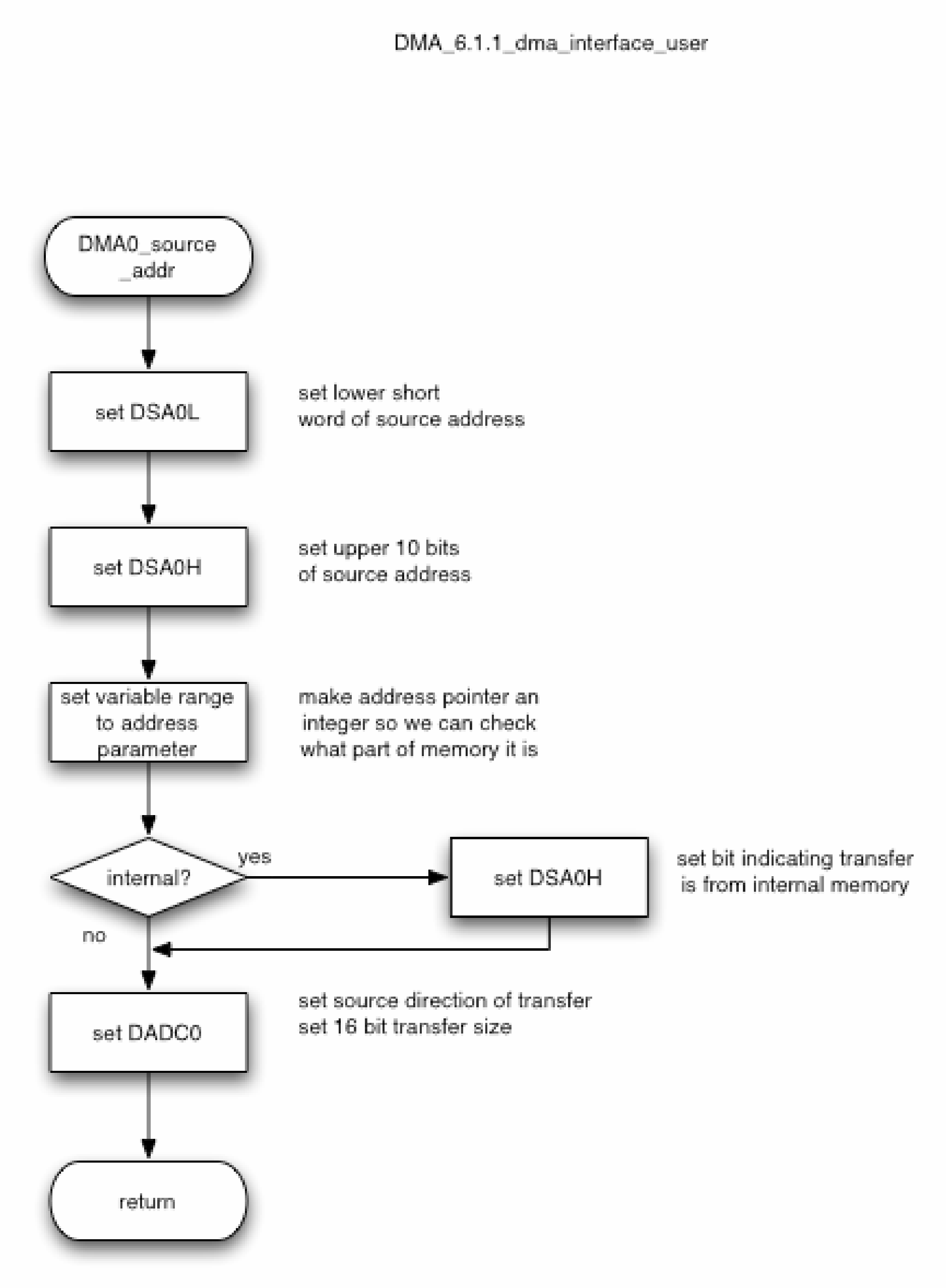

DMA_6.1.1_dma_interface_user
DMA0_source _addr
set DSA0L
set lower short word of source address
set DSA0H
set upper 10 bits of source address
set variable range to address parameter
make address pointer an integer so we can check what part of memory it is
internal?
yes
no
set DSA0H
set bit indicating transfer is from internal memory
set DADC0
set source direction of transfer set 16 bit transfer size
return

Figure 110. DMA_6.1.2_dma_interface_user

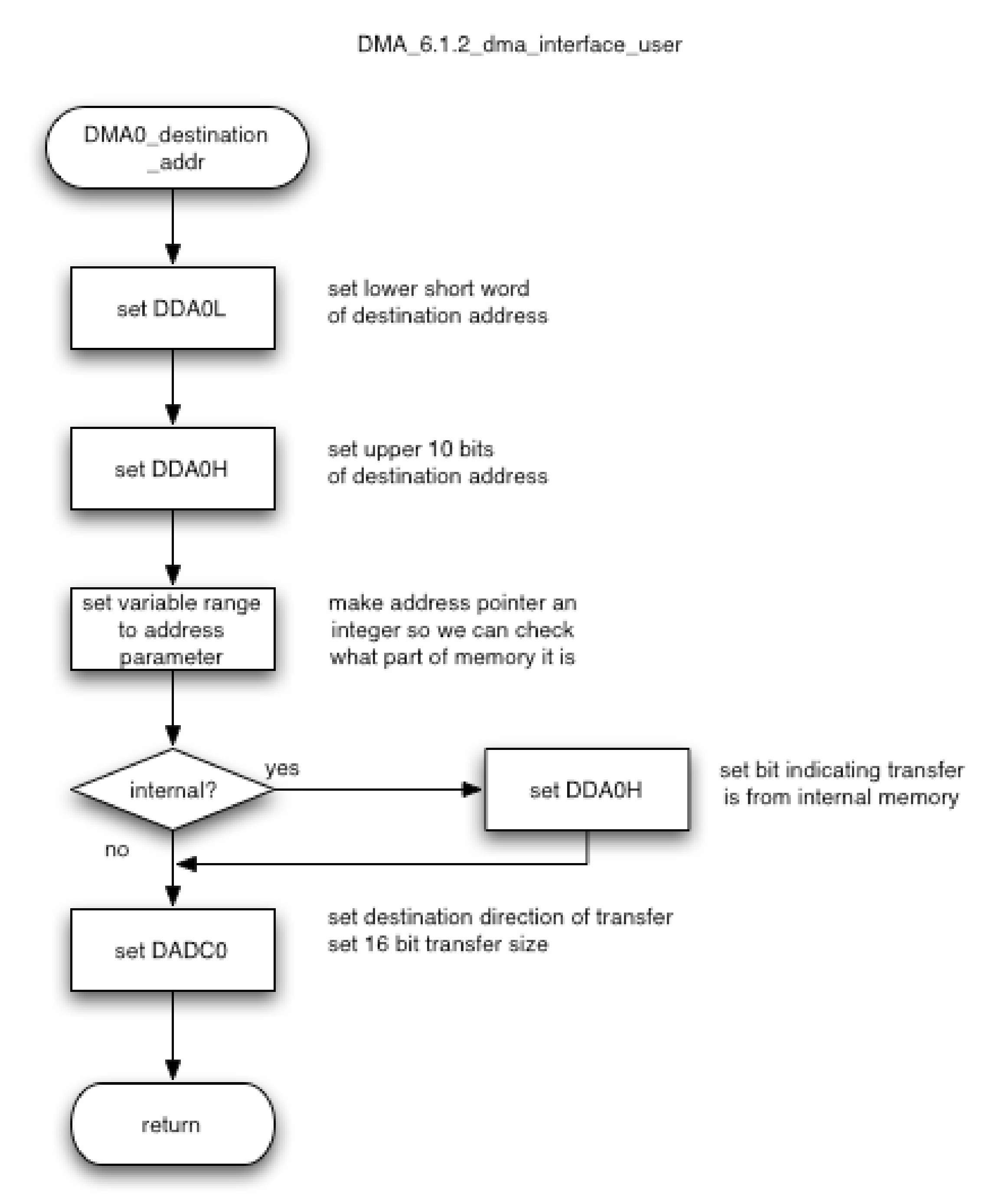

DMA_6.1.2_dma_interface_user
DMA0_destination _addr
set DDA0L
set lower short word of destination address
set DDA0H
set upper 10 bits of destination address
set variable range to address parameter
make address pointer an integer so we can check what part of memory it is
internal?
yes
set DDA0H
set bit indicating transfer is from internal memory
no
set DADC0
set destination direction of transfer set 16 bit transfer size
return

Figure 111. DMA_6.1.3_dma_interface_user

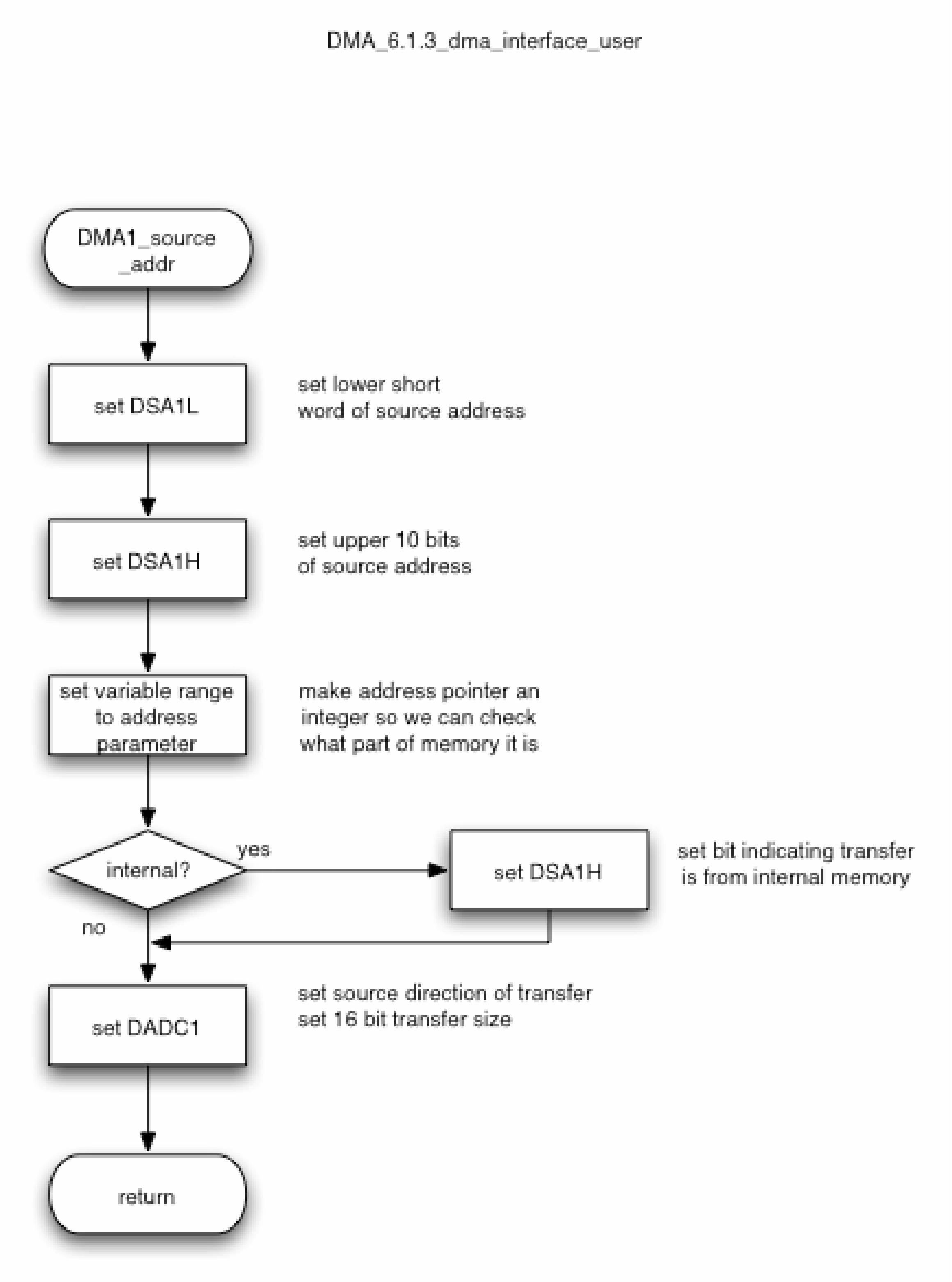

DMA_6.1.3_dma_interface_user
DMA1_source _addr
set DSA1L
set lower short word of source address
set DSA1H
set upper 10 bits of source address
set variable range to address parameter
make address pointer an integer so we can check what part of memory it is
internal?
yes
no
set DSA1H
set bit indicating transfer is from internal memory
set DADC1
set source direction of transfer set 16 bit transfer size
return

Figure 112.   DMA_6.1.4_dma_interface_user

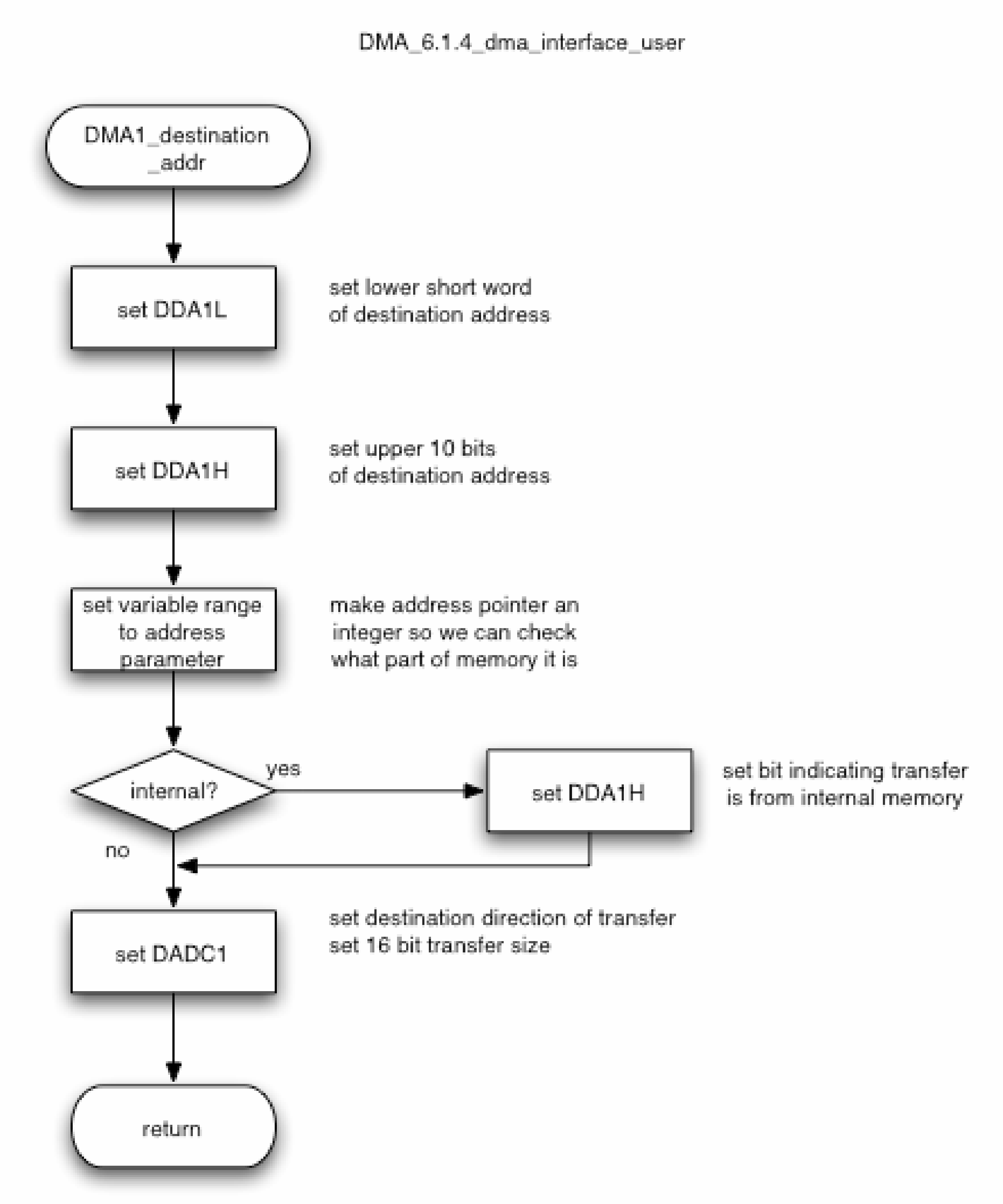

DMA_6.1.4_dma_interface_user
DMA1_destination_addr
set DDA1L
set lower short word of destination address
set DDA1H
set upper 10 bits of destination address
set variable range to address parameter
make address pointer an integer so we can check what part of memory it is
internal?
yes
no
set DDA1H
set bit indicating transfer is from internal memory
set DADC1
set destination direction of transfer set 16 bit transfer size
return

Figure 113. DMA_6.1.5_dma_interface_user

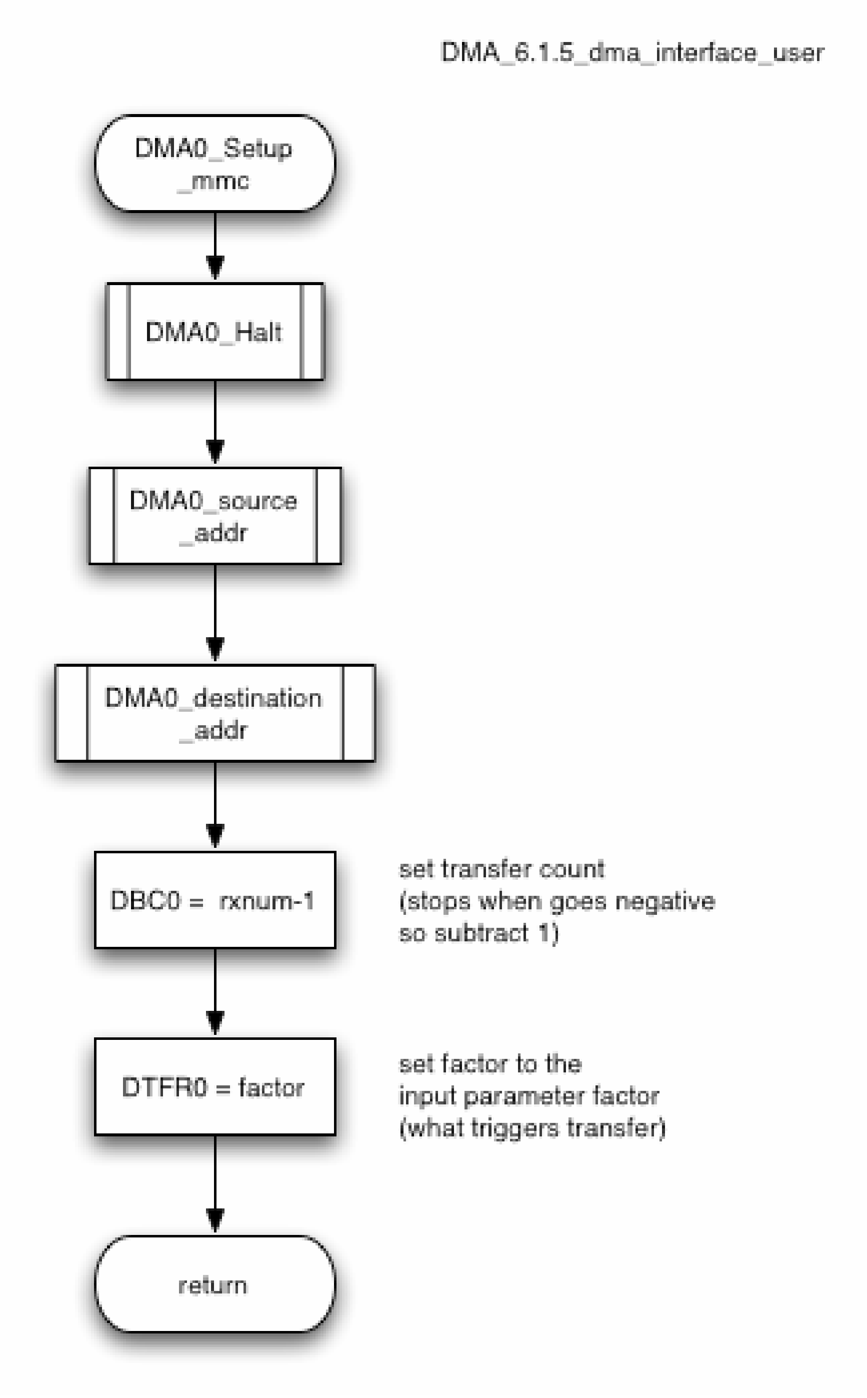
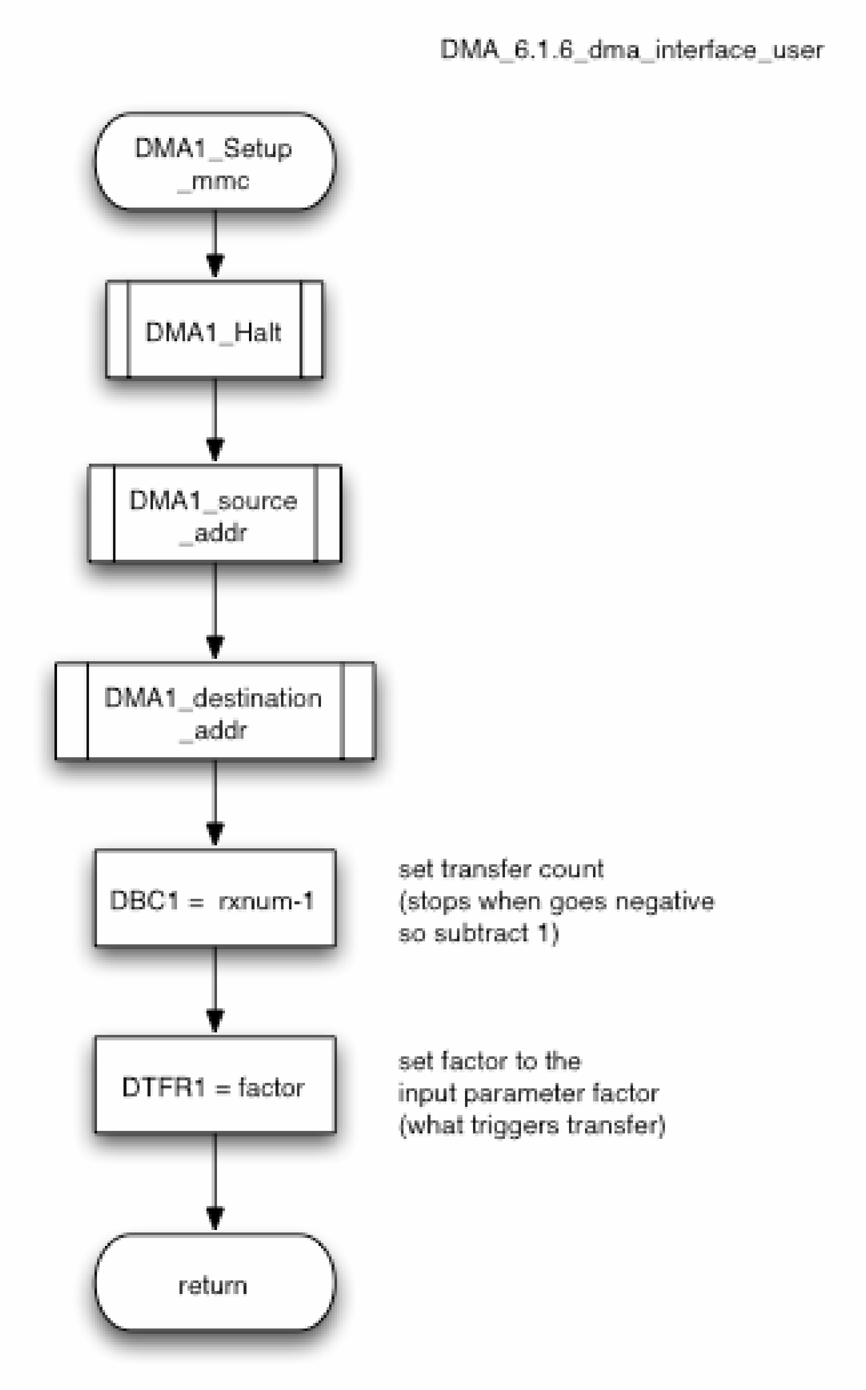

DMA_6.1.5_dma_interface_user
DMA0_Setup _mmc
DMA0_Halt
DMA0_source _addr
DMA0_destination _addr
DBC0 = rxnum-1
set transfer count (stops when goes negative so subtract 1)
DTFR0 = factor
set factor to the input parameter factor (what triggers transfer)
return

Figure 114. DMA_6.1.6_dma_interface_user


DMA_6.1.6_dma_interface_user
DMA1_Setup _mmc
DMA1_Halt
DMA1_source _addr
DMA1_destination _addr
DBC1 = rxnum-1
set transfer count (stops when goes negative so subtract 1)
DTFR1 = factor
set factor to the input parameter factor (what triggers transfer)
return

8.12 A/D Interface

Figure 115. DMA_7.0_a_to_d_interface

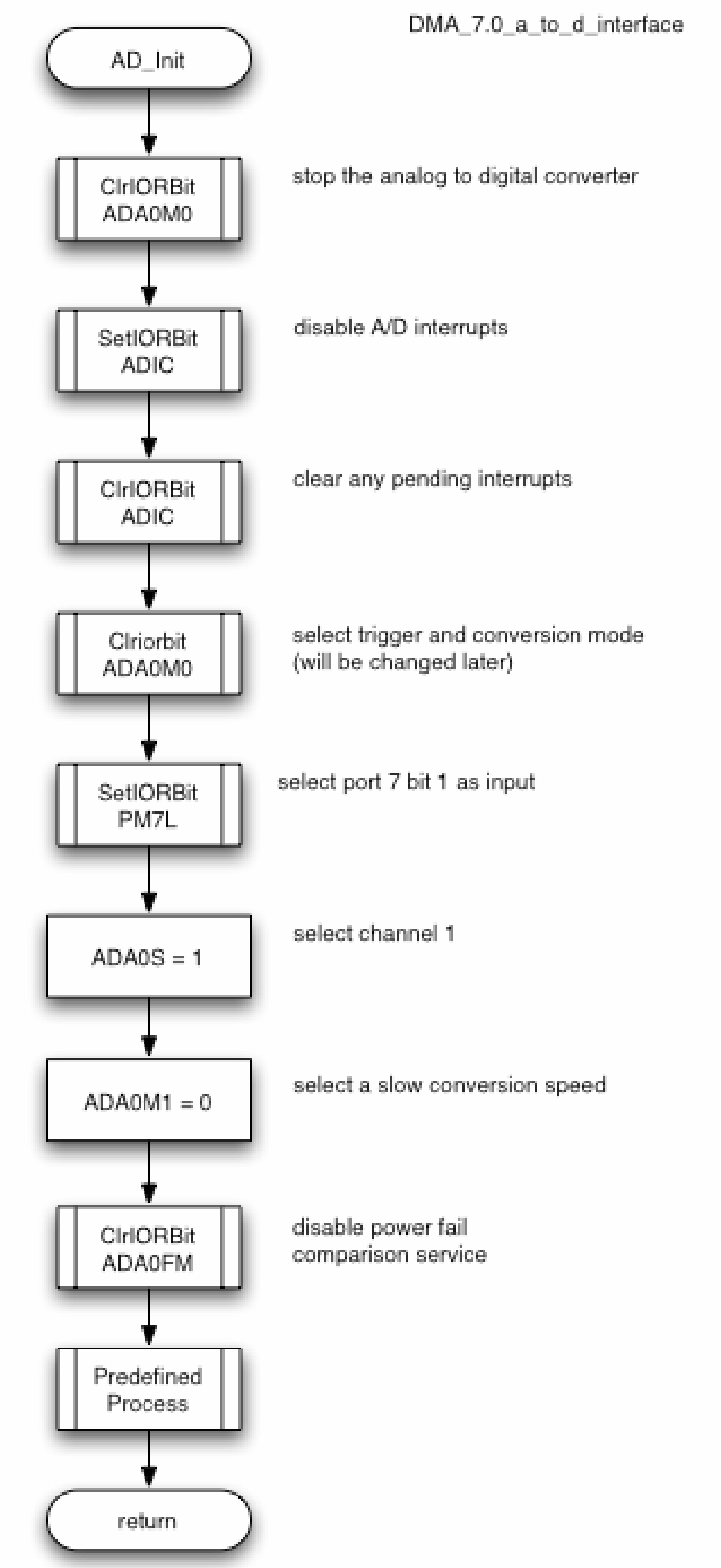
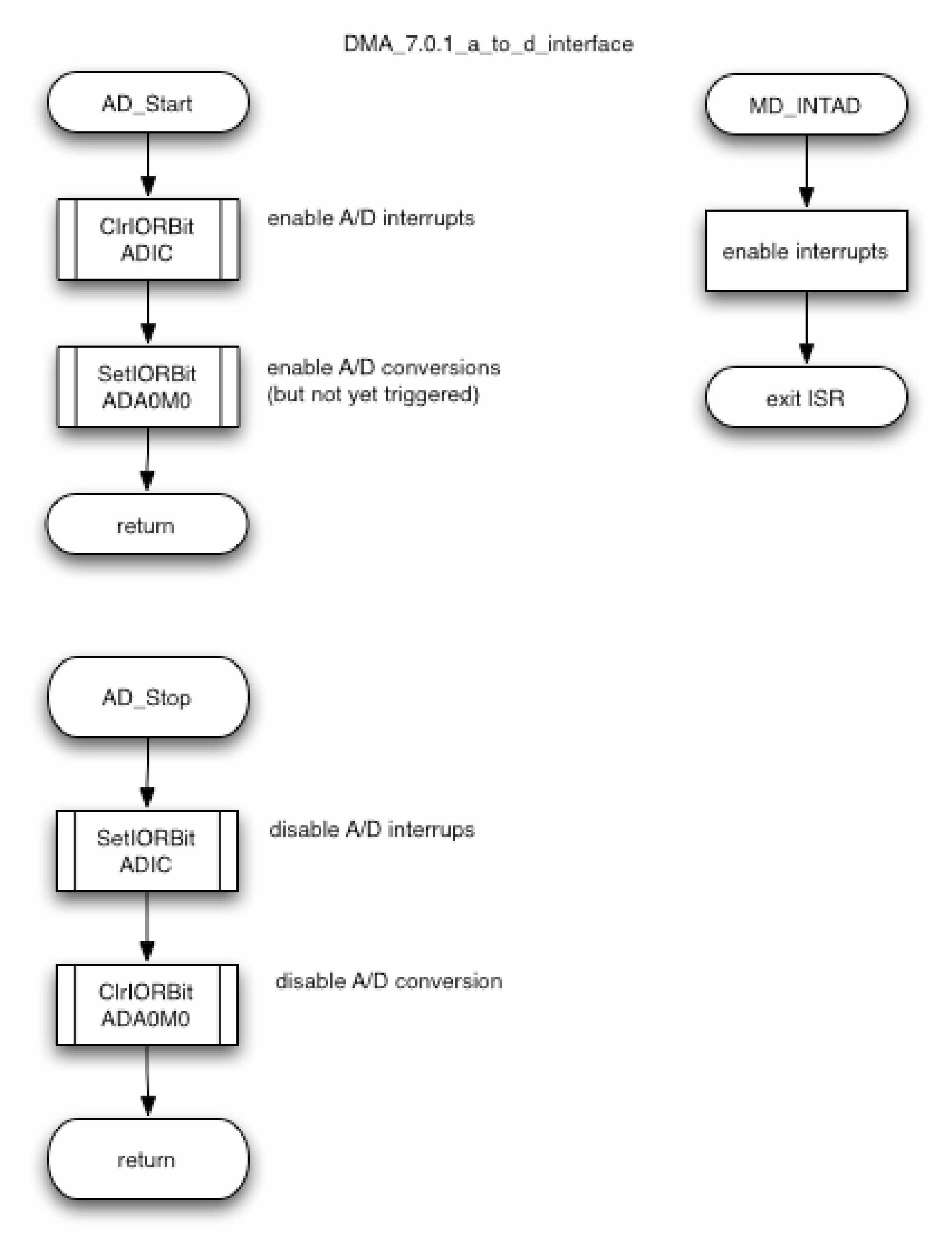


Figure 116.   DMA_7.0.1_a_to_d_interface



9. Appendix B — Source Code Listings

The demonstration source code includes the following files:

- crte.s
- system.s
- inttab.s
- systeminit.c
- macrodriver.h
- ad.c
- ad.h
- bus.c
- bus.h
- dma.c
- dma_user.c
- dma.h
- int.c
- int_user.c
- int.h
- main.c
- port.c
- system.inc
- system_user.c
- port.h
- sdmemory.c
- sdmemory.h
- serial.c
- serial_user.c
- serial.h
- timer.c
- timer_user.c
- timer.h

9.1 crte.s

```
#   Copyright (C) NEC Electronics Corporation 1998,2002
#   NEC ELECTRONICS CONFIDENTIAL AND PROPRIETARY
#   All rights reserved by NEC Electronics Corporation.
#   This program must be used solely for the purpose for which
#   it was furnished by NEC Electronics Corporation.  No part of this
#   program may be reproduced or disclosed to others, in any
#   form, without the prior written permission of NEC Electronics
#   Corporation.  Use of copyright notice does not evidence
#   publication of the program.

#   @(#)crtE.s  1.8 02/12/12 15:19:37

#=======================================================================
# NAME
#    crtE.s -  start up module for ca850(V850E)
#
# DESCRIPTIONS:
#      This assembly program is a sample of start-up module for ca850(V850E).
#    If you modified this program, you must assemble this file, and
#    locate a given directory.
#
#    Unless -G is specified, sections are located as the following.
#
#                  |        :        |
#                  |        :        |
#          tp -> -+-----------------+ __start     __tp_TEXT
#                  |  start up       |
#                  |----------------  |
#   text section   |                 |
#                  | user program    |
#                  |                 |
#                  |-----------------|
#                  | library         |
#                 -+-----------------+
#                  |        :        |
#                  |        :        |
#                 -+-----------------+ __argc
#                  |      0          |
#                  |---------------- | __argv
#   data section   |    #.L16        |
#                  |---------------- | .L16
#                  | 0x0,0x0,0x0,0x0 |
#                 -+-----------------+
#                  |                 |
#   sdata section  |                 |
#                  |                 |
#           gp-> -+-----------------+             __ssbss
#                  |                 |
#   sbss section   |                 |
#                  |                 |
#                  +-----------------+ __stack     __esbss     __sbss
#                  | stack area      |
#   bss section    |                 |
#                  |  0x200 bytes    |
#            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
    .extern _BUS_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, 0x880
    .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
    jarl    _BUS_Init, lp       -- call BUS_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 -------------------#
#                               #
```

9.2 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*fx
st.b    r11, PRCMD[r0]
st.b    r11, CKC[r0]
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]
```

9.3 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 "INTTQ0OV", text      --TQ0OV                0x0110
    reti
    .section "INTTQ0CC0", text     --TQ0CC0
    reti
    .section "INTTQ0CC1", text     --TQ0CC1
    reti
    .section "INTTQ0CC2", text     --TQ0CC2
    reti
    .section "INTTQ0CC3", text     --TQ0CC3
    reti
    .section "INTTP0OV", text      --TP0OV                0x0160
    reti
--  .section "INTTP0CC0", text     --TP0CC0   use timer.c definition
--  reti
    .section "INTTP0CC1", text     --TP0CC1
    reti
    .section "INTTP1OV", text      --TP1OV
    reti
    .section "INTTP1CC0", text     --TP1CC0
    reti
    .section "INTTP1CC1", text     --TP1CC1
    reti
    .section "INTTP2OV", text      --TP2OV
    reti
--  .section "INTTP2CC0", text     --TP2CC0   use timer.c definitions
--  reti
    .section "INTTP2CC1", text     --TP2CC1
    reti
    .section "INTTP3OV", text      --TP3OV
    reti
    .section "INTTP3CC0", text     --TP3CC0
    reti
    .section "INTTP3CC1", text     --TP3CC1
    reti
    .section "INTTP4OV", text      --TP4OV
    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  defined in a to d initialization
--  reti

--  .section "INTDMA0", text        --INTDMA0 defined in dma initialization
--  reti

--  .section "INTDMA1", text        --INTDMA1 defined in dma initialization
--  reti

    .section "INTDMA2", text        --INTDMA2
    reti
    .section "INTDMA3", text        --INTDMA3
    reti
    .section "INTKR", text          --INTKR
    reti
    .section "INTWTI", text         --INTWTI
    reti
    .section "INTWT", text          --INTWT
    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 "INTCB5T", text        --INTCB5T
--  reti

-- end of file
```

9.4 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 "bus.h"
#include "ad.h"
#include "port.h"
#include "timer.h"
#include "serial.h"
#include "dma.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) 1wait */

    PORT_Init( );           /* Port initiate, call first as other   */
                            /* functions like to overwrite settings */
    UART3_Init( );          /* UART3 initiate */
    CSIB5_Init( );          /* CSIB5 initiate */
    AD_Init( );             /* AD initiate */
    BUS_Init();             /* external memory initiate */
    TMP0_Init();            /* Timer 0 initiate */
    TMP2_Init();            /* Timer 2 initiate */
    DMA0_Init( );           /* DMA0 initiate */
    DMA1_Init( );           /* DMA1 initiate */
    __EI( );                /* enable interrupt */
}
```

9.5 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:  uPD70F3721
**
**  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 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+0x8   /* IIC no ACK*/
#define MD_SLAVE_SEND_END   MD_STATUSBASE+0x9   /* 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 fucntion 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
```

9.6 ad.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 : ad.c
**  Abstract : This file implements a device driver for the AD module
**  APIlib: v850esJx2.lib V1.50 [23 Feb. 2006]
**
**  Device:  uPD70F3721
**
**  Compiler:  NEC/CA850
**
********************************************************************************
*/
/*
********************************************************************************
** Include files
********************************************************************************
*/
#include "macrodriver.h"
#include "ad.h"
#pragma interrupt INTAD MD_INTAD
/*
**------------------------------------------------------------------------------
**
**  Abstract:
**     A/D control initialization settings
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**------------------------------------------------------------------------------
*/
void AD_Init( void )
{
       ClrIORBit(ADA0M0, 0x80);             /* stop the A/D converter before making changes
*/
       SetIORBit(ADIC, 0x40);               /* disable interrupt servicing */
       ClrIORBit(ADIC, 0x80);               /* clear previous interrupts, if any */
       ClrIORBit(ADA0M0, 0x02);             /* software trigger mode             */
       ClrIORBit(ADA0M0, 0x30);             /* select one shot conversion mode   */
       SetIORBit(PM7L, 0x02 );              /* bit 1 of port 7 lower is input connected to
potentiometer */
```

```
      ADA0S = 0x01;                        /* channel 1 selected */
      ADA0M1 = 0x00;                       /* selection conversion time 65/fxx */
      ClrIORBit(ADA0PFM, 0xc0);            /* disable power fail comparison service */
      SetIORBit(ADIC, 0x07);               /* interrupt priority setting level 7 */
      return;
}

/*
**--------------------------------------------------------------------------
**
**  Abstract:
**    This function enables the interrupt starts the A/D converter .
**
**  Input Parameters:
**        none
**  Returns:
**          MD_OK
**
**---------------------------------------------------------------------------
*/
MD_STATUS AD_Start( void )
{
      ClrIORBit(ADIC, 0x40);       /* enable interrupt servicing */

      SetIORBit(ADA0M0, 0x80);     /* enable A/D conversion */
      return MD_OK;
}

/*
**---------------------------------------------------------------------------
**
**  Abstract:
**    This function can be called after an A/D conversion is completed,
**    and returns the conversion result(s) in the buffer. It is not used
**      in this demonstration as all the read operations are performed by
**      the DMA controller.
**
**  Input Parameters:
**    USHORT* buffer : The address where to write the conversion result.
**
**  Returns:
**    MD_OK
**
**-----------------------------------------------------------------------------
* this function is not used in this demonstration program as the dma controller
* will read the data from the A/D results register */
MD_STATUS AD_Read( USHORT* buffer )
{
      USHORT AD_Reg;
      USHORT i;
      if( (ADA0M0 & 0x10) == 0 ){          /* select mode */
          AD_Reg = *( volatile unsigned short* )((AD_CHANNEL<<1) + ADCR_BASE);
          *buffer = ( AD_Reg >> 6 );
      }
      else{                    /* scan mode */
          for(i=0; i<=AD_CHANNEL; i++){
              AD_Reg = *( volatile unsigned short* )( (i<<1)  + ADCR_BASE );
              *buffer = ( AD_Reg >> 6 );
              buffer++;
          }
      }
      return MD_OK;
}

/*
**---------------------------------------------------------------------------
**
**  Abstract:
**  This function stops the A/D converter.
**
**  Input Parameters:
**    None
**
**  Returns:
**    None
**
**----------------------------------------------------------------------------
*/
void AD_Stop( void )
{

      SetIORBit(ADIC, 0x40);          /* disable interrupt servicing */

      ClrIORBit(ADA0M0, 0x80);
      return;
}

/*
**---------------------------------------------------------------------------
**
**  Abstract:
**    INTAD Interrupt service routine
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**---------------------------------------------------------------------------
*/
__interrupt void MD_INTAD( void )
{
/*TODO. Add user defined interrupt service routine */
    __EI();    /* enable the a to d interrupt */
}
```

FILEID: ad.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 : ad.h
**  Abstract : This file implements the device driver for the AD module
**  APIlib: v850esJx2.lib V1.50 [23 Feb. 2006]
**
**  Device:  uPD70F3721
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
*/
#ifndef    _MDAD_
#define    _MDAD_
/*
** **************************************************************************
** MacroDefine
** **************************************************************************
*/
#define  AD_CHANNEL    0

#define ADCR_BASE  0xfffff210
#define ADCRH_BASE 0xfffff211

enum AD_Channel { channel0, channel1, channel2, channel3,
       channel4, channel5, channel6, channel7,
       channel8, channel9, channel10, channel11,
       channel12, channel13, channel14, channel15
       };

enum INTMode{INTMode0, INTMode1, INTMode2};
void AD_Init ( void );
MD_STATUS AD_Start( void );
MD_STATUS AD_Read( USHORT*  buffer );
void AD_Stop( void );
__interrupt  void MD_INTAD( void );
#endif
```

9.7 bus.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 : bus.c
**  Abstract : This file implements a device driver for the bus module
**  APIlib :   V850ESJx2.lib V1.50 [23 Feb. 2006]
**
**  Device :   uPD70F3721
**
**  Compiler : NEC/CA850
**
*******************************************************************************
*/
/*
*******************************************************************************
**  Include files
*******************************************************************************
*/
#include "macrodriver.h"
#include "bus.h"
/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**    Bus width, bus wait insert, bus idle insert configurator setting
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-----------------------------------------------------------------------------
*/
void BUS_Init( void )
{
      PMCCM = 0x00;                    /* not using clkout, hldrq, hldak or wait */
      PMCCT = 0x13;                    /* wr0, wr1, rd*/
      PMCCS = 0x02;                    /* cs1 */
      EXIMC=1;                         /* seperate bus mode (not multiplexed)*/
      __asm("nop");
      BSC = 0x5551;                    /* bus size config register BSC */
                                       /* /cs1 = 8 bit memory bank */
      DWC0 = 0x3373;                   /* data wait control register DWC0 - bank1 wait7*/
      AWC  = 0xfff3;                   /* address wait control register AWC */
      BCC  = 0xaaa2;                  /* bus cycle control register BCC */

      PMCDH = 0xff;                   /* set PMCDH alt function A16-23 */
      PMCDL = 0x00ff;                 /* set PMCDL for 8bit data bus */
      PFC9  = 0x0000;                 /* set PFC9 for address line output */
      PFCE9 = 0x0000;                 /* set PFCE9 for address line output */
      PMC9  = 0xffff;                 /* set PMC9 for address option */
    PF9   = 0x0000;                   /* normal cmos levels */
}

#if 0
/* dump memory manager - bus register info */
void bus_dump(void)
{

// PDL0-15
sprintf(msg_buf,"\r\nPMCDL  0x%04x PMCDH  0x%02x\r\n",PMCDL,PMCDH);
uart3_tx_msg(msg_buf);
// P90-15
sprintf(msg_buf,    "PMC9   0x%04x PFC9  0x%04x PFCE9  0x%04x PF9
0x%04x\r\n",PMC9,PFC9,PFCE9,PF9);
uart3_tx_msg(msg_buf);
// PDH0-PDH7
sprintf(msg_buf,    "PMCCM    0x%02x PMCCT    0x%02x  PMCCS
0x%02x\r\n",PMCCM,PMCCT,PMCCS);
uart3_tx_msg(msg_buf);
sprintf(msg_buf,    "PCM      0x%02x PMCM     0x%02x\r\n",PCM,PMCM);
uart3_tx_msg(msg_buf);
// 0xffffffbe  EXIMC[SMSEL] lsb = 1
sprintf(msg_buf,"EXIMC    0x%02x\r\n",EXIMC);
uart3_tx_msg(msg_buf);

// 0xfffff066  BSC
sprintf(msg_buf,"BSC    0x%04x\r\n",BSC);
uart3_tx_msg(msg_buf);
// DWC0
sprintf(msg_buf,"DWC0   0x%04x\r\n",DWC0);
uart3_tx_msg(msg_buf);

    sprintf(msg_buf,"AWC    0x%04x\r\n",AWC);
    uart3_tx_msg(msg_buf);
    // AWC
    sprintf(msg_buf,"BCC    0x%04x\r\n",BCC);
    uart3_tx_msg(msg_buf);
}
#endif
```

FILE ID: bus.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 : bus.h
**  Abstract : This file implements a device driver for the bus module
**  APIlib :   V850ESJx2.lib V1.50 [23 Feb. 2006]
**
**  Device :   uPD70F3721
**
**  Compiler : NEC/CA850
**
*******************************************************************************
*/
#ifndef    _MDBUS_
#define    _MDBUS_
void BUS_Init(void);
#endif
```

9.8 dma.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 :  dma.c
**  Abstract :  This file implements a device driver for the dma module
**  APIlib :V850ESJx2.lib V1.50 [23 Feb. 2006]
**
**  Device :uPD70F3721
**
**  Compiler :  NEC/CA850
**
*******************************************************************************
*/
/*
*******************************************************************************
** Include files
*******************************************************************************
*/
#include "macrodriver.h"
#include "dma.h"

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
** This function is used to initialize the DMA0 module.
**
**  Parameters:
** None
**
**  Returns:
** None
**
**-----------------------------------------------------------------------------
*/
void DMA0_Init( void )
{
SetIORBit(DMAIC0, 0x40);      /* disable interrupt */
ClrIORBit(DCHC0,  0x01);        /* disable the DMA transfer */
ClrIORBit(DMAIC0, 0x80);      /* clear the DMA interrupt */
SetIORBit(DCHC0, 0x04);       /* INIT0 bit is set to 1 */
ClrIORBit(DADC0, 0x4000);     /* data size--16 bits */
//ClrIORBit(DADC0, 0x00c0);      /* source address count increment */
//ClrIORBit(DADC0, 0x30);        /* destination address count increment */
DTFR0 = 0x00;           /* disable trigger */
```

```
 SetIORBit(DMAIC0, 0x05);         /* Interrupt Priority Setting Level 5 */
 //ClrIORBit(DMAIC0, 0x40);    /* Enable Interrupt Servicing */
 return;
}

/*
**------------------------------------------------------------------------------
**
**  Abstract:
** This function starts the DMA.
**
**  Parameters:
** None
**
**  Returns:
** MD_OK
** MD_ERROR
**
** MEMO:
** if DMA still transfer, function return value will be MD_ERROR.
**
**------------------------------------------------------------------------------
*/
MD_STATUS DMA0_Start( void )
{
 if( (DCHC0 & 0x81) == 1 ){ // check if still doing transfer
    return MD_ERROR;
 }
 SetIORBit(DCHC0, 0x1);    // set the Enn bit, DMA transfer enabled
 return MD_OK;
}

/*
**------------------------------------------------------------------------------
**
**  Abstract:
** This function stops the DMA.
**
**  Parameters:
** None
**
**  Returns:
** None
**
**------------------------------------------------------------------------------
*/
void DMA0_Stop( void )
{
 ClrIORBit(DCHC0, 0x1);  /* DMA transfer disabled */
 return;
}

void DMA0_Halt(void)
{
 ClrIORBit(DCHC0, 0x01);  /* DMA transfer disable */
 SetIORBit(DCHC0, 0x04);  /* initialize bit */
}
/*
```

```
**-----------------------------------------------------------------------------
**
**  Abstract:
** This function is used to initialize the DMA1 module for CSIB5 output.
**
**  Parameters:
** None
**
**  Returns:
** None
**
**-----------------------------------------------------------------------------
*/
void DMA1_Init( void )
{
SetIORBit(DMAIC1, 0x40);        /* disable interrupt */
DCHC1 = 0x01;                   /* disable the DMA transfer, clear bits 3..6 */
ClrIORBit(DMAIC1, 0x80);        /* clear the DMA interrupt */
SetIORBit(DCHC1, 0x04);       /* INIT1 bit is set to 1 */
    SetIORBit(DADC1,0x4000);        /* data size--16 bits */
// ClrIORBit(DADC1, 0x00c0);     /* source address count increment */
// SetIORBit(DADC1, 0x20);       /* destination address count fix */
DTFR1 = 0x00;                   /* disable trigger */
SetIORBit(DMAIC1, 0x06);        /* Interrupt Priority Setting Level 6 */
// ClrIORBit(DMAIC1, 0x40);        /* Enable Interrupt Servicing */
return;
}

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
** This function starts the DMA.
**
**  Parameters:
** None
**
**  Returns:
** MD_OK
** MD_ERROR
**
** MEMO:
** if DMA still transfer, function return value will be MD_ERROR.
**
**-----------------------------------------------------------------------------
*/
MD_STATUS DMA1_Start( void )
{
if( (DCHC1 & 0x81) == 0x01 ){ // check if still doing transfer
   return MD_ERROR;
}
SetIORBit(DCHC1, 0x01);     // enable dma transfer
return MD_OK;
}

/*
**-----------------------------------------------------------------------------
**
```

```
**  Abstract:
** This function stops the DMA.
**
**  Parameters:
** None
**
**  Returns:
** None
**
**----------------------------------------------------------------------------
*/
void DMA1_Stop( void )
{
 ClrIORBit(DCHC1, 0x01);
 return;
}

void DMA1_Halt(void)
{
 ClrIORBit(DCHC1, 0x01);
 SetIORBit(DCHC1, 0x04);  /* this is not done */
}
```

FILE ID: dma_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 : dma_user.c
**  Abstract : This file implements a device driver for the DMA module
**  APIlib :   V850ESJx2.lib V1.50 [23 Feb. 2006]
**
**  Device :   uPD70F3721
**
**  Compiler : NEC/CA850
**
*******************************************************************************
*/
/*
*******************************************************************************
**  Include files
*******************************************************************************
*/
#include <stdlib.h>
#include "macrodriver.h"
#include "dma.h"
#include "serial.h"

#pragma interrupt INTDMA0  MD_INTDMA0  /* vector set */
#pragma interrupt INTDMA1  MD_INTDMA1  /* vector set */

extern volatile int CSIB5_rcv_done;  /* in serial_user.c */
extern volatile int CSIB5_send_done; /* in serial_user.c */

//char msg_buf[60]; global in main

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
** DMA0 INTDMA0 Interrupt service routine
**
**  Parameters:
** None
**
**  Returns:
** None
**
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTDMA0( void )
{
  CSIB5_rcv_done = 1;
    __EI();
}

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
** DMA1 INTDMA1 Interrupt service routine
**
**  Parameters:
** None
**
**  Returns:
** None
**
**-----------------------------------------------------------------------------
*/
__interrupt void MD_INTDMA1( void )
{
    __EI();       /* enable interrupts */
  CSIB5_send_done = 1;

}

/*********************************************************************/
/* Function:    DMA0_source_addr                                     */
/* Description: set DMA0 source address. figure out internal or      */
/*              external based on address                            */
/* Input:       address  - DMA0 source address                       */
/*              mode     - increment, decrement or fixed address     */
/* Return:      none                                                 */
/*********************************************************************/
void DMA0_source_addr(unsigned char *address, unsigned short mode)
{
    unsigned int range;
    unsigned short temp;

    DSA0L = ((int)address & 0x0000FFFF);
    DSA0H = (((int)address >> 16) & 0x000003FF);
    range = (unsigned int)address;
    /* check for internal or external, default is external */

    if(range >= 0x03ff0000 && range < 0x03ffefff)
        DSA0H |= 0x8000;  // internal RAM
    /* set count direction of transfer source address */
    temp = DADC0;
    temp &= 0xff3f;    /* clear source direction of transfer */
    temp |= (mode & 0x0003) << 6;
    temp |= 0x4000;    /* 16 bit transfer */
    DADC0 = temp;
}

/*********************************************************************/
/* Function:    DMA0_destination_addr                                */
/* Description: set DMA0 destination address, figure out internal or */
/*              external from address. set count mode                */
/* Input:       address  - DMA0 destination address                  */
/*              mode     - increment, decrement or fixed address     */
/* Return:      none                                                 */
/*********************************************************************/
void DMA0_destination_addr(unsigned char *address, unsigned short mode)
{
    unsigned int range;
    unsigned short temp;

    DDA0L = ((int)address & 0x0000FFFF);
    DDA0H = (((int)address >> 16) & 0x000003FF);
    range = (unsigned int)address;
    /* check for internal or external, default is external */
    if(range >= 0x03ff0000 && range < 0x03ffefff)
        DDA0H |= 0x8000;  // internal RAM
    /* set count direction of transfer source address */
    temp = DADC0;
    temp &= 0xffcf;    /* clear destination direction of transfer */
    temp |= (mode & 0x0003) << 4;
    temp |= 0x4000;    /* 16 bit transfer */
    DADC0 = temp;
}

/*********************************************************************/
/* Function:    DMA1_source_addr                                     */
/* Description: set DMA1 source address                              */
/* Input:       address  - DMA1 source address                       */
/*              mode     - increment, decrement or fixed address     */
/* Return:      none                                                 */
/*********************************************************************/
void DMA1_source_addr(unsigned char *address, unsigned short mode)
{
    unsigned int range;
    unsigned short temp;

    DSA1L = ((int)address & 0x0000FFFF);
    DSA1H = (((int)address >> 16) & 0x000003FF);
    range = (unsigned int)address;
    /* check for internal or external, default is external */
    if(range >= 0x03ff0000 && range < 0x03ffefff)
        DSA1H |= 0x8000;  // internal RAM
    /* set count direction of transfer source address */
    temp = DADC1;
    temp &= 0xff3f;    /* clear source direction of transfer */
    temp |= (mode & 0x0003) << 6;
    temp |= 0x4000;    /* 16 bit transfer */
    DADC1 = temp;
}

/*********************************************************************/
/* Function:    DMA1_destination_addr                                */
/* Description: set DMA1 destination address                         */
/* Input:       address  - DMA1 destination address                  */
/*              mode     - increment, decrement or fixed address     */
/* Return:      none                                                 */
/*********************************************************************/
void DMA1_destination_addr(unsigned char *address, unsigned short mode)
{
    unsigned int range;
    unsigned short temp;

    DDA1L = ((int)address & 0x0000FFFF);
    DDA1H = (((int)address >> 16) & 0x000003FF);

    range = (unsigned int)address;
    /* check for internal or external, default is external */
    if(range >= 0x03ff0000 && range < 0x03ffefff)
        DDA1H |= 0x8000;  // internal RAM
    /* set count direction of transfer source address */
    temp = DADC1;
    temp &= 0xffcf;  // clear source direction of transfer
    temp |= (mode & 0x0003) << 4;
    temp |= 0x4000;    /* 16 bit transfer */
    DADC1 = temp;
}

/*********************************************************************/
/* Function:    DMA0_Setup_mmc                                       */
/* Description: set up DMA0 for CSIB5 input using 16 bit register    */
/* Input:       rxbuf  - pointer to buffer destination               */
/*              rxnum  - number of short words to read               */
/*              mode   - increment, decrement or fixed address       */
/*              factor - interrupt control                           */
/* Return:      none                                                 */
/*********************************************************************/
void DMA0_Setup_mmc(unsigned short *rxbuf, short rxnum, unsigned char factor)
{
    DMA0_Halt();

    DMA0_source_addr((unsigned char *)0xfffffd54, DMA_FIXED);   // &CB5RX

    DMA0_destination_addr((unsigned char *)rxbuf, DMA_INCR);

    DBC0 = rxnum-1;           /* stops when it goes negative */
    DTFR0 = (0x80 | factor);  /* 38 for INTCB5R  39 for INTCB5T */
}

/********************************************************************/
/* Function:    DMA1_Setup_mmc                                      */
/* Description: set up DMA1 for CSIB5 output using 16 bit register  */
/*              the transfer count stops when it goes negative      */
/* Input:       txbuf  - pointer to buffer source                   */
/*              txnum  - number of short words to send              */
/*              factor - interrupt control                          */
/* Return:      none                                                */
/********************************************************************/
void DMA1_Setup_mmc(unsigned short *txbuf, short txnum, unsigned char factor)
{
    DMA1_Halt();

    DMA1_source_addr((unsigned char *)txbuf, DMA_INCR);

    DMA1_destination_addr((unsigned char *)0xfffffd56, DMA_FIXED);  // &CB5TX

    DBC1 = txnum-1;
    DTFR1 = (0x80 | factor);    /* 38 for INTCB5R  39 for INTCB5T */
}
```

9.9 dma.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 : dma.h
**  Abstract : This file implements a device driver for the dma module
**  APIlib :   V850ESJx2.lib V1.50 [23 Feb. 2006]
**
**  Device :   uPD70F3721
**
**  Compiler : NEC/CA850
**
*******************************************************************************
*/
#ifndef _MDDMA_
#define _MDDMA_
/*
*******************************************************************************
**  MacroDefine
*******************************************************************************
*/
#include "macrodriver.h"


#define DMA_EXTERNAL  0x00000000  // external or peripheral I/O
#define DMA_INTERNAL  0x80000000  // internal RAM
#define DMA_ADDR_MASK 0x03FFFFFF  // to clear bits 14 to 10
                                  //
#define DMA_INCR      0
#define DMA_DECR      1
#define DMA_FIXED     2

void DMA0_Init( void );
MD_STATUS DMA0_Start( void );
void DMA0_Stop( void );
void DMA0_HALT( void );
void DMA1_Init( void );
MD_STATUS DMA1_Start( void );
void DMA1_Stop( void );
void DMA1_Halt( void );
__interrupt void MD_INTDMA0( void );
__interrupt void MD_INTDMA1( void );

void DMA0_source_addr(unsigned char *address, unsigned short mode);
void DMA0_destination_addr(unsigned char *address, unsigned short mode);
void DMA1_source_addr(unsigned char *address, unsigned short mode);
void DMA1_destination_addr(unsigned char *address, unsigned short mode);
void DMA0_Setup(unsigned short *rxbuf, short rxnum, unsigned char factor);
void DMA1_Setup(unsigned short *txbuf, short txnum, unsigned char factor);
void DMA0_dump(void);
void DMA1_dump(void);
#endif
```

FILE ID: int.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 : int.c
**  Abstract : This file implements a device driver for the INT module
**  APIlib :   V850ESJx2.lib V1.50 [23 Feb. 2006]
**
**  Device :   uPD70F3717
**
**  Compiler : NEC/CA850
**
*******************************************************************************
*/
/*
****************************************************************************
**  Include files
****************************************************************************
*/
#include "macrodriver.h"
#include "int.h"

/*
****************************************************************************
**  MacroDefine
****************************************************************************
*/


/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**     This function initializes INT.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-----------------------------------------------------------------------------
*/
void  INT_Init( void )
{
      SetIORBit(PIC0, 0x40);              /* stop external interrupt */
      ClrIORBit(PIC0, 0x80);
```

```
    ClrIORBit(PFC0, 0x08);                  /* set INTP0 pin */
    SetIORBit(PMC0, 0x08);
    ClrIORBit(INTR0, 0x08);
    SetIORBit(INTF0, 0x08);
    SetIORBit(PIC0, Lowest);
    ClrIORBit(PIC0, 0x40);                  /* enable INTP0 */
}
```

9.10 int_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 : int_user.c
**  Abstract : This file implements a device driver for the INT module
**  APIlib :   V850ESJx2.lib V1.50 [23 Feb. 2006]
**
**  Device :   uPD70F3717
**
**  Compiler : NEC/CA850
**
*********************************************************************************
*/
/*
******************************************************************************
**  Include files
******************************************************************************
*/
#include "macrodriver.h"
#include "int.h"

#pragma interrupt INTP0 MD_INTP0
/*
******************************************************************************
**  MacroDefine
******************************************************************************
*/

unsigned char mode = 1;

/*
**-------------------------------------------------------------------------------
**
**  Abstract:
**     This function is INTP0 Interrupt service routine.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------------
*/
```

```
__interrupt void MD_INTP0( void )
{
	/* TODO. Add user defined interrupt service routine */
	mode ^= 1;
	if(mode){
	PCM.2 = 1;
	PCM.3 = 1;
	}else
	{
	PCM.2 = 1;
	PCM.3 = 0;
	}

}
```

9.11 int.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 : int.h
**  Abstract : This file implements a device driver for the INT module
**  APIlib :   V850ESJx2.lib V1.50 [23 Feb. 2006]
**
**  Device :   uPD70F3717
**
**  Compiler : NEC/CA850
**
*******************************************************************************
*/
#ifndef   _MDINT_
#define   _MDINT_
/*
** ***************************************************************************
** MacroDefine
** ***************************************************************************
*/
#define   IC_BASE     0xfffff110  /*interrupt control register base address*/
enum ExternalINT { EX_NMI, EX_INTP0, EX_INTP1, EX_INTP2, EX_INTP3, EX_INTP4, EX_INTP5,
EX_INTP6,EX_INTP7 };
enum
MaskableSource{INT_LVI,INT_INTP0,INT_INTP1,INT_INTP2,INT_INTP3,INT_INTP4,INT_INTP5,INT_INTP
6,INT_INTP7,
          INT_TQ0OV,INT_TQ0CC0,INT_TQ0CC1,INT_TQ0CC2,INT_TQ0CC3,INT_TP0OV,
          INT_TP0CC0,INT_TP0CC1,INT_TP1OV,INT_TP1CC0,INT_TP1CC1,INT_TP2OV,
          INT_TP2CC0,INT_TP2CC1,INT_TP3OV,INT_TP3CC0,INT_TP3CC1,INT_TP4OV,
          INT_TP4CC0,INT_TP4CC1,INT_TP5OV,INT_TP5CC0,INT_TP5CC1,INT_TM0EQ0,
          INT_CB0R,INT_CB0T,INT_CB1R,INT_CB1T,INT_CB2R,INT_CB2T,INT_CB3R,
          INT_CB3T,INT_UA0R,INT_UA0T,INT_UA1R,INT_UA1T,INT_UA2R,INT_UA2T,
          INT_AD,INT_DMA0,INT_DMA1,INT_DMA2,INT_DMA3,INT_KR,INT_WTI,INT_WT,
          INT_ERR0,INT_WUP0,INT_REC0,INT_TRX0,
          INT_IIC1=33, INT_CB4R=41,INT_CB4T=42,INT_IIC2=43,INT_IIC0=45,INT_ERR=55,
          INT_STA=56,INT_IE1=57,INT_IE2=58};
void INT_Init( void );
__interrupt void MD_INTP0( void );
#endif
```

9.12 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 <stdlib.h>
#include "dma.h"
#include "macrodriver.h"
#include "int.h"
#include "port.h"
#include "timer.h"
#include "serial.h"
#include "sdmemory.h"

extern volatile int sample_ticks;

/* prototypes */
void dump_buffers(unsigned short *, unsigned short *, unsigned char *, unsigned char *);
void dump_byte(UCHAR *data, USHORT length);
void dump_word(unsigned short *data, int length);
int  get_sector(int *sector);
void fill_mem_rand(unsigned char *buffer, int size);
int  compare_buf(unsigned short *buf1, unsigned short *buf2, int size);
void flip_short(unsigned short *buffer, int size);
int  operator_data(unsigned char *buffer);
void shift_buffer(unsigned short *buffer);
void mem_to_mmc(int sector, USHORT *buffer);
void mem_to_mem(unsigned char *source,unsigned char *dest, unsigned short size);
void mmc_to_mem(int sector, unsigned short *buffer);
void ad_to_mem(int samples,unsigned short *buffer);
```

```
/*
** **************************************************************************
** MacroDefine
** **************************************************************************
*/
#define SECTOR   512
#define BUF_SIZE 520
#undef DEBUG_MD
// global data
char msg_buf[120];

/*
**---------------------------------------------------------------------------
**
**  Abstract:
**     main function
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**---------------------------------------------------------------------------
*/
void  main( void )
{
char msg_nl[] = {"\r\n"};
const char msg1[]     = {"\r\n\r\nDMA Demonstration program   (using interrupt I/O)"};
const char msg2[]     = {"\r\n   demonstrate read and write memory via DMA interface\r\n"};
const char msg3[]     = {"   1 - Internal Memory                 -> Peripheral-I/O (Memory
Card)\r\n"};
const char msg4[]     = {"   2 - Internal Memory                 -> External Memory (SRAM-
Module)\r\n"};
const char msg5[]     = {"   3 - External Memory                 -> External Memory\r\n"};
const char msg6[]     = {"   4 - Peripheral-I/O (Memory Card)   -> Internal Memory\r\n"};
//External Memory\r\n"};
const char msg7[]     = {"   5 - Peripheral-I/O (A/D converter) -> Internal Memory\r\n"};
const char msg8[]     = {"   6 - fill buffer with test data\r\n"};
const char msg9[]     = {"   7 - fill buffer with random data\r\n"};
const char msg10[]    = {"   8 - dump memory\r\n"};
const char msg11[]    = {"   enter number of operation to perform - "};
const char msg_sd1[]  = {"\r\n  enter sector number to use (00-99) "};
const char msg_a[]    = {"main - SD memory card init status 0x%02x\r\n"};

MD_STATUS status;
MD_STATUS mem_stat, mem_stat9, mem_stat10;
UCHAR data[20];
int err,i;
int line;
int sector;
int samples;
int process;
unsigned char *ext_mem  = (unsigned char *)0x00200000; /* base of external memory*/
unsigned char *ebuffer1 = (unsigned char *)0x00201000; /* external buffer 1 start address */
unsigned char *ebuffer2 = (unsigned char *)0x00202000; /* external buffer 2 start address */
USHORT done, SD_status;
unsigned short buffer1[BUF_SIZE/2];
unsigned short buffer2[BUF_SIZE/2];
unsigned char temp;  // dbg external memory
unsigned char *buf1;



    TMP0_Start();  /* start timer for delay counting */

    delay(250);     /* the settup 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);

    mem_stat = SDmemory_Init(); /* initialize SD memory access */
    if(mem_stat != MD_OK) {
        sprintf(msg_buf, msg_a ,mem_stat);  /* SD memory init status 0x%02x */
        uart3_tx_msg(msg_buf);
    }
    else
    {
        // check for done
        uart3_tx_msg("SDmemory_init done\r\n"); // rk-dbg

    }
    if(mem_stat == MD_OK)
    {
        delay(200);
#if DEBUG_SD
        mem_stat = SDReadStatus(&SD_status);
        if(mem_stat == MD_MASTER_RCV_END)
            mem_stat = MD_OK;

        sprintf(msg_buf,"read status after init 0x%02x card status
0x%04x\r\n",mem_stat,SD_status);
        uart3_tx_msg(msg_buf);
#endif
     }

    /* after initialization has completed, enter an endless loop to ask for the */
    /* process to be performed (1..6).  If process 1 or 4 is selceted, ask for  */
    /* sector number to read or write, then wait for data to be entered or for  */
    /* one of the switches to be pressed                                        */
 while (1) {
        /* output message to ask for process number */

        uart3_tx_msg((char *)msg2);
        uart3_tx_msg((char *)msg3);
        uart3_tx_msg((char *)msg4);
        uart3_tx_msg((char *)msg5);
        uart3_tx_msg((char *)msg6);
        uart3_tx_msg((char *)msg7);
        uart3_tx_msg((char *)msg8);
        uart3_tx_msg((char *)msg9);
        uart3_tx_msg((char *)msg10);
        uart3_tx_msg((char *)msg11);     /* could make this into a for loop */

        UART3_User_Init();    /* reset buffer pointers */
        process = 0;          /* default to invalid process selection */
        /* read the operator process number input, use the get_sector function */
        /*     if error, just request input again */
        err = get_sector(&process);
        if(err || (process > 8)) continue;
        uart3_tx_msg(msg_nl);

        if(process == 1 || process == 4)
        {
            /* output message to enter the sector number */
            uart3_tx_msg((char *)msg_sd1);
            /* 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_nl);

        }

        // do requested process
        switch (process)
        {
        case 1:  // Internal Memory                -> Peripheral-I/O (Memory Card)
            mem_to_mmc(sector,buffer1);
            mmc_to_mem(sector,buffer2); // test sd card read, transfer to internal memory
            shift_buffer(buffer1);      // shift up, don't compare start data token
            err = compare_buf(buffer1, buffer2, 256);
            if(err) {
                err--;  // because 0 is not an error
                sprintf(msg_buf,"\r\ndifference found at offset 0x%04x\r\nsend
buffer1\r\n",err);
                uart3_tx_msg(msg_buf);
                sprintf(msg_buf,"expected 0x%04x received
0x%04x\r\n",buffer1[err],buffer2[err]);
                uart3_tx_msg(msg_buf);
                dump_word(buffer1, 256);
                uart3_tx_msg("\r\nreceive buffer2\r\n");
                dump_word(buffer2, 256);

            } else {
                uart3_tx_msg("\r\n *** buffer data equal ***\r\n");
            }
            break;

        case 2:  // Internal Memory                -> External Memory (SRAM-Module)
            /* test - write to external memory */
            /* next step is to dma it */
            for(i=0; i< 16; i++) // dbg
            {
                ext_mem[i] = 0x55;
                temp = ext_mem[i]; /* test one location */
                sprintf(msg_buf,"read 0x%02x\r\n",temp);
                uart3_tx_msg(msg_buf);
            }

            //mem_to_mem((unsigned char *)buffer1,ebuffer2,BUF_SIZE);
            break;

        case 3:  // External Memory                -> External Memory
```

```
            mem_to_mem(ebuffer2,ebuffer1,BUF_SIZE);
            break;

        case 4:  // Peripheral-I/O (Memory Card)   -> External Memory
//            mmc_to_mem(sector,(unsigned short *)ebuffer1);
 mmc_to_mem(sector,buffer2); // test sd card read, transfer to internal memory
 dump_byte((unsigned char *)buffer2,512);
            break;

        case 5:  // Peripheral-I/O (A/D converter) -> Internal Memory
            samples = 256;  /* these are 16 byte samples */
            for(i=0;i<BUF_SIZE;i++)
                buffer2[i]=i;   // fill the buffer with some other pattern
            uart3_tx_msg("\r\ntakes 10 seconds\r\n");
            ad_to_mem(samples,buffer2);
            dump_word(buffer2,samples);  // show the data
            uart3_tx_msg(msg_nl);
            break;

        case 6:  // fill a buffer with test data
            operator_data((unsigned char *)buffer1);

            uart3_tx_msg("buffer1\r\n");                    // rk-dbg
            dump_byte((unsigned char *)buffer1,514);      // rk-dbg
            dump_word(buffer1,514/2);                     // rk-dbg
            flip_short(buffer1,514/2);
            dump_word(buffer1,514/2);                     // rk-dbg
            break;

        case 7:  // fill a buffer with start token, random data, dummy checksum
            fill_mem_rand((unsigned char *)buffer1,SECTOR);
            uart3_tx_msg("buffer1\r\n");  // rk-dbg
            flip_short(buffer1,516/2);
            dump_word(buffer1,516/2);     // rk-dbg
            uart3_tx_msg(msg_nl);         // rk-dbg
            break;

        case 8:  // dump requested memory
            dump_buffers(buffer1, buffer2, ebuffer1, ebuffer2);
            break;

        default:
            continue;
        }
 }
}

/*********************************************************************/
/* Function:    operator_data()                                      */
/* Description: enter data into indicated buffer                     */
/* Input:       *data   - pointer to start dumping data from         */
/*              length  - number of bytes of data to display         */
/* Return:      size    - number of bytes entered                    */
/*********************************************************************/
int operator_data(unsigned char *buffer)
{
    int done = 0;
    int size = 0;
    int line = 0;
    UCHAR data[20];
    char msg_nl[] = {"\r\n"};

    uart3_tx_msg("enter data\r\n");
    data[1] = 0;
    buffer[0]= (unsigned char)START_BLOCK;  /* transmit data always starts with START_BLOCK
*/
    size++;
    while(done == 0)
    {
     /* check for any receive characters, display the character entered */
     if( Check_UART3_Receive(&data[0]) == 1)
     {
         buffer[size] = data[0];
         if(data[0] == 0x08)
         {
             size--;
             /* move cursor back one, print a space and move back one again */
             line--;
         }
         else
         {
             size++;
             line++;
         }
         if(data[0] == '\r')
         {
             buffer[size] = '\n';
             size++;
             uart3_tx_msg(msg_nl);
             line = 0;
         }
            /* check if to many characters entered, exit when full  */
            /* allow for start token and crc bytes                  */
         if(size > BUF_SIZE-3)
            {
                uart3_tx_msg("\r\ndone\r\n");
                return (size);
            }
         uart3_tx_msg((char *)data);
         if(line >= 80)
         {
             line = 0;
             uart3_tx_msg(msg_nl);
         }
     }
     } // while(done)
     return(size);
}

/**********************************************************************/
/* 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);
}

/********************************************************************/
/* Function:    dump_word()                                         */
/* Description: dump memory in word form to serial port             */
/* Input:       *data   - pointer to start dumping data from        */
/*              length  - number of bytes of data to display        */
/* Return:      none                                                */
/********************************************************************/
void dump_word(unsigned short *data, int length)
{
    int i,j;
    char buff[8];


    for(i=0,j=0; i<length; i++, j=i%8)
    {
         if(j == 0)
         {
             uart3_tx_msg("\r\n");
         }

         sprintf(buff,"%04x ", *data);
         uart3_tx_msg(buff);
         data++;
    }
```

```
}

/*********************************************************************/
/* Function:    dump_buffers()                                       */
/* Description: dump memory in byte form, also show ascii equivalent */
/*              dump to serial port. buffers are all 514 bytes long  */
/* Input:       *buffer1  - pointer to internal data buffer one      */
/*              *buffer2  - pointer to internal data buffer two      */
/*              *ebuffer1 - pointer to external data buffer one      */
/*              *ebuffer2 - pointer to external data buffer two      */
/* Return:      none                                                 */
/*********************************************************************/
void dump_buffers(unsigned short *buffer1,
                  unsigned short *buffer2,
            unsigned char *ebuffer1,
            unsigned char *ebuffer2)
{
    uart3_tx_msg("buffer1\r\n");
 dump_byte((unsigned char *)buffer1,514);  // mmc write data

 uart3_tx_msg("buffer2\r\n");
 dump_byte((unsigned char *)buffer2,514);  // A->D sample data
                                              // mmc read data
 uart3_tx_msg("ebuffer1\r\n");
 //dump_byte(ebuffer1,514);

 uart3_tx_msg("ebuffer2\r\n");
 //dump_byte(ebuffer2,514);
}

/**********************************************************************/
/* 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:    fill_mem_rand()                                       */
/* Description: fill memory buffer with pseudo random data pattern    */
/* Input:       buffer - pointer to where data is to be written       */
/*              size   - number of characters to write                */
/* Return:      none                                                  */
/**********************************************************************/
void fill_mem_rand(unsigned char *buffer, int size)
{
    int i;
    unsigned int seed;

 uart3_tx_msg("random data\r\n");
 buffer[0] = START_BLOCK;          // sd mem start data token

    /* get a seed from timer free run counter */
    seed = TP0CNT;
    /* set the seed */
    srand(seed);

    for(i=1; i<=size; i++)
        buffer[i] = rand(); /* get a random number*/

 buffer[size+1] = 0xff;       // dummy checksum
 buffer[size+2] = 0xff;       // fix up the buffer
    buffer[size+3] = 0xff;       // for short words
}

int  compare_buf(unsigned short *buf1, unsigned short *buf2, int size)
{
    int i;

    for(i=0; i<size; i++)
    {
        if(buf1[i] != buf2[i])
            return(i+1);
    }
    return(0);
}
/* shift buffer up one byte (little endian chicken dance) */
void shift_buffer(unsigned short *buf)
{
    int i;

    for(i=0; i<256; i++)
    {
       buf[i] = (buf[i] << 8) & 0xff00;
       buf[i] = buf[i] | ((buf[i+1] >> 8) & 0x00ff);
    }
}
```

```
/*********************************************************************/
/* Function:    mem_to_mmc()                                         */
/* Description: transfer one sector from internal memory buffer to    */
/*              specified sector of SD/MMC using DMA                  */
/*              (after write initialization has been performed)       */
/* Input:       sector - sector to write to                           */
/*              buffer - pointer to data to be written                */
/* Return:      none                                                  */
/*********************************************************************/
void mem_to_mmc(int sector, USHORT *source)
{
    MD_STATUS status;

    /* stop both dma channels */
    DMA0_Stop();
    DMA1_Stop();

    /* do mmc initialization for write - select sector to be written */
    /* then write it                                                 */
    status = SDWriteSector(source, sector);
    if(status != MD_MASTER_SEND_END)
    {
        uart3_tx_msg("SDWriteSector error \r\n");
    }

    DMA0_Stop();
    DMA1_Stop();
}

/*********************************************************************/
/* Function:    mem_to_mem()                                          */
/* Description: transfer from one memory buffer to another buffer     */
/* Input:       source - pointer to source of data                    */
/*              dest   - pointer to where data is to be written       */
/*              size   - number of bytes to transfer                  */
/* Return:      none                                                  */
/*********************************************************************/
void mem_to_mem(unsigned char *source, unsigned char *dest, unsigned short size)
{
    MD_STATUS status;
    int done  = 0;
    unsigned char save;

    /* set up dma */
    DMA0_Stop();
    DBC0 = size;
    /* set source to internal memory incrementing */
    DMA0_source_addr(source, DMA_INCR);

    /* set destination to external memory incrementing */
    DMA0_destination_addr(dest, DMA_INCR);

    /* set addressing control, 8 bit transfer, increment increment */
    DADC0 = 0x0000;
    /* clear dma trigger factor */
    DTFR0 = 0x80;

    /* enable the DMA, sets the Enn enable bit in DCHC0 */
    status = DMA0_Start();

    /* set channel control to start the transfer */
    SetIORBit(DCHC0, 0x2);    // set the STGn bit to start the transfer

    /* check for completion */
    while( done == 0)
    {
        save = DCHC0;
        // reading DCHC0 clears TC0 bit
        if((save & 0x01) == 0)
        {
            if ((save & 0x80) == 0x80)
                done = 1;
        }
    }
}

/**********************************************************************/
/* Function:    emem_to_emem()                                        */
/* Description: transfer one sector from external memory buffer to    */
/*              specified external memory buffer                      */
/* Input:       source - pointer to source of data                    */
/*              dest   - pointer to where data is to be written       */
/*              size   - number of bytes to transfer                  */
/* Return:      none                                                  */
/**********************************************************************/

/* these functions are the same, have DMA0_source and destination figure out
if address is internal or external */
#if 0  // deprecate
void emem_to_emem(unsigned char *source, unsigned char *dest, unsigned short size)
{

    MD_STATUS status;
    int done  = 0;
    unsigned char save;

    /* set up dma channel*/
    DMA0_Stop();
    DMA1_Stop();

    DBC0 = size;

    /* set source to external memory incrementing */
    DMA0_source_addr(source, DMA_EXTERNAL);

    /* set destination to external memory incrementing */
    DMA0_destination_addr(dest, DMA_EXTERNAL);

    /* set addressing control, 8 bit transfer, increment increment */
    DADC0 = 0x0000;
    /* clear dma trigger factor */
    DTFR0 = 0x80;
    /* enable the DMA, sets the Enn enable bit in DCHC0 */
    status = DMA0_Start();

    /* set channel control to start the transfer */
    SetIORBit(DCHC0, 0x2);    // set the STGn bit to start the transfer

    /* check for completion */
    while( done == 0)
    {

        save = DCHC0;
        // reading DCHC0 clears TC0 bit
        if((save & 0x01) == 0)
        {
            if ((save & 0x80) == 0x80)
                done = 1;
        }
    }
}
#endif

/**********************************************************************/
/* Function:    mmc_to_mem()                                          */
/* Description: transfer one sector from SD/MMC to external memory    */
/*              buffer using CSIB5 RX register in interrupt mode.     */
/* Input:       sector - sector to read from                          */
/*              dest   - pointer to where data is to be written       */
/* Return:      none                                                  */
/**********************************************************************/
void mmc_to_mem(int sector, unsigned short *dest)
{
    MD_STATUS status;

    /* set up dma channel*/
    DMA0_Stop();
    DMA1_Stop();

    /* do mmc initialization - select sector to be read from,  */
    /* read the sector using dma transfer and CSIB5 interrupts */
    status = SDReadSector(dest, sector);
    if(status != MD_MASTER_RCV_END)
    {
        sprintf(msg_buf,"SDReadSector error 0x%02x\r\n",status);
        uart3_tx_msg(msg_buf);
    }

    DMA0_Stop();
    DMA1_Stop();
}

/**********************************************************************/
/* Function:    ad_to_mem()                                           */
/* Description: transfer n samples from analog to digital converter   */
/*              to internal memory buffer at timed interval.          */
/*              The potentiometer POT1 is sampled on P71/ANI1         */
/*              The timer starts each conversion, the AtoD completion */
/*              interrupt signals the DMA to store the result.        */
/* Input:       samples - number of samples to perform                */
/*              dest    - pointer to where data is to be written      */
/* Return:      none, errors reported locally                         */
/**********************************************************************/
void ad_to_mem(int samples, unsigned short *dest)
```

```
{
    MD_STATUS status;
    int done = 0;
    unsigned char save;
    char msg_buf[40];

    /* set up A to D convertor */
    ADA0M0 = 0x02;  /* stop any conversion */
    ADA0M1 = 0x06;  /* 20.7us conversion time */
    ADA0M2 = 0x01;  /* use timer trigger interrupt INTTP2CC0 */
                    /* to start timed conversions            */
    /* select input channel to sample */
    ADA0S =  0x01;  /* select channel 1, pin P71  */

    DMA0_Stop();
    DMA1_Stop();

    /* set number of samples to take */
    DBC0 = samples;

    /* set dma source to single address ADA0CR1, interrupt driven */
    DMA0_source_addr((unsigned char *)0xFFFFF212, DMA_FIXED);

    /* set dma destination to internal memory incrementing */
    DMA0_destination_addr((unsigned char *)dest, DMA_INCR);

    /* clear dma trigger factor, select trigger factor INTAD */
    DTFR0 = 0x80 + 0x2C;

    /* start the DMA */
    AD_Start();            /* enable the interrupts */
    status = DMA0_Start(); /* ok to start after dma has started */
                        /* as we have not enabled the trigger yet */
    if(status == MD_ERROR) uart3_tx_msg("DMA0 start error\r\n");

    /* start timer 2 (enable the timer interrupt) */
    TMP2_Start();

    /* check for completion with no errors */
    while(done == 0)
    {
        save = DCHC0;
        // reading DCHC0 clears TC0 bit
        if((save & 0x01) == 0)
        {
            if ((save & 0x80) == 0x80)
                done = 1;
        }
        if(sample_ticks > 260) {
            done = 1;
            uart3_tx_msg("timeout\r\n");
            sprintf(msg_buf,"status %d\r\n",status);
            uart3_tx_msg(msg_buf);
            sprintf(msg_buf,"DBC0 %04x DCHC0 %02x\r\n",DBC0,save);
            uart3_tx_msg(msg_buf);

        }
    }
    AD_Stop();      /* stop A to D interrupts */
    TMP2_Stop();    /* stop timer 2 interrupts */
    DMA0_Stop();
}

/* flip little endian bytes */
void flip_short(unsigned short *buffer, int size)
{
    int i;
    unsigned char *cbuf;
    unsigned char temp;

    cbuf = (unsigned char *)buffer;
    for (i=0; i<size*2; i=i+2) {
        temp = cbuf[i+1];
        cbuf[i+1] = cbuf[i];
        cbuf[i] = temp;
    }

}

void DMA0_dump(void)
{
    sprintf(msg_buf,"rx DDA0 0x%04x%04x\r\n",DDA0H,DDA0L);/* destination address count 
increment */
    uart3_tx_msg(msg_buf);
    sprintf(msg_buf,"   DSA0 0x%04x%04x  DBC0 0x%04x\r\n",DSA0H,DSA0L,DBC0);/* source address 
*/
    uart3_tx_msg(msg_buf);
    sprintf(msg_buf,"   DCHC0 0x%02x  DTFR0 0x%02x  DMAIC0 0x%02x\r\n",DCHC0,DTFR0,DMAIC0);
    uart3_tx_msg(msg_buf);
}

void DMA1_dump(void)
{
    sprintf(msg_buf,"tx DDA1 0x%04x%04x\r\n",DDA1H,DDA1L);/* destination address count 
increment */
    uart3_tx_msg(msg_buf);
    sprintf(msg_buf,"   DSA1 0x%04x%04x  DBC1 0x%04x  DADC1 
0x%04x\r\n",DSA1H,DSA1L,DBC1,DADC1);/* source address */
    uart3_tx_msg(msg_buf);
    sprintf(msg_buf,"   DCHC1 0x%02x  DTFR1 0x%02x  DMAIC1 0x%02x\r\n",DCHC1,DTFR1,DMAIC1);
    uart3_tx_msg(msg_buf);
}

void CSIB5_dump(void)
{
    sprintf(msg_buf,"CB5CTL 0-0x%02x 1-0x%02x  2-0x%02x\r\n",CB5CTL0,CB5CTL1,CB5CTL2);
    uart3_tx_msg(msg_buf);
    sprintf(msg_buf,"CB5RX %04x CB5TX %04x CB5STR %02x CB5RIC %02x 
CB5TIC %02x\r\n",CB5RX,CB5TX,CB5STR,CB5RIC,CB5TIC);
    uart3_tx_msg(msg_buf);
    sprintf(msg_buf,"PMC6   %04x\r\n",PMC6);
    uart3_tx_msg(msg_buf);
}
```

9.13 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;     // used by external memory
        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;
        PMCCT &= ~PORT_PMCCT;
        PMCDH &= ~PORT_PMCDH;
        PMCDL &= ~PORT_PMCDL;
        return;
}
```

9.14 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
```

9.15 system_user.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 : system_user.c
**  Abstract : This file implements a device driver for the SYSTEM interrupt service
routine
**  APIlib: v850esJx2.lib V1.50 [23 Feb. 2006]
**
**  Device:  uPD70F3717
**
**  Compiler:  NEC/CA850
**
*****************************************************************************
*/
/*
** ************************************************************************
** Include files
** ************************************************************************
*/
#include "macrodriver.h"
/*
** ************************************************************************
** MacroDefine
** ************************************************************************
*/
```

9.16 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    0x00
#define    PORT_PM7H   0xff
#define    PORT_P7H    0x00

#define    PORT_PMC8   0x00     // UART3
#define    PORT_PM8    0xff
#define    PORT_PF8    0x00
#define    PORT_P8     0x00

#define    PORT_PMC9   0xffff   // address lines A0-15
#define    PORT_PM9    0x0000   // all are output
#define    PORT_P9     0x0000   // all are output
#define    PORT_PFC9   0x0000   // alt function is address output
#define PORT_PFCE9  0x0000   // alt function is address output
#define PORT_PF9    0xffff   // normal open drain output

#define    PORT_PMCD   0xff  // address
#define    PORT_PCD    0x00
#define    PORT_PMCM   0xff
#define    PORT_PCM    0x00
#define    PORT_PMCCM  0x00  // no alt function

#define    PORT_PMCS   0xff
#define    PORT_PCS    0x00
#define    PORT_PMCCS  0x00

#define    PORT_PMCT   0xff
#define    PORT_PCT    0x00
#define    PORT_PMCCT  0x00
/* port DH - 8 bit port */
#define    PORT_PMDH   0x00   // all output
#define    PORT_PDH    0x00   // all output
#define    PORT_PMCDH  0xff   /* alternate function A16-23 */
/* port DL - 16 bit port */
#define    PORT_PMDL   0xffff  // all input (overriden by PMCDL)
#define    PORT_PDL    0x0000  // all
#define    PORT_PMCDL  0x00ff  // ADn I/O 0-7

void PORT_Init(void);
#endif
```

9.17 sdmemory.c

```
/* sd memory.c */
#include <stdlib.h>
#include "macrodriver.h"
#include "sdmemory.h"
#include "serial.h"
#include "timer.h"

MD_STATUS err_val(UCHAR response);
void dump_byte(UCHAR *data, USHORT length);
void dump_word(USHORT *data, USHORT length);
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];
};

extern MD_STATUS csib5_snd_flag;  /* dbg - serial.c */
extern UINT csib5_snd_count;      /* dbg - serial.c */
extern UINT csib5_rcv_count;      /* dbg - serial.c */
extern USHORT csib5_snd_size;     /* 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 sucessfuly*/
/*              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);

    /* SD/MMC initialization */
    /* 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, send number of bytes expected */
        R1_Initiate();

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

        /* send pad, build CMD0 to go into SPI mode and send it */
        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 */
#if DEBUG_SD
    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);
        /* look for the response */
        R1_message = SDmemory_R_query(R1,20);
        CSIB5_deselect_SPI();   /* deselect all spi devices */

        if(R1_message == 0x00)  /* ready, no longer in idle */
        {
#if DEBUG_SD
            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         */
/**********************************************************************/
#if 0
MD_STATUS SDmemory_CMD_R16(char index, UCHAR *data)
{
    MD_STATUS status;
    UCHAR R1_message;
    UCHAR DT_message;
    int i;

    R1_Initiate();  /* set up to receive an R1 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

    R1_message = SDmemory_R_query(R1,100);
    if(R1_message)
    {
        #if DEBUG_SD
        sprintf(buffer,"R_query %d 0x%02x error\r\n",index,R1_message);
        uart3_tx_msg(buffer);
        #endif

        CSIB5_deselect_SPI();
        return(err_val(R1_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)
    {
        #if DEBUG_SD
        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);
}
#endif


/**********************************************************************/
/* 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         */
/*              CSIB5_rcv_done = 0     clear interrupt flag           */
/* 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 qurery 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 dr_txbuf[2] = {0xff,0xff};
    MD_STATUS status;

    retry = 0;  // dbg - count number of times before we get response

    while(Byte == 0xff)
    {
        retry++; // dbg
        /* get ready to read a one byte response message */
        R1_Initiate();
        status = CSIB5_SendData(dr_txbuf, 1, &Byte);
        if(status != MD_OK) {
            uart3_tx_msg("SDmemory_DR_query error\r\n");
            return(status);
        }
        i--;
        if(i == 0) return(0x80);
//        sprintf(buffer,"q 0x%02x tx 0x%02x\r\n",Byte,dr_txbuf[0]);
//        uart3_tx_msg(buffer);  // rk-dbg
        delay(50); // rk-dbg - to see scope data
    }

    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   */
/*                        (must be word (16) aligned)                 */
/*              Sector  - sector number to read                       */
/* Return:      MD_STATUS */
/*********************************************************************/
MD_STATUS SDReadSector(USHORT *pBuffer, int Sector)
{
    MD_STATUS status;
    UCHAR DT_message,odd_byte;
    int i, block_address;
    unsigned short *bs;

    block_address =  Sector << 9;
    #if DEBUG_SD
    sprintf(buffer,"SDReadSector from addr 0x%08x (sector %d) into buffer 0x%08x\r\n",
            block_address, Sector, pBuffer);
    uart3_tx_msg(buffer);
    #endif
    R1_Initiate();  /* set up to receive an R1 response */

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

    build_cmd(17, block_address); /* send command as to which sector will be read */

    R1_message = SDmemory_R_query(R1,400);
    if(R1_message)
    {
        #if DEBUG_SD
        sprintf(buffer,"query 17 0x%02x error (retry %d) \r\n",R1_message,retry);
        uart3_tx_msg(buffer);
        #endif
        CSIB5_deselect_SPI();
        return(err_val(R1_message));
    }
    for(i=0; i<517; i++)
        buffer[i] = 0xff;  // small delay  (is this needed & could it be put into query?)

    DT_message = SDmemory_DT_query(200); /* locate the start data token */
    if(DT_message == 0x80)
    {
        #if DEBUG_SD
        uart3_tx_msg("DT query timeout\r\n");
        #endif
        CSIB5_deselect_SPI();
        return(MD_REQ_TIMEOUT);
    }
    /* token received, now we can send the data */
#if 1
    CSIB5_polled_mode(OFF);  // switch to 16 bit mode

    status = CSIB5_SendDataBlock(bs, 518);

    status = CSIB5_ReceiveDataBlock((USHORT *)pBuffer, 516); //515);
    /* now send 516 dummy bytes to clock in one sector of data + CRC16 */
    bs =(unsigned short *)&buffer[0];  /* point to first word in buffer */
                              /* address convert will round down */

    delay(20);
//    sprintf(buffer,"after sector write & read count %d size %d
done %d\r\n",csib5_snd_count, csib5_snd_size,CSIB5_send_done);
//    uart3_tx_msg(buffer); // rk-dbg
    // this is checking for the dma transfer complete
    // the delay above was to allow CSIB5 to finish transfer
    if(check_send_done(400) != 0)
    {
        CSIB5_polled_mode(ON);  // set mode back to 8 bit polled
        CSIB5_deselect_SPI();
        return(MD_REQ_TIMEOUT);
    }
#else
/* doing it 8 bit polled */
    status = CSIB5_SendData((unsigned char *)buffer, 516, (unsigned char *)pBuffer);
#endif
    CSIB5_polled_mode(ON);

    /* 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 - successsful write                */
/*              MD_REQ_TIMEOUT - */
/*********************************************************************/
MD_STATUS SDWriteSector(USHORT *pBuffer, int Sector)
{
    MD_STATUS status;
    UCHAR DR_message;
    int i, block_address;

    block_address =  Sector << 9;

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

    R1_Initiate();  /* set up to receive an R1 response */
    build_cmd(24, block_address);

    R1_message = SDmemory_R_query(R1,NCR);
    if(R1_message)
    {
        #if DEBUG_SD
        sprintf(buffer,"R1 query cmd24 0x%02x error\r\n",R1_message);
        uart3_tx_msg(buffer);
        dump_byte((UCHAR *)R1_message,1); // rk-dbg - error value
        #endif
        CSIB5_deselect_SPI();
        return(err_val(R1_message));
    }

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

  // dbg write in dma mode if enabled else polled mode
    CSIB5_polled_mode(OFF);  // set mode to 16 bit
                             // is the data in the right order ????
    /* initialize for receipt of start token + 512 bytes + 2 bytes CRC, always returns good
status */
    status = CSIB5_ReceiveDataBlock((USHORT *)buffer, 514);

    /* now send 515 data bytes (token, data, crc) */
    /* if you send to much, you will miss the reply */
    status = CSIB5_SendDataBlock(pBuffer, 514);
    if(status != MD_OK)
        uart3_tx_msg("sector write error\r\n"); // rk-dbg
    if(check_send_done(400) != 0)
    {
        CSIB5_polled_mode(ON);  // set mode back to 8 bit polled
        CSIB5_deselect_SPI();
        uart3_tx_msg("timeout\r\n"); // rk-dbg
        return(MD_REQ_TIMEOUT);
    }
    delay(5);
//    sprintf(buffer,"tx interrupt count %d\r\n",csib5_snd_count);  // rk-dbg
//    uart3_tx_msg(buffer);  // rk-dbg
    CSIB5_polled_mode(ON);  // set mode back to 8 bit polled
//uart3_tx_msg("sector write DR check\r\n"); // rk-dbg
    delay(5);
#if 0
    /* do it 8 bit polled */
    status = CSIB5_SendData((unsigned char *)pBuffer, 516, (unsigned char *)buffer);
#endif
    /* check for data response token */
    DR_message = SDmemory_DR_query(20);
#if DEBUG_SD
    sprintf(buffer,"DR_message 0x%02x\r\n",DR_message); // dbg
    uart3_tx_msg(buffer); // dbg
#endif
    if(DR_message == MD_NO_START)
    {
        #if DEBUG_SD
        uart3_tx_msg("DR query no start\r\n");
        #endif
        CSIB5_deselect_SPI();
        return(MD_NO_START);
    }
    if(DR_message == 0x80)
    {
        #if DEBUG_SD
        uart3_tx_msg("DR query timeout\r\n");
        #endif
        CSIB5_deselect_SPI();
        return(MD_REQ_TIMEOUT);
    }
    #ifdef DEBUG_SD
    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 rk-dbg */

    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_SD
        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();

    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 */
    c.cmd_arg = arg;
    /* build the command in cmd_buf (little endian order)  */
    /* include NCS=2 beginning pad characters */
    cmd_buf[0] = 0xff;  // pad byte
    cmd_buf[1] = 0xff;  // pad byte
    cmd_buf[2] = 0x40 | (index & 0x3f);
    cmd_buf[3] = c.cmd_ch[0];
    cmd_buf[4] = c.cmd_ch[1];
    cmd_buf[5] = c.cmd_ch[2];
    cmd_buf[6] = c.cmd_ch[3];
    cmd_buf[7] = do_crc7(&cmd_buf[2],5);
    cmd_buf[8] = 0xff;
    CB5STR = 0; // clear overflow
    CSIB5_SendData(cmd_buf, 9, rxbuf);  /* send the command with leading pad */

    while(CSIB5_send_done  == 0){
    }  // wait for send to complete
    //dump_word(&b.cs[0],5); // rk-dbg the command being sent
    //sprintf(buffer,"send cmnd done rcv_done %d\r\n",CSIB5_rcv_done);
    //uart3_tx_msg(buffer); // rk-dbg
    //dump_word(rxbuf,5); // rk-dbg
}

/*********************************************************************/
/* 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:    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;
    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));

    /* 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);
}

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

```
/* 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%c|     Product Revison    %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 messate            */
/*********************************************************************/
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");
}

/*********************************************************************/
/* Function:    check_send_done                                      */
/* Description: check that CSIB5 send done flag gets set.  Check for */
/*              a specified period of time.  If it does not get set  */
/*              return timeout error.                                */
/* Input:       period - maximum time to check for completion        */
/* Return:      MD_OK */
/*              MD_REQ_TIMEOUT                                       */
/*********************************************************************/
int check_send_done(int period)
{
    //check for done, but will only wait so long and then abort
    SetMsecTimer(period);
    while(CSIB5_send_done == 0)
    {
        if(CheckMsecTimer() == MD_TRUE)
        {   //sprintf(buffer,"timeout - snd %d rcv %d\r\n",csib5_snd_count,csib5_rcv_count);
            //uart3_tx_msg(buffer);
            DMA0_dump();
            DMA1_dump(); // rk-dbg
            return(MD_REQ_TIMEOUT);
        }
    }
    return(0);
}

#if 0
/***********************************************************************/
int check_rcv_done(int period)
{
//check for done, but should only wait so long and then abort
    SetMsecTimer(period);
    while(CSIB5_rcv_done == 0)
    {
        if(CheckMsecTimer() == MD_TRUE)
        {   sprintf(buffer,"snd %d rcv %d 
data %2x\r\n",csib5_snd_count,csib5_rcv_count,CB5TXL);
            uart3_tx_msg(buffer);
            // DMA0_dump();  // rk-dbg
            // CSIB5_dump();  // rk-dbg
            return(MD_REQ_TIMEOUT);
        }
    }
    return(0);
}
#endif
```

9.18 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"

#define DEBUG_SD  1

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  /* these have been rounded up because of dma */
#define NAC          2
#define NEC          3
#define NCX          1
#define NWR          3

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

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

MD_STATUS SDmemory_Write(USHORT *buffer, USHORT size);
MD_STATUS SDmemory_Read(USHORT *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);
MD_STATUS SDmemory_CMD32(unsigned int data_addr);   // erase start address
MD_STATUS SDmemory_CMD33(unsigned int data_addr);   // erase end address
MD_STATUS SDmemory_CMD38(unsigned int stuff_bits);  // erase
MD_STATUS SDmemory_CMD58(OCR_REG *reg);

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 CSIB5_DR_query(short max_retry);
unsigned char CSIB5_DT_query(short max_retry, unsigned char *data);
void build_cmd(char index, unsigned int arg);
unsigned char  do_crc7(unsigned char *data, unsigned short size);
unsigned short do_crc16(unsigned char *data, unsigned short size);
char *err_text(UCHAR response);

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

9.19 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 "dma.h"
#include "macrodriver.h"
#include "serial.h"

#pragma interrupt INTUA3R MD_INTUA3R
#pragma interrupt INTUA3T MD_INTUA3T
#pragma interrupt INTCB5R MD_INTCB5R
USHORT uart3_snd_count;
volatile UCHAR *uart3_snd_pbuf;
USHORT uart3_rcv_count;
UCHAR  *uart3_rcv_pbuf;
volatile UCHAR send3_done;

UINT    xfr_size = 1;
USHORT csib5_snd_size;
UINTcsib5_snd_count;
UCHAR  *csib5_snd_pbuf;
MD_STATUS csib5_snd_flag;

MD_STATUS csib5_rcv_flag;
USHORT csib5_rcv_size;
UCHAR  *csib5_rcv_pbuf;
UINTcsib5_rcv_count;

/*-------------------------------------------------------------------------
**  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 fuclk=fxx/8  fxx=20Mhz */
 UA3CTL2 = UART3_BAUDRATE_K0;    /* 0x82 fuclk/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 = 0x04;               /* type 1, fxx/32 =  625KHz */
    CB5CTL1 = 0x05;                 /* type 1, fxx/64 =  312.5KHz*/
 CB5CTL2 = 0x00;                 /* data length - 8bit (only switch to 16 for dma)*/
//    CB5CTL2 = 0x08;                 /* data length - 16 bits (because of dma) */
//SetIORBit(CB5RIC,  0x05);       /* reception interrupt priority setting level 5 */
// ClrIORBit(CB5RIC,  0x40);   /* enable interrupt servicing */
 SetIORBit(CB5CTL0, 0x40);     /* enable send operation */
 SetIORBit(CB5CTL0, 0x20);     /* enable receive operation */
    SetIORBit(CB5CTL0, 0x81);       /* enable operation, communication start trigger valid */

//    SetIORBit(CB5TIC,  0x05);       /* transmit interrupt priority setting level 5 */
//    ClrIORBit(CB5TIC,  0x40);       /* enable transmit interrupt servicing */

    // no user init function to perform

 return;
}

/*-------------------------------------------------------------------------
** Abstract:
**  This function receives data when mode is Master to destination for CSIB5.
**
** Parameters:
**  UCHAR* rxbuf : Address of receive buffer.
**  USHORT rxnum : The number of bytes that should be received.
**
** Returns:
**  MD_OK
**  MD_ARGERROR
```

```
**-------------------------------------------------------------------------*/
MD_STATUS CSIB5_ReceiveDataBlock(USHORT* rxbuf, USHORT rxnum)
{
 volatile USHORT   dummy;
    MD_STATUS status;

 /* init receive parameters */
 csib5_rcv_count = 0;
    CSIB5_rcv_done = 0;
 csib5_rcv_size = rxnum;
 csib5_rcv_pbuf = (unsigned char *) rxbuf;
    /* init transmit parameters*/
    //csib5_snd_count = 0; CSIB5_send_done = 0;
    csib5_snd_size = rxnum;

    /* set up DMA channel 0 to receive */
    /* would normally use trigger factor 0x38 but since we only get one interrupt */
    /* use it for the transmit side */
    DMA0_Setup_mmc(rxbuf, (rxnum+1)/2, 0); // destination is internal RAM , round count up
    status = DMA0_Start();                    // enable
 if(status != MD_OK) return status;
    CB5RIC = 0x05;                  // enable receive interrupt trigger
 return MD_OK;
}


/*-------------------------------------------------------------------------
** Abstract:
**  This function is responsible for transfer of data out CSIB5.
**  It is polled mode, ie no interrupts are being used
**  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;

 /* 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 = CB5RXL;  // read receive register byte and save
         csib5_rcv_pbuf++;
     }
     else
     {
         return(MD_NO_START);
     }
    }
    CSIB5_send_done = 1;
    return(MD_OK);
}

/*-------------------------------------------------------------------------
** Abstract:
**  This function is responsible for initiating transfer of data out CSIB5
**  using DMA channel 1.  Since every byte sent is a byte received, DMA
**  channel 0 should previously beens set up to receive an equal number
**  of data bytes.
**
** Parameters:
**  USHORT* txbuf :   Address of transmit buffer.
**  USHORT  txnum :   The number of data bytes to transmit.
**
** Returns:
**  MD_OK
**  MD_NO_START
**-------------------------------------------------------------------------*/
MD_STATUS CSIB5_SendDataBlock(USHORT* txbuf, USHORT txnum)
{
    char cb5_status;
    MD_STATUS status;

    /* set up DMA channel 1 to transmit */
    DMA1_Stop();  // disable DMA channel 1  - redundant code

 /* init transmit parameters (do not care about receive) */
    csib5_snd_count = 0;
    CSIB5_send_done = 0;
 csib5_snd_size = txnum;
    DMA1_Setup_mmc(txbuf,((txnum+1)/2),0x38);  // round count up since we always send 2 bytes
                                               // using the receive interrupt trigger factor
    status = DMA1_Start();                     // enable DMA1 controller
 if(status != MD_OK) return status;
    CB5STR = 0x00;   // clear overflow
    //CB5TIC = 0x06;    // turn on CSIB5 transmit interrupt, priority 6 (there is no transmit
interrupt)

    SetIORBit(DCHC1,0x02);       // kick start the transmit operation
 return(MD_OK);
}
```

```
/*********************************************************************/
/* Function:    CSIB5_polled_mode                                    */

/* Description: switch CSIB5 port from 8 bit length for polled to    */
/*              16 bit length for dma operation                      */
/* Input:       mode    - ON  for 8 bit polled mode                  */
/*                        OFF for 16 bit dma mode                    */
/* Return:      none                                                 */
/*********************************************************************/
void CSIB5_polled_mode(char mode)
{
        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 */
    if(mode == ON)
    {
//uart3_tx_msg("ON-8 bit mode\r\n"); //rk-dbg
        xfr_size = 1;

        /* switching from 16 bit mode to 8 bit polled mode */
        CB5CTL2 = 0x00;                 /* data length - 8bit */
        SetIORBit(CB5CTL0, 0xE1);       /* enable send and receive operation, communication
start trigger valid */
    } else
    {
//uart3_tx_msg("OFF-16 bit mode\r\n"); //rk-dbg
        xfr_size = 2;

        /* switching from 8 bit mode to 16 bit interrupt dma mode */
        /* interrupts on */

        CB5CTL2 = 0x08;                 /* data length - 16 bits (because of dma) */
     SetIORBit(CB5RIC,  0x05);       /* reception interrupt priority setting level 5 */
        ClrIORBit(CB5RIC,  0x40);    /* enable receive interrupt servicing */
        SetIORBit(CB5CTL0, 0xE1);       /* enable operation, communication start trigger
valid */

        SetIORBit(CB5TIC,  0x05);       /* transmit interrupt priority setting level 5 */
        ClrIORBit(CB5TIC,  0x40);       /* enable transmit interrupt servicing */
    }
 CB5STR = 0;         // clear overflow
}

/*-------------------------------------------------------------------------
** Abstract:
**  This function is called at interrupt request when data received.
**
**  Parameters: None.
**
** Returns:
**  MD_OK
**  MD_ERROR
**
**  NOTE:
**  If over run error happened, return MD_ERROR.
**  User should be programing when all data received.
**---------------------------------------------------------------------------*/
MD_STATUS CSIB5_Data_Handler( void )
{
char cb5_status;


 cb5_status = CB5STR;
 CB5STR = 0;

 /* check over run error */
 if( CB5STR & 1 ) {
    CALL_CSIB5_Error();
    return MD_ERROR;
 }

 /* single transfer mode, 16 bit data length */

 csib5_snd_count += xfr_size;
 if( csib5_snd_count >= csib5_snd_size ) /* send finished? */
 {
    CALL_CSIB5_Send();    /* yes, call user function */
 }

    SetIORBit(DCHC0,0x02);       // time to receive ???
 csib5_rcv_count += xfr_size;
 if( csib5_rcv_count >= csib5_rcv_size ) /* receive finish */
 {
    CALL_CSIB5_Receive();   /* yes, call user function */
 }

 return MD_OK;
}

/*---------------------------------------------------------------------------
** Abstract:
**  This function is the receive interrupt handler for INTCB5R.
**
** Parameters:  None
**
** Returns:  None
**---------------------------------------------------------------------------*/
__multi_interrupt void MD_INTCB5R( void )

{
 __EI();
 csib5_rcv_flag = CSIB5_Data_Handler();
}

/*******************************************************************/
__interrupt void MD_INTCB5T( void )

{
 __EI();
#if 0  // there is no transmit interrupt in single transfer mode
 csib5_snd_count += xfr_size;

 if( csib5_snd_count >= csib5_snd_size ) /* send finished? */
```

```
  {
     CALL_CSIB5_Send();    /* yes, call user function */
  }

   SetIORBit(DCHC1,0x02);       // kick start the transmit operation again
#endif
}
```

9.20 serial_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 : serial_user.c
**  Abstract : This file gives callback functions for serial module.
**  APIlib :   V850ESJx2.lib V1.50 [23 Feb. 2006]
**
**  Device :   uPD70F3721
**
**  Compiler : NEC/CA850
**
*****************************************************************************/

/******************************************************************************
**  Include files
*******************************************************************************/
#include "macrodriver.h"
#include "serial.h"

volatile int CSIB5_rcv_done;
volatile int CSIB5_send_done;

#define RX_BUF_SIZE 64

UCHAR rx3_rr_buf[RX_BUF_SIZE];  // round robin buffer for uart3 receive data
volatile int rr3_buf_put, rr3_buf_get;
extern volatile UCHAR send3_done;

volatile int R1_received;
volatile int R2_received;
volatile int R3_received;
volatile char  R1_message;
volatile short R2_message;
volatile int   R3_message;

/*-----------------------------------------------------------------------------
**  Abstract:
**    This function is an empty function for user code when UART3 initializing
**
**  Parameters:    None
**
**  Returns:   None
**----------------------------------------------------------------------------*/
void UART3_User_Init( void )
{
    rr3_buf_put = rr3_buf_get = 0;
}

/*-----------------------------------------------------------------------------
**  Abstract:
**     This function puts the receive data into the round robin buffer.
**
**  Parameters:    data - data received by the uart3
**
**  Returns:   None
**----------------------------------------------------------------------------*/
void UART3_Receive( UCHAR data)
{
    rx3_rr_buf[rr3_buf_put] = data;

    rr3_buf_put++;
    if(rr3_buf_put >= RX_BUF_SIZE)
        rr3_buf_put = 0;

}

/*-----------------------------------------------------------------------------
**  Abstract:
**     This function gets the receive data from round robin buffer.
**
**  Parameters:    *data - pointer to store received data
**
**  Returns:   0 = no data available
**              1 = a byte of data returned
**----------------------------------------------------------------------------*/
char Check_UART3_Receive( UCHAR *data)
{
    if(rr3_buf_put == rr3_buf_get)
        return (0);
    *data = rx3_rr_buf[rr3_buf_get];
    rr3_buf_get++;
    if(rr3_buf_get >= RX_BUF_SIZE)
        rr3_buf_get = 0;
    return (1);
}

/*-----------------------------------------------------------------------------
**  Abstract:
**     This function outputs a null terminated string to uart3 tx port.
**
**  Parameters:    *msg - pointer to string
**
**  Returns:   none
**----------------------------------------------------------------------------*/
void uart3_tx_msg( char *msg)
{
USHORT txnum;

//    txnum = strlen(msg);    cannot step through this
    txnum = 0;
    while(msg[txnum] != 0)  /* find end of string */
    {
        txnum++;
    }
    /* start the interrupt driven transmit process */
    /* this clears send3_done */
    UART3_SendData((UCHAR *)msg, txnum);

    /* wait for it to complete */
      while (send3_done == 0)
          {};  // just hang until it is done sending
}

/*-----------------------------------------------------------------------------
**  Abstract:
**    This function inputs a string from uart3 tx port.  The input ends when a
**  carriage return (0x0d) or line feed (0x0a) or null (0x00) is received.
**
**  Parameters:    *msg - pointer buffer to put string in
**                 max - size of msg buffer, max characters allowed
**
**  Returns:   number of characters received and placed in buffer
**----------------------------------------------------------------------------*/
#if 0
/* this function is not needed as the interrupt service routine puts the */
/* received data in the round robin buffer */
short uart3_rx_msg(UCHAR *msg, int max)
{
short rxnum;
UCHAR *point = msg;
UCHAR done = 0;

    rxnum = 0;
    while(done == 0)  /* done set at end of string */
    {
        /* start the interrupt driven receive process to get 1 character */
        /* this clears rcv3_done */
        UART3_ReceiveData(point, 1);

        /* wait for it to complete */
          while (send3_done == 0)
              {};  // just hang until it is done receiving
        if(*point == '\r' || *point == '\n' || *point == '\0')
            done = 1;
        if(rxnum++ == max)
            done = 1;
        point++;
    }
    return(rxnum);
}
#endif

/*-----------------------------------------------------------------------------
**  Abstract:
**    This function is a call back function to deal with data process after
**    some frame(s) data transfering of CSIB5 interface.
**  Called by interrupt service routine when send count equals send size.
**
**  Parameters:    None
**
**  Returns:   None
```

```
**---------------------------------------------------------------------------*/
void CALL_CSIB5_Send( void )
{
     CSIB5_send_done = 1;
}

/*----------------------------------------------------------------------------
**  Abstract:
**    This function is a call back function to deal with data process after
**    some frame(s) data receiving of CSIB5 interface.
**  Called by interrupt service routine when receive count equals expected
**  receive size.
**
**  Parameters:    None
**
**  Returns:   None
**---------------------------------------------------------------------------*/
void CALL_CSIB5_Receive( void )
{
    CSIB5_rcv_done = 1;
}

/*------------------------------------------------------------------------
**  Abstract:
**    This function is a call back function to deal with data process after
**    some frame(s) data receiving error of CSIB5 interface.
**
**  Parameters:    None
**
**  Returns:   None
**---------------------------------------------------------------------------*/
void CALL_CSIB5_Error( void )
{
    R1_received = -1;
    R2_received = -1;
    R3_received = -1;
    CSIB5_rcv_done = -1;
}

/*------------------------------------------------------------------------
**  Abstract:
**    This function is a called to select the specified CSIB5 SPI device
**  by clearing the chip select port bit.
**
**  Parameters:    device - device to select
**                       SDMEM1 - SD Memory
**
**  Returns:   None
**---------------------------------------------------------------------------*/
void CSIB5_select_SPI(int device)
{
    /* if there are other chip selects, make sure they are all high */
    CSIB5_deselect_SPI();
    switch (device) {
    case SDMEM1:
        /* assert chip select (low) for SD memory 1 on P35 */

        P3 = P3 & ~SPI_CS5;  /* clear bit 5 */
        break;
    case SDMEM1T:
        /* assert chip select low and trigger scope */
        P3 = P3 & ~(SPI_CS5+SPI_CS4);
        break;
    }
}

/*-------------------------------------------------------------------------
**  Abstract:
**      This function is a called to deselect all CSIB5 SPI chip selects
**
**  Parameters:    None
**
**  Returns:   None
**----------------------------------------------------------------------------*/
void CSIB5_deselect_SPI(void)
{
    /* deassert chip select for CSIB5 SPI */
    /* make sure they are all high */
    P3 |= SPI_CS4;   /* deselect zigbee    */
    P3 |= SPI_CS5;   /* deselect sd memory */
}
```

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

#define IIC_RECEIVEBUFSIZE      32

#define UART3_BAUDRATE_M0   0x03
#define UART3_BAUDRATE_K0   0x82

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

#define ON   1
#define OFF  0


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* txbuf, USHORT txnum, UCHAR* rxbuf);
MD_STATUS CSIB5_SendDataBlock( USHORT* , USHORT);
MD_STATUS CSIB5_ReceiveDataBlock( USHORT* , 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_*/
```

9.22 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"
/*
*******************************************************************************
**  Variables
*******************************************************************************
*/
volatile unsigned int sample_ticks;

/*
**---------------------------------------------------------------------------
**
**  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
**
**---------------------------------------------------------------------------
** it is not used by this application */
```

```
#if 0
void TMP0_Stop( void )
{
        ClrIORBit(TP0CTL0,0x80);    /* stop counting */
        /* Mask interrupt */
        SetIORBit(TP0CCIC0,0x40);
        /* Clear interrupt request flag */
        ClrIORBit(TP0CCIC0,0x80);
        return;
}
#endif
/*
**---------------------------------------------------------------------------
**
**  Abstract:
**        This function changes TMP0 condition.
**
**  Parameters:
**        USHORT*:    array_reg
**        USHORT: array_num
**
**  Returns:
**        MD_OK
**        MD_ARGERROR
**
**---------------------------------------------------------------------------
** it is not used by this application */
#if 0
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
/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**        This function initializes TMP2.
**
**  Parameters:
**        None
**
**  Returns:
**        None
**
**-----------------------------------------------------------------------------
*/
void TMP2_Init( void )
{
        /* Stop counting */
        ClrIORBit(TP2CTL0, 0x80);
        /* Mask interrupt */
        SetIORBit(TP2CCIC0, 0x40);
        SetIORBit(TP2CCIC1, 0x40);
        SetIORBit(TP2OVIC, 0x40);
        /* Clear interrupt request flag */
        ClrIORBit(TP2CCIC0, 0x80);
        ClrIORBit(TP2CCIC1, 0x80);
        ClrIORBit(TP2OVIC, 0x80);

        ClrIORBit(TP2CTL1, 0x20);   /* disable external event count input */
        TP2CTL0 |= TM_TMP2_CLOCK;   /* internal count clock */
        /* Interval timer mode */
        ClrIORBit(TP2CTL1, 0x07);
        TP2CCR0 = TM_TMP2_INTERVALVALUE;
        TP2CCR1 = 0xffff;
        /* Interrupt INTTP2CC0 */
        SetIORBit(TP2CCIC0, 0x07);
}

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**    This function starts TMP2 counter.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-----------------------------------------------------------------------------
*/
void TMP2_Start( void )
{
        ClrIORBit(TP2CCIC0,0x40);   /* enable interrupt INTTP2CC0 */
        SetIORBit(TP2CTL0,0x80);    /* start counting */
        sample_ticks = 0;           /* count the number of interrupts for debug */
        return;
}

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**    This function stops the TMP2 counter and clear the count register.
**
**  Parameters:
**    None
**
**  Returns:
**    None
**
**-----------------------------------------------------------------------------
*/
void TMP2_Stop( void )
```

```
{
        ClrIORBit(TP2CTL0,0x80);    /* stop counting */
        /* Mask interrupt */
        SetIORBit(TP2CCIC0,0x40);
        /* Clear interrupt request flag */
        ClrIORBit(TP2CCIC0,0x80);
        return;
}

/*
**-----------------------------------------------------------------------------
**
**  Abstract:
**     This function changes TMP2 condition.
**
**  Parameters:
**     USHORT*:    array_reg
**     USHORT:     array_num
**
**  Returns:
**     MD_OK
**     MD_ARGERROR
**
**-----------------------------------------------------------------------------
** not used by this application */
#if 0
MD_STATUS TMP2_ChangeTimerCondition( USHORT* array_reg, USHORT array_num )
{
        if((array_num < 1) || (array_num > 2)){
            return MD_ARGERROR;
        }
        if( array_num >= 1 ){
            TP2CCR0 = *array_reg;
        }
        if( array_num >= 2){
            TP2CCR1 = *(array_reg + 1);
        }
        return MD_OK;
}
#endif
```

9.23 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"

#pragma interrupt INTTP0CC0 MD_INTTP0CC0
#pragma interrupt INTTP2CC0 MD_INTTP2CC0

/**************************************************************************
**  variables
***************************************************************************/
/* counter for millisecond timer */
volatile unsigned int milliseconds;
extern volatile unsigned int sample_ticks;
/*-------------------------------------------------------------------------
**
**  Abstract:
**     TMP0 initializing.
**
**  Parameters:
**     None
**
**  Returns:
**     None
**
**-------------------------------------------------------------------------*/
void TMP0_User_Init( void )
{
    milliseconds = 0;
}
```

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

      /* count down millisecond timer if it is non zero */
      if (milliseconds > 0)
          milliseconds--;
}

/*--------------------------------------------------------------------------------
**  Abstract:
**     This function is TMP2 INTTP2CC0 interrupt service routine.
**
**  Parameters:    None
**
**  Returns:   None
**
**--------------------------------------------------------------------------------
*/
__interrupt void MD_INTTP2CC0( void )
{
      /* TODO. Add user defined interrupt service routine */
      //uart3_tx_msg("T2");
      sample_ticks++;
      __EI();
}

/***********************************************************************/
/* 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
}
```

9.24 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   0x6
#define TM_TMP2_INTERVALVALUE   0x2423    //0xc34
#define TM_TMP2_INTERVALVALUE2  0x3d08
#define TM_TMP2_ONESHOTOUTPUTCYCLE  0xc34
#define TM_TMP2_ONESHOTOUTPUTDELAY  0x3d08
#define TM_TMP2_EXTTRIGGERCYCLE 0xc34
#define TM_TMP2_EXTTRIGGERDELAY 0x3d08
#define TM_TMP2_PWMCYCLE    0xc34
#define TM_TMP2_PWMWIDTH    0x3d08
#define TM_TMP2_CCR0COMPARE 0xc34
#define TM_TMP2_CCR1COMPARE 0x3d08
#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 TMP2_Init( void );
void delay(int count);
void TMP2_Start( void );
void TMP2_Stop( void );
MD_STATUS TMP2_ChangeTimerCondition(USHORT* array_reg,USHORT array_num);
__interrupt void MD_INTTP2CC0( void );

#endif
```

10. Appendix C — Development Tools

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

10.1 Software Tools

Applilet code generation tool: applilet_v850es_jx2_v150.exe

Compiler, assembler and linker: part of a package called CA850 Compiler

Project Manager PM+: the integrated development environment (IDE)

10.2 Hardware Tools

PC Windows 2000 or Windows XP

Demo Board AF-EV850 Basic Rev 1.0

MiniCube2 with USB interface

1 Appendix D — Reference 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 MuliMediaCard
   - Product Manual Version 1.3 Document No. 80-36-00320 April 2005
9. MultiMediaCard Specification Ver 0.9 June 2004
   - Samsung Electronics., LTD
10. uPD4440008L MOS Integrated Circuit Data Sheet
    - 4M-Bit CMOS Fast SRAM 512K-Word by 8-Bit
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
13. 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 transferring a program to be embedded in a microcontroller’s on-chip flash memory. The MiniCube2 connects to the development PC via USB.

The MiniCube2 uses a piece of monitor code that is loaded with the development code. To accomodate 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 monittor 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 mointor 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 the memory layout to accomodate 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 for this application.

12.1 Port Assignment

P71	ANI1	input	analog input

P80	RXDA3	input	UART3 receive
P81	TXDA3	out	UART3 transmit

P90	A0	out
P91	A1	out
P92	A2	out
P93	A3	out
P94	A4	out
P95	A5	out
P96	A6	out
P97	A7	out
P98	A8	out
P99	A9	out
P910	A10	out
P911	A11	out
P912	A12	out
P913	A13	out
P914	A14	out
P915	A15	out

PDH0	A16	out
PDH1	A17	out
PDH2	A18	out

PDL0	AD0	I/O
PDL1	AD1	I/O
PDL2	AD2	I/O
PDL3	AD3	I/O
PDL4	AD4	I/O
PDL5	AD5	I/O
PDL6	AD6	I/O
PDL7	AD7	I/O

(8bit bus so do not specify AD8-AD15)

PCS1 /CS1 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
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		

12.2 uPD444008L_list

The pin connections for the external SRAM (4M-bit - 512K x 8 bit 8ns) are as follows.

pin		CN1	function	
1	A0	1	A0/TXD1/KR6/P90	
2	A1	3	A1/RXD1/KR7/P91	
3	A2	5	A2/TO02/TI020/P92	
4	A3	7	A3/TI021/P93	
5	A4	9	A4/TO03/TI030/P94	
14	A5	11	A5/TI031/P95	
15	A6	13	A6/TO51/TI51/P96	
16	A7	15	A7/SI01/P97	
17	A8	17	A8/SO01/P98	
18	A9	19	A9/SCK01/P99	
20	A10	21	A10/SIA1/P910	
21	A11	23	A11/SOA1/P911	
22	A12	25	A12/SCKA1/P912	
23	A13	27	A13/INTP4/P913	
24	A14	29	A14/INTP5/P914	
32	A15	31	A15/INTP6/P915	
33	A16	65	A16/PDH0	
34	A17	67	A17/PDH1	
35	A18	69	A18/PDH2	
7	I/O1	33	AD0/PDL0	
8	I/O2	35	AD1/PDL1	
11	I/O3	37	AD2/PDL2	
12	I/O4	39	AD3/PDL3	
25	I/O5	41	AD4/PDL4	
26	I/O6	43	AD5/PDL5/FLMD1	(MiniCube)
29	I/O7	45	AD6/PDL6	
30	I/O8	47	AD7/PDL7	
6	/CS	16	/CS1/PCS1	0x00200000-0x0028FFFF address range
13	/WE	42	/WR0/PCT0	
31	/OE	50	/RD/PCT4	
85	/Wait	22	/Wait/PCM0	(MiniCube)
9,27	VCC	60	BVDD	
10,28	GND	2,4,58		
19,36	NC			

2nd adapter board header -> pin	
1 - n/c	2 - 35
3 - 34	4 - 33
5 - 32	6 - 31
7 - 30	8 - 29
9 - 28	10 -27
11-26	12 - 25
13 - 24	14 - 23
15 - 22	16 - 21
17 - 20	18 - n/c
19 - n/c	20 - n/c