User's Manual

32

RX62G Group

Renesas Starter Kit Software Manual

RENESAS MCU
RX Family / RX600 Series

Renesas Electronics
www.renesas.com

Rev.1.00 Sep 2012

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   ¾ The input pins of CMOS products are generally in the high-impedance state. In operation with unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   ¾ The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   ¾ The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   ¾ When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to one with a different part number, confirm that the change will not lead to problems.

   ¾ The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.

How to Use This Manual

1.Purpose and Target Readers

This manual is designed to provide the user with an understanding of the RSK hardware functionality, and electrical characteristics. It is intended for users designing sample code on the RSK platform, using the many different incorporated peripheral devices.

The manual comprises of an overview of the capabilities of the RSK product, but does not intend to be a guide to embedded programming or hardware design. Further details regarding setting up the RSK and development environment can found in the tutorial manual.

Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section, and in the Usage Notes section.

The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.

The following documents apply to the RX62G Group. Make sure to refer to the latest versions of these documents. The newest versions of the documents listed may be obtained from the Renesas Electronics Web site.

Document Type	Description	Document Title	Document No.
User's Manual	Describes the technical details of the RSK hardware.	RSKRX62G User's Manual	R20UT2223EG
Software Manual	Describes the functionality of the sample code, and its interaction with the Renesas Peripheral Driver Library (RPDL)	RSKRX62G Software Manual	R20UT2246EG
Tutorial	Provides a guide to setting up RSK environment, running sample code and debugging programs.	RSKRX62G Tutorial Manual	R20UT2224EG
Quick Start Guide	Provides simple instructions to setup the RSK and run the first sample, on a single A4 sheet.	RSKRX62G Quick Start Guide	R20UT2225EG
Schematics	Full detail circuit schematics of the RSK.	RSKRX62G Schematics	R20UT2222EG
Hardware Manual	Provides technical details of the RX62G microcontroller.	RSKRX62G Hardware Manual	R01UH0321EJ

2. List of Abbreviations and Acronyms

Abbreviation	Full Form
ADC	Analog to Digital Converter
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DTC	Data Transfer Controller
HEW	High-performance Embedded Workshop
IIC	Inter-IC Connect
IWDT	Independent Watchdog Timer
LIN	Local Inter-connect Network
LVD	Low Voltage Detection
MCU	Microcontroller Unit
PC	Personal Computer
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
RSK	Renesas Starter Kit
SCI	Serial Communication Interface
SPI	Serial Peripheral Interface
WDT	Watch Dog Timer

Table of Contents

1. Overview 8
1.1 Purpose 8

2. RSK Sample Code Concept 9
2.1 Sample Code Structure 9
2.2 List of Sample Code 10

3. Tutorial Samples 11
3.1 Tutorial 11
3.1.1 Description 11
3.1.2 Operation 12
3.1.3 Sequence Diagram 12
3.1.4 RPDL Integration 13
3.2 Application 13
3.2.1 Description 13

4. Peripheral Samples 14
4.1 ADC10_OneShot 14
4.1.1 Description 14
4.1.2 Operation 14
4.1.3 Sequence Diagram 14
4.1.4 RPDL Integration 15
4.2 ADC10_Repeat 15
4.2.1 Description 15
4.2.2 Operation 15
4.2.3 Sequence Diagram 16
4.2.4 RPDL Integration 16
4.3 ADC12_OneShot 17
4.3.1 Description 17
4.3.2 Operation 17
4.3.3 Sequence Diagram 17
4.3.4 RPDL Integration 18
4.4 ADC12_Repeat 18
4.4.1 Description 18
4.4.2 Operation 18
4.4.3 Sequence Diagram 19
4.4.4 RPDL Integration 19
4.5 SPI 20
4.5.1 Description 20
4.5.2 Operation 20
4.5.3 Sequence Diagram 20
4.5.4 RPDL Integration 21
4.6 LIN Master 21
4.6.1 Description 21
4.6.2 Operation 21
4.6.3 Sequence Diagram 22
4.6.4 RPDL Integration 22
4.7 Async_Serial 23
4.7.1 Description 23
4.7.2 Operation 23
4.7.3 Sequence Diagram 24
4.7.4 RPDL Integration 24

4.8 Sync_Serial 25
4.8.1 Description 25
4.8.2 Operation 25
4.8.3 Sequence Diagram 25
4.8.4 RPDL Integration 26
4.9 Power_Down 26
4.9.1 Description 26
4.9.2 Operation 26
4.9.3 Sequence Diagram 27
4.9.4 RPDL Integration 27
4.10 LVD 28
4.10.1 Description 28
4.10.2 Operation 28
4.10.3 Sequence Diagram 28
4.10.4 RPDL Integration 29
4.11 IIC_Master 29
4.11.1 Description 29
4.11.2 Operation 29
4.11.3 Sequence Diagram 30
4.11.4 RPDL Integration 30
4.12 IIC_Slave 31
4.12.1 Description 31
4.12.2 IIC Slave Commands 31
4.12.3 Operation 31
4.12.4 Sequence Diagram 32
4.12.5 RPDL Integration 33
4.13 CRC 34
4.13.1 Description 34
4.13.2 Operation 34
4.13.3 Sequence Diagram 34
4.13.4 RPDL Integration 35
4.14 Timer_Capture 35
4.14.1 Description 35
4.14.2 Operation 35
4.14.3 Sequence Diagram 36
4.14.4 RPDL Integration 36
4.15 Timer Compare 37
4.15.1 Description 37
4.15.2 Operation 37
4.15.3 Sequence Diagram 37
4.15.4 RPDL Integration 37
4.16 Timer Event 38
4.16.1 Description 38
4.16.2 Operation 38
4.16.3 Sequence Diagram 38
4.16.4 RPDL Integration 38
4.17 Timer Mode 39
4.17.1 Description 39
4.17.2 Operation 39
4.17.3 Sequence Diagram 39
4.17.4 RPDL Integration 39
4.18 Flash_Data 41
4.18.1 Description 41
4.18.2 Operation 41
4.18.3 Sequence Diagram 42

4.18.4 RPDL Integration 42
4.19 DTC 43
4.19.1 Description 43
4.19.2 Operation 43
4.19.3 Sequence Diagram 43
4.19.4 RPDL Integration 44
4.20 PWM 45
4.20.1 Description 45
4.20.2 Operation 45
4.20.3 Sequence Diagram 45
4.20.4 RPDL Integration 45
4.21 WDT 46
4.21.1 Description 46
4.21.2 Operation 46
4.21.3 Sequence Diagram 47
4.21.4 RPDL Integration 47
4.22 IWDT 48
4.22.1 Description 48
4.22.2 Operation 48
4.22.3 Sequence Diagram 48
4.22.4 RPDL Integration 49

5. Additional Information 50

1. Overview

1.1 Purpose

This RSK is an evaluation tool for Renesas microcontrollers. This manual explains the operation of the sample code provided, and its interaction with the Renesas Peripheral Driver Library (RPDL). The Renesas Peripheral Driver Library (hereinafter “this library” or RPDL) is based upon a unified API (Application Programming Interface) for the microcontrollers made by Renesas Electronics Corporation.

This manual is not intended to be a tutorial on using RPDL, or how RPDL works – it simply aims to explain to the reader how the RPDL was used to create the sample code supplied with the RSK. For further information regarding RPDL, visit the PDG (Peripheral Driver Group) section of the Renesas website:

http://www.renesas.com/pdg

Note:

This document explains by text and diagrams the functionality of the sample code and its interaction with the Renesas Peripheral Driver Library (RPDL). The RPDL is preliminary and functionally tested to operate with the RSKRX62G sample code only.

2. RSK Sample Code Concept

2.1 Sample Code Structure

The basic structure of all RSK sample code is shown in Figure 2-1 below. The first two functions, 'PowerOn_ResetPC' and 'HardwareSetup', configure the MCU before the main program code executes.

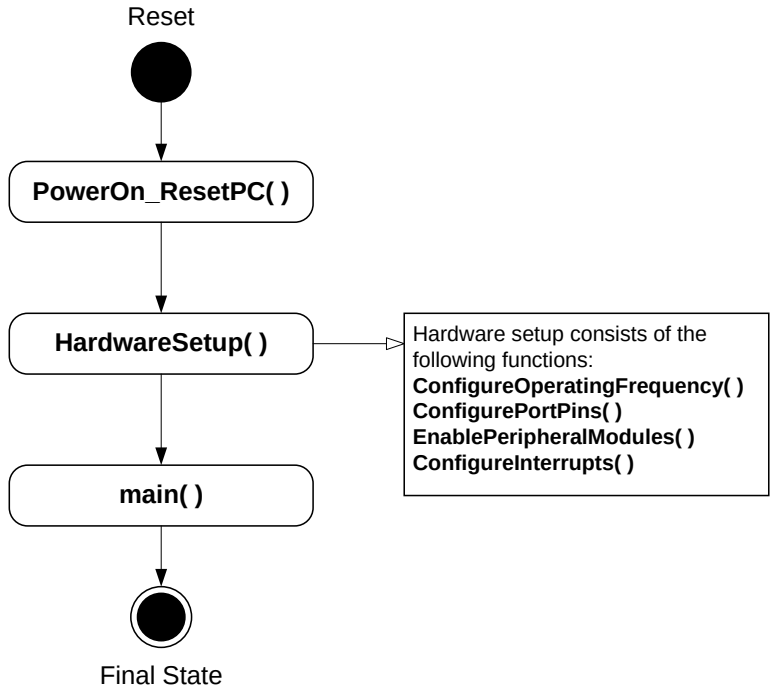


Figure 2-1: Sample Code Structure

The purpose of the functions included in the 'HardwareSetup' function are detailed in Table 2-1 below.

Function Name	Purpose	RPDL Functions Used
ConfigureOperatingFrequency	Initialises the main MCU, bus and peripheral clocks; as well as any real-time clocks and PLL settings.	R_CGC_Set
ConfigurePortPins	Configures the MCU port pins as inputs or outputs, depending on the devices on the RSK and the intended function of the sample code. Also sets some pins to suitable initial logic levels.	R_IO_PORT_Set R_IO_PORT_Write

Table 2-1: Hardware Setup Functions

* RPDL functions can not be used to manually enable/disable MCU peripherals, as this is controlled with the Create/Destroy functions for each RPDL group; therefore RPDL functions are not required in this section.
** The R_INTC_CreateExtInterrupt RPDL function is indirectly called by the function ConfigureInterrupts.

2.2 List of Sample Code

Table 2-2 below lists the sample code supplied with the RSKRX62G, and describes their function.

Sample Code	Description
ADC10_Oneshot	Demonstrates usage of the 10bit ADC module, in one shot mode.
ADC10_Repeat	Demonstrates usage of the 10bit ADC module, in repeat mode.
ADC12_Oneshot	Demonstrates usage of the 12bit ADC module, in one shot mode.
ADC12_Repeat	Demonstrates usage of the 12bit ADC module, in repeat mode.
Application	Blank project, used for development. Includes device initialisation code.
Async_Serial	Demonstrates usage of the SCI module, in asynchronous mode.
CRC	Demonstrates usage of the CRC module, by creating checksums of keyboard inputs.
CAN	Demonstrate the CAN Bus API and it's available functionality.
DTC	Demonstrates usage of the DTC module, by performing interrupt requested transfers.
IIC_Master	Demonstrates usage of the IIC module in master mode.
IIC_Slave	Demonstrates usage of the IIC module acting as a slave device.
IWDT	Demonstrates usage of the IWDT by allowing the timer to underflow.
LIN_Master	Demonstrates usage of the LIN module in master mode.
LVD	Demonstrates usage of the LVD circuit to generate interrupts on low voltage detections.
Power_Down	Demonstrates usage of the MCU power modes, by entering standby.
PWM	Demonstrates usage of the GPT by outputting a PWM signal on a dedicated I/O pin
SPI	Demonstrates usage of the SPI by performing transfers in a loopback mode.
Sync_Serial	Demonstrates usage of the SCI module, in synchronous mode.
Timer_Capture	Demonstrates usage of the CMT module, in external capture mode.
Timer_Compare	Demonstrates usage of the CMT module, in compare-match timer mode.
Timer_Event	Demonstrates usage of the MTU3 module by using an external clock.
Timer_Mode	Demonstrates usage of the MTU3 module by outputting a 1KHz square wave.
Flash_Data	Demonstrates usage of the FCU module, by writing to data flash memory.
Tutorial	Demonstrates basic usage of the debugger, and RSK hardware.
Watchdog	Demonstrates usage of the watchdog timer, by causing a WDT overflow interrupt.

Table 2-2: Sample Code List

3. Tutorial Samples

3.1 Tutorial

The sample code in this section is basic tutorial code, used to demonstrate basic usage of the RSK and help the user to begin writing their own basic sample code.

3.1.1 Description

The tutorial sample code demonstrates basic usage of the debugger and RSK hardware, and is common to all RSKs. This sample is supplied programmed onto the MCU, and executes out of the box when power is applied.

The sample calls three main functions to demonstrate port pin control, interrupt usage and C variable initialisation. These functions are shown in Figure 3-1 below.

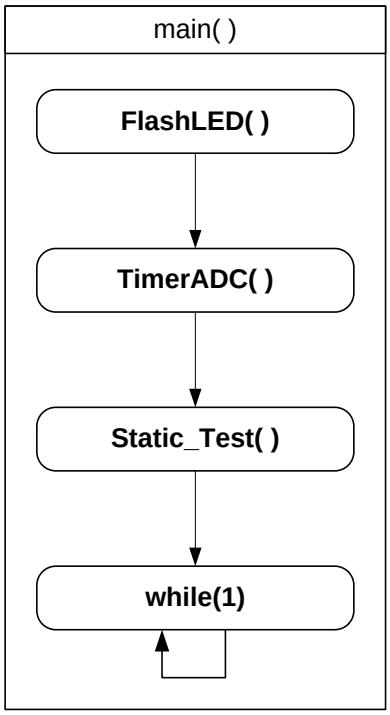


Figure 3-1: Tutorial Sample Flow

3.1.2 Operation

1. The tutorial code initialises the LCD module, and displays ‘Renesas’ on the first line of the LCD, and the name of the MCU on the second line.
2. The tutorial code calls the Flash LED function, which creates a CMT interrupt to toggle the LEDs repeatedly and waits in a loop until either a switch is pressed or the LEDs flash 200 times.
3. The tutorial then calls the TimerADC function which configures the ADC unit, and a timer unit to periodically trigger an ADC conversion. The ADC unit is configured to call the function CB_ADConversion, after every AD conversion completes.
4. When the timer unit period elapses, it triggers an AD conversion. Once the AD conversion completes, the callback function CB_ADConversion is executed. The callback function fetches the ADC result, and uses it to calculate a new timer period. The callback function also toggles the user LEDs.
5. After calling TimerADC and setting up the timer & ADC interrupts, the tutorial calls the Statics_Test function.
6. The Statics_Test function displays the string STATIC on the second line of the debug LCD, and replaces it letter by letter with the constant string, TESTTEST. Once replacement is complete, the LCD reverts back to its original display. The tutorial then waits in an infinite while loop.

3.1.3 Sequence Diagram

Figure 3- below shows the program execution flow of the tutorial sample.

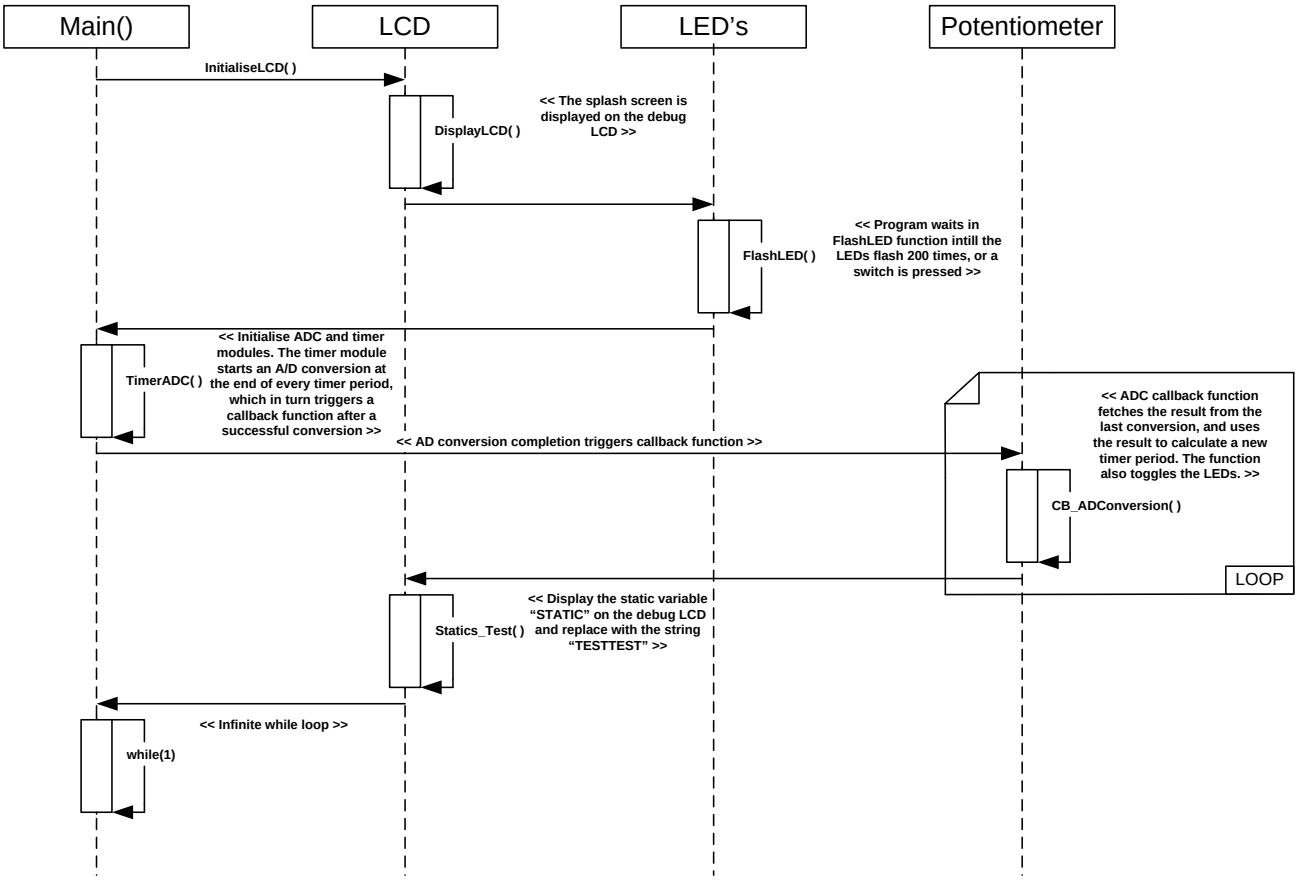


Figure 3-2: Tutorial Sequence Diagram

3.1.4 RPDL Integration

Table 3-1 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Table 3-1: Tutorial Sample RPDL Integration

Function	RPDL Function
FlashLED	R_CMT_Create
	R_CMT_Destroy
TimerADC	R_CMT_Create
	R_ADC_10_Create
CB_ADConversion	R_ADC_10_Read
	R_CMT_Control
CB_TimerADC	R_ADC_10_Control
Statics_Test	R_CMT_CreateOneShot

3.2 Application

3.2.1 Description

The application sample is intended as a starting platform for the user to write his/her own code. The sample includes all the necessary initialisation code and configuration settings from previous samples. The main( ) function contains no sample code, and performs no additional functionality.

For more information regarding the hardware initialisation performed before the main( ) function starts, refer back to §2.

4. Peripheral Samples

The sample code in this section provides examples of initialisation and usage of some of the MCU's peripheral modules. The sample code also provides examples of how to debug MCU peripherals.

4.1 ADC10_OneShot

4.1.1 Description

This sample code demonstrates usage of the on-chip 10-bit analogue to digital converter (ADC), in one shot mode. The sample configures the ADC to read from the potentiometer fitted to the RSK (RV1) when user switch 'SW3' is pressed.

Note: The potentiometer is fitted to offer an easy method of supplying a variable analogue input to the microcontroller. It does not necessarily reflect the accuracy of the controllers ADC. Refer the device hardware manual for further details.

4.1.2 Operation

1. The sample first initialises the debug LCD, and displays instructions on the screen.
2. The sample then calls the Init_ADC10Oneshot function, which configures the ADC unit and the switch callback function.
3. The sample then waits in an infinite while loop, and the rest of the sample's functionality is completed through interrupts.
4. When switch SW3 is pressed, the switch interrupt calls the callback function CB_ReadADC is excuted. This function triggers an AD conversion, converts the result into a character string and then displays the string on the debug LCD.
5. Repressing switch SW3 will trigger another conversion, with the result being displayed on the debug LCD.

4.1.3 Sequence Diagram

Figure 4-1 below shows the program execution flow of the ADC10_OneShot sample.

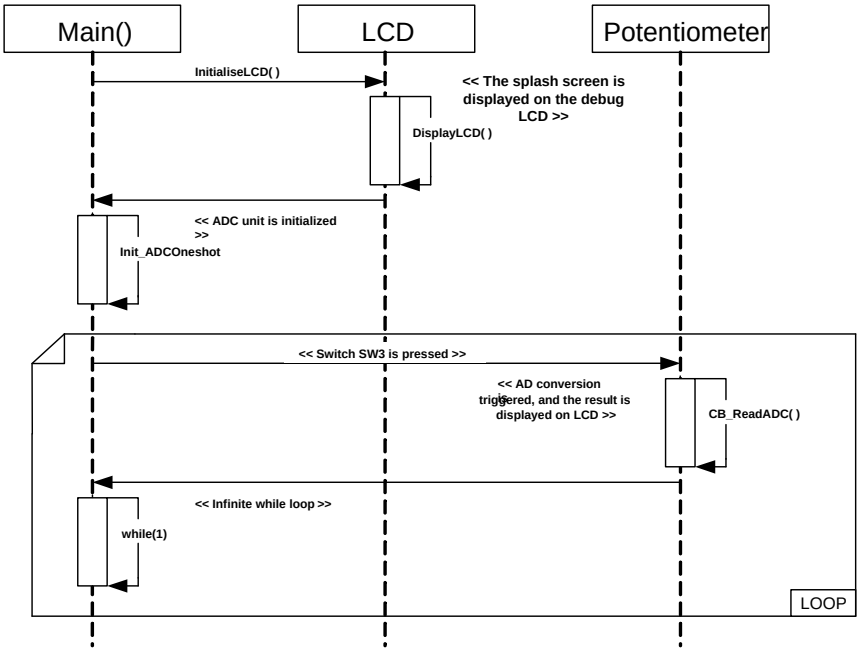


Figure 4-1: ADC_OneShot Sequence Diagram

4.1.4 RPDL Integration

Table 4-1 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_ADCOneshot	R_ADC_10_Create
CB_ReadADC	R_ADC_10_Control
	R_ADC_10_Read

Table 4-1: ADC_OneShot Sample RPDL Integration

4.2 ADC10_Repeat

4.2.1 Description

This sample code demonstrates usage of the on-chip 10-bit analogue to digital converter (ADC), in repeat mode. The sample configures the ADC to repeatedly take readings of the potentiometer voltage (RV1). The sample then updates the conversion value displayed on the LCD, by periodic interrupts from the timer module.

Note: The potentiometer is fitted to offer an easy method of supplying a variable analogue input to the microcontroller. It does not necessarily reflect the accuracy of the controllers ADC. Refer the device hardware manual for further details.

4.2.2 Operation

1. The sample first initialises the debug LCD, and displays the name of the sample.
2. The sample then calls the Init_ADC10Repeat function, which configures the ADC unit to operate in repeat mode, and also configures a compare match timer to generate a periodic interrupt and call the callback function, CB_CMTADC.
3. The sample then enters the infinite while loop, and waits until a CMT period interrupt occurs and calls the callback function, CB_CMTADC.
4. The callback function CB_CMTADC fetches the last AD conversion result from the ADC unit, converts it into a character string and then displays on the debug LCD. The periodic interrupt is called every 250ms.

4.2.3 Sequence Diagram

Figure 4-2 below shows the program execution flow of the ADC10_Repeat sample.

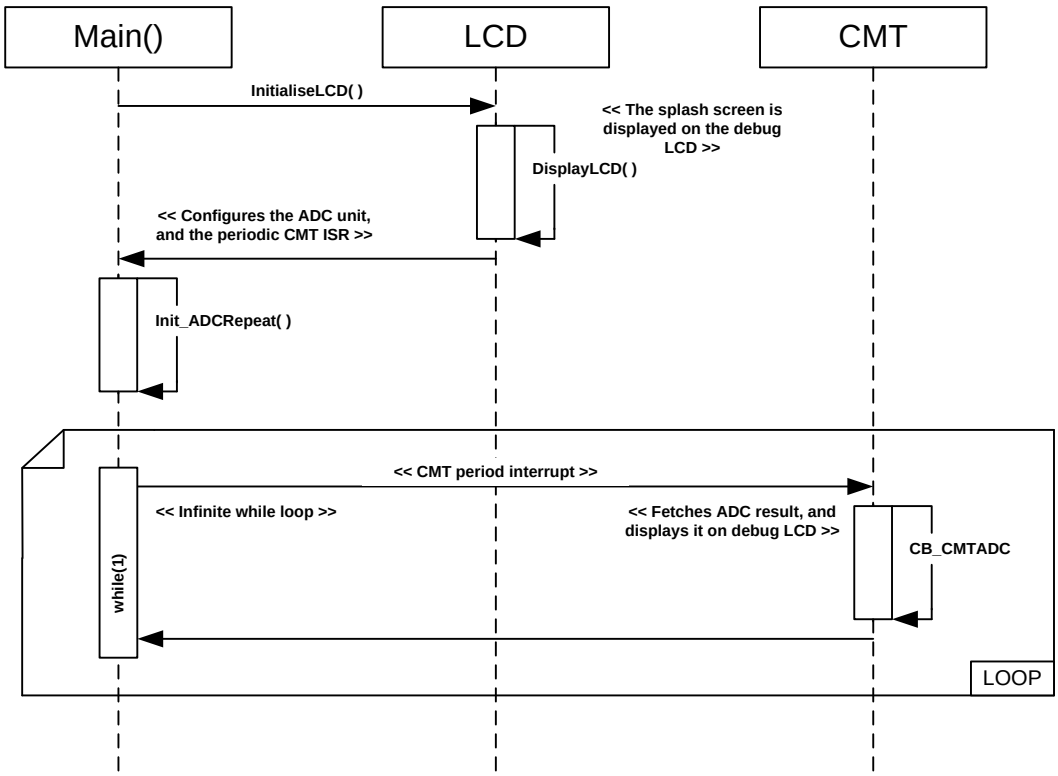


Figure 4-2: ADC_Repeat Sequence Diagram

4.2.4 RPDL Integration

Table 4-2 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_ADCRepeat	R_ADC_10_Create
	R_ADC_10_Control
	R_CMT_Create
CB_CMTADC	R_ADC_10_Read

Table 4-2: ADC10_Repeat Sample RPDL Integration

4.3 ADC12_OneShot

4.3.1 Description

This sample code demonstrates usage of the on-chip 12-bit analogue to digital converter (ADC), in one shot mode. The sample configures the ADC to read from the potentiometer fitted to the RSK (RV1) when user switch 'SW3' is pressed.

Note: The potentiometer is fitted to offer an easy method of supplying a variable analogue input to the microcontroller. It does not necessarily reflect the accuracy of the controllers ADC. Refer the device hardware manual for further details.

4.3.2 Operation

1. The sample first initialises the debug LCD, and displays instructions on the screen.
2. The sample then calls the Init_ADC12Oneshot function, which configures the ADC unit and the switch callback function.
3. The sample then waits in an infinite while loop, and the rest of the sample's functionality is completed through interrupts.
4. When switch SW3 is pressed, the switch interrupt calls the callback function CB_ReadADC is excuted. This function triggers an AD conversion, converts the result into a character string and then displays the string on the debug LCD.
5. Repressing switch SW3 will trigger another conversion, with the result being displayed on the debug LCD.

4.3.3 Sequence Diagram

Figure 4-1 below shows the program execution flow of the ADC12_OneShot sample.

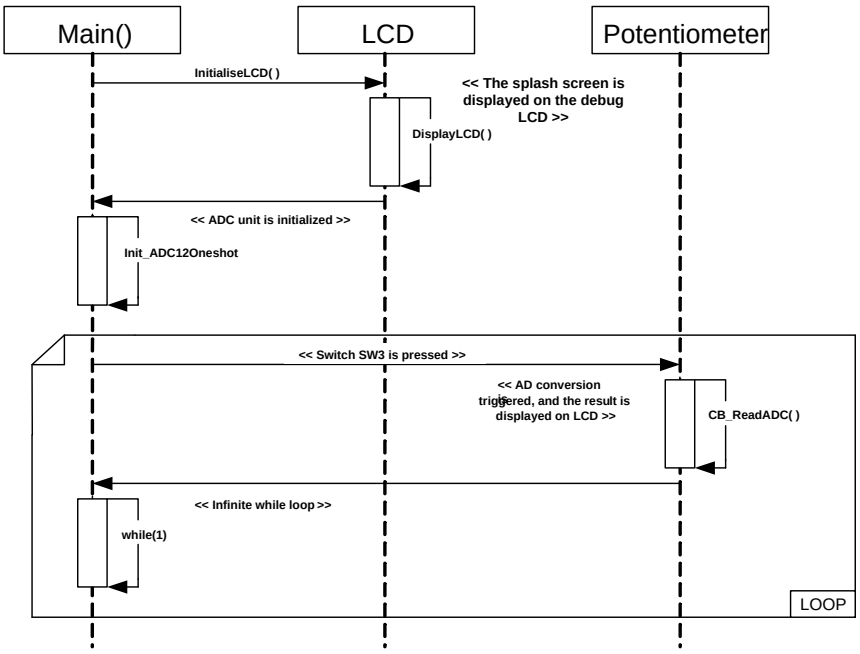


Figure 4-3: ADC12_OneShot Sequence Diagram

4.3.4 RPDL Integration

Table 4-2 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_ADC12OneShot	R_ADC_12_CreateUnit
CB_ReadADC	R_ADC_12_Control
	R_ADC_12_Read

Table 4-3: ADC12_OneShot Sample RPDL Integration

4.4 ADC12_Repeat

4.4.1 Description

This sample code demonstrates usage of the on-chip 12-bit analogue to digital converter (ADC), in repeat mode. The sample configures the ADC to repeatedly take readings of the potentiometer voltage (RV1). The sample then updates the conversion value displayed on the LCD, by periodic interrupts from the timer module.

Note: The potentiometer is fitted to offer an easy method of supplying a variable analogue input to the microcontroller. It does not necessarily reflect the accuracy of the controllers ADC. Refer the device hardware manual for further details.

4.4.2 Operation

1. The sample first initialises the debug LCD, and displays the name of the sample.
2. The sample then calls the Init_ADC12Repeat function, which configures the ADC unit to operate in repeat mode, and also configures a compare match timer to generate a periodic interrupt and call the callback function, CB_CMTADC.
3. The sample then enters the infinite while loop, and waits until a CMT period interrupt occurs and calls the callback function, CB_CMTADC.
4. The callback function CB_CMTADC fetches the last AD conversion result from the ADC unit, converts it into a character string and then displays on the debug LCD. The periodic interrupt is called every 250ms.

4.4.3 Sequence Diagram

Figure 4-2 below shows the program execution flow of the ADC12_Repeat sample.

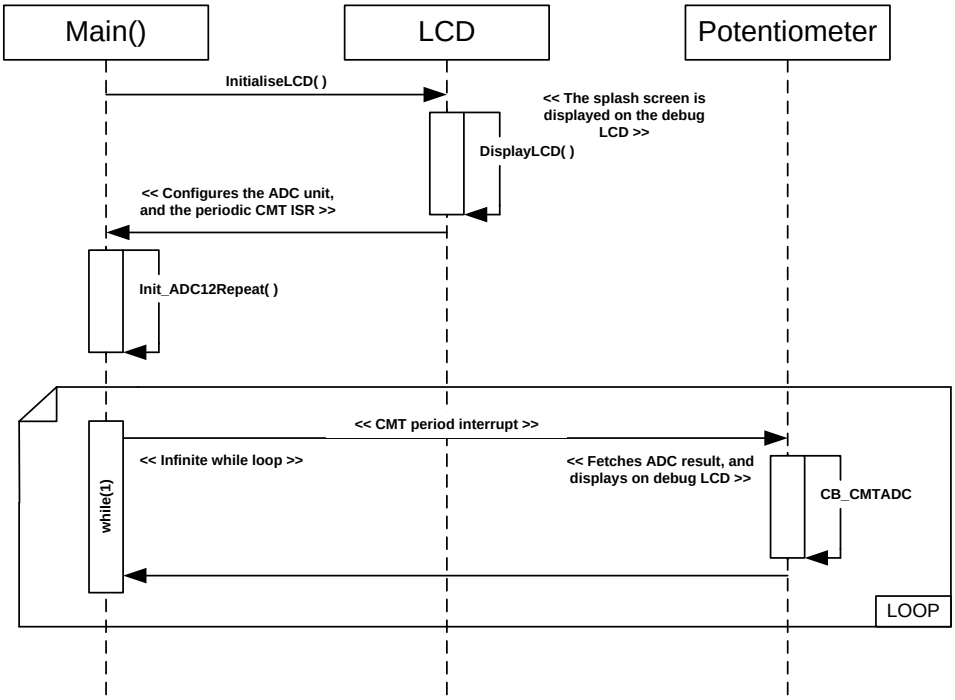


Figure 4-4: ADC12_Repeat Sequence Diagram

4.4.4 RPDL Integration

Table 4-2 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_ADC12Repeat	R_ADC_12_CreateUnit
	R_ADC_12_Control
	R_CMT_Create
CB_CMTADC	R_ADC_12_Read

Table 4-4: ADC12_Repeat Sample RPDL Integration

4.5 SPI

4.5.1 Description

This sample code demonstrates usage of serial peripheral interface (SPI), configured in loopback mode. The SPI module is setup to perform loop back communication between its internal transmitter and receiver pins.

4.5.2 Operation

5. The sample initialises the LCD module, and displays the sample name.
6. The sample then calls the Init_SPI function which configures the SPI channel in master mode with the number of frame transfers set to 1. The loopback mode with direct data transfer, no bit manipulation is used. The number of bits in a frame is set to 16.
7. An ADC channel is configured for single-shot mode to provide the transmit data. Callback function CB_Switch is specified as the switch release callback function.
8. The sample then waits in an infite loop.
9. Switch presses causes an interrupt generation which calls the function CB_Switch. This function checks if the switch pressed was SW3, and in the case of a SW3, a transfer is started.
10. The transmitted data is compared to the received data. Upon a successful transfer, the debug LCD is updated with the received data. Should the transfer be unsuccessful, a message will be displayed notifying the user of a transfer fail.

4.5.3 Sequence Diagram

Figure 4-**5** below shows the program execution flow of the SPI sample.

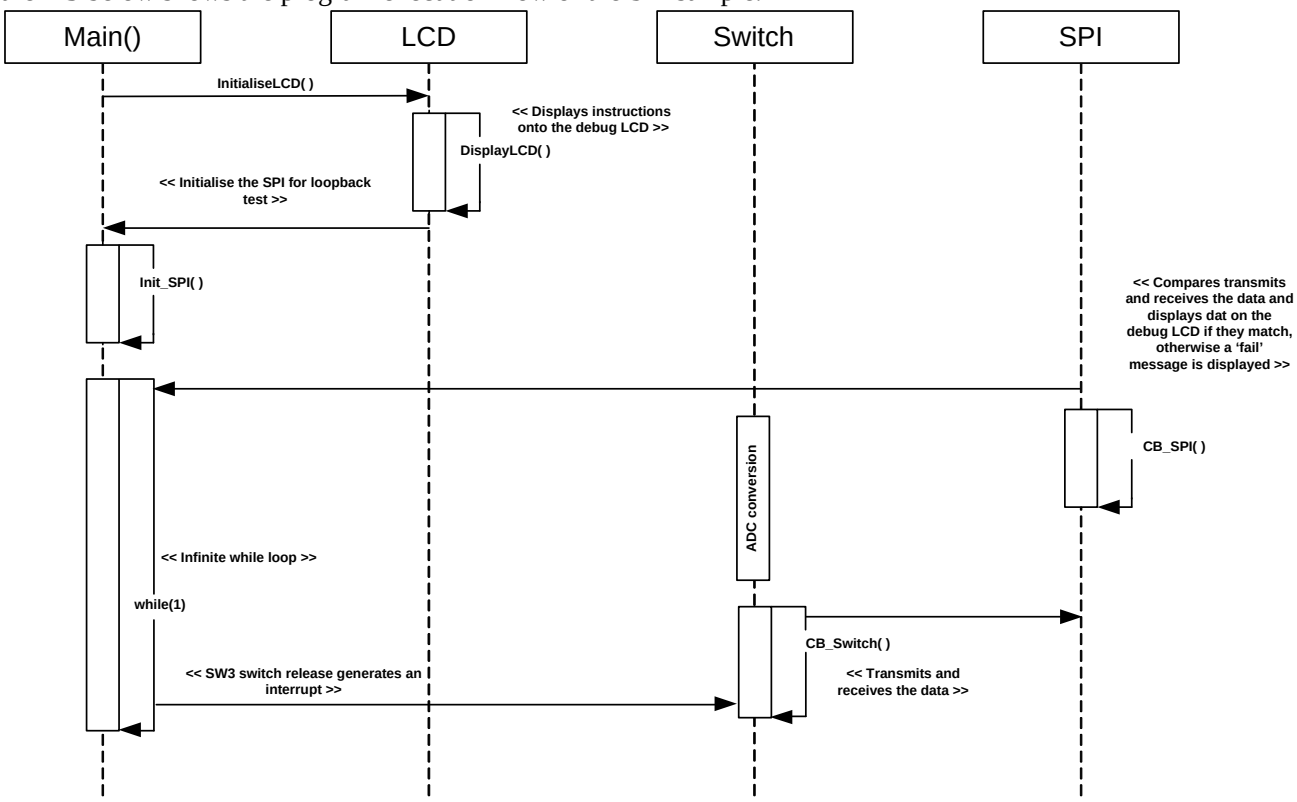


Figure 4-5: SPI Sequence Diagram

4.5.4 RPDL Integration

Table 4-5 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_SPI	R_SPI_Create
	R_SPI_Control
	R_SPI_Command
	R_ADC_Create
CB_Switch	R_ADC_Control
	R_ADC_Read
	R_SPI_Transfer

Table 4-5: SPI Sample RPDL Integration

4.6 LIN Master

4.6.1 Description

This sample code demonstrates usage of Local Interconnect Network (LIN), configured in master or loopback mode, depending on user selection. The LIN module is setup to perform loopback communication between its internal transmitter and receiver pins.

4.6.2 Operation

1. The sample initialises the LCD module, and displays either “LIN_Mstr” or “LIN LOOP” on the first line of the debug LCD, depending on the selected operation mode, and “Push SW3” on the second line.
2. The sample then calls the Init_LIN function which configures the LIN channel in master mode with the interrupt callback function CB_ReadStatus. In master mode only one switch callback function is configured, CB_FrameTransmit. Two switch callback functions are configured for the loopback mode. Switch callback function CB_Self_Test is called on switch press detections to configure the LIN module in self-test mode and configure an ADC channel for single mode operations. Switch callback function LIN_Self_Test is called on switch release detections to start an A/D conversion, read it and display the value on the debug LCD.
3. The sample then waits in an infinite loop.
4. In master mode, switch presses causes an interrupt generation which calls the function CB_FrameTransmit. This function checks if the switch pressed was SW3, and in the case of a SW3 press, a transfer is started. The status of the transfer is verified to determine whether it was successful or not. If successful, the received data is displayed on the debug LCD otherwise a fail message is displayed. In loopback mode, SW3 presses configures the LIN module in self-test mode and sets up the ADC for single shot mode operations. Releasing the switch starts an A/D conversion and reads the result before converting the 16-bit result to an 8b-byte ASCII string. The bytes are bitwise inverted and loaded into the LIN data buffers. The data is shifted out of the data buffers through the TX pin and shifted back in from the RX pin which is connected to the TX pin.
5. The transmitted data is compared to the received data. Upon a successful transfer where the sent data and received data match, the debug LCD is updated with the received data. Should the transfer be unsuccessful, a message will be displayed notifying the user of a transfer fail.

4.6.3 Sequence Diagram

Figure 4-**6** below shows the program execution flow of the SPI sample.

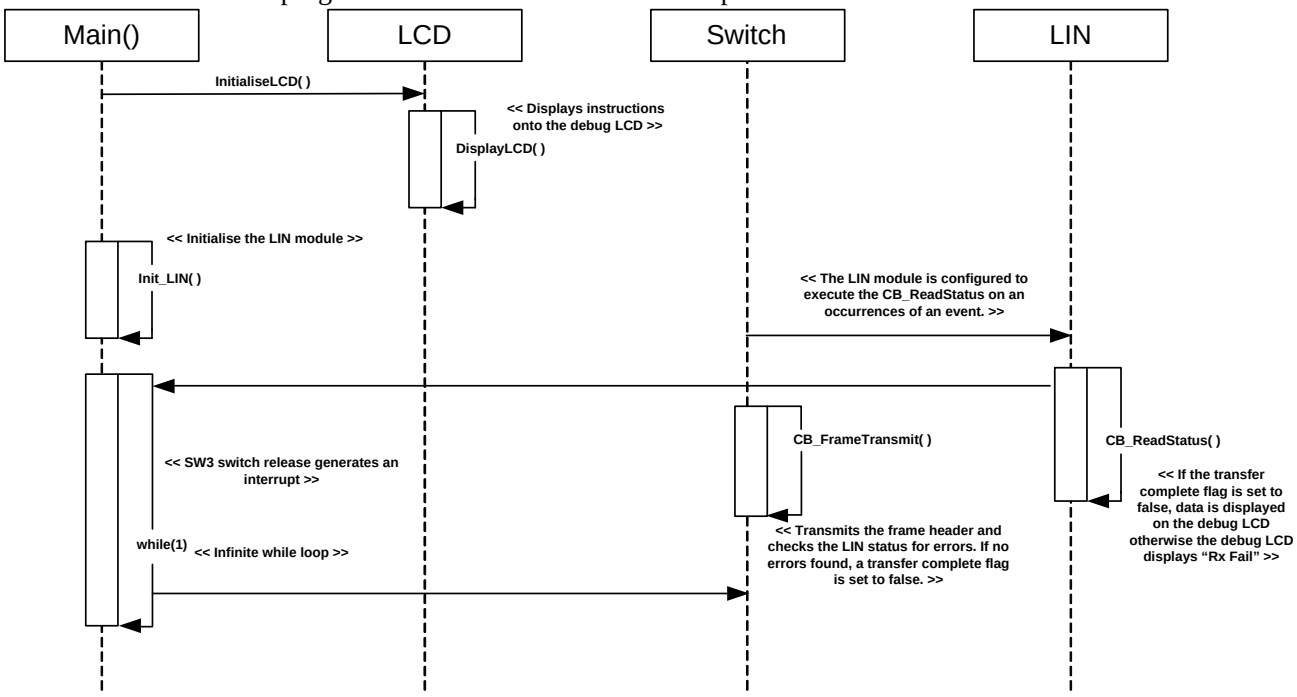


Figure 4-6: LIN Master Sequence Diagram

4.6.4 RPDL Integration

Table 4-6 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_LIN	R_IO_PORT_Write
	R_IO_PORT_Set
	R_LIN_Create
	R_CMT_CreateOneShot
CB_FrameTransmit	R_CMT_CreateOneShot
	R_LIN_Transfer
	R_LIN_Read
	R_LIN_GetStatus
LIN_Self_Test	R_ADC_10_Control
	R_ADC_10_Read
	R_LIN_Transfer
	R_LIN_Read
CB_Self_Test	R_LIN_Control
	R_ADC_10_Create

Table 4-6: LIN Master Sample RPDL Integration

4.7 Async_Serial

4.7.1 Description

This sample code demonstrates usage of serial communications interface (SCI), configured in asynchronous mode. The SCI module is setup to communicate to a PC running a terminal emulator program, via an RS-232 cable.

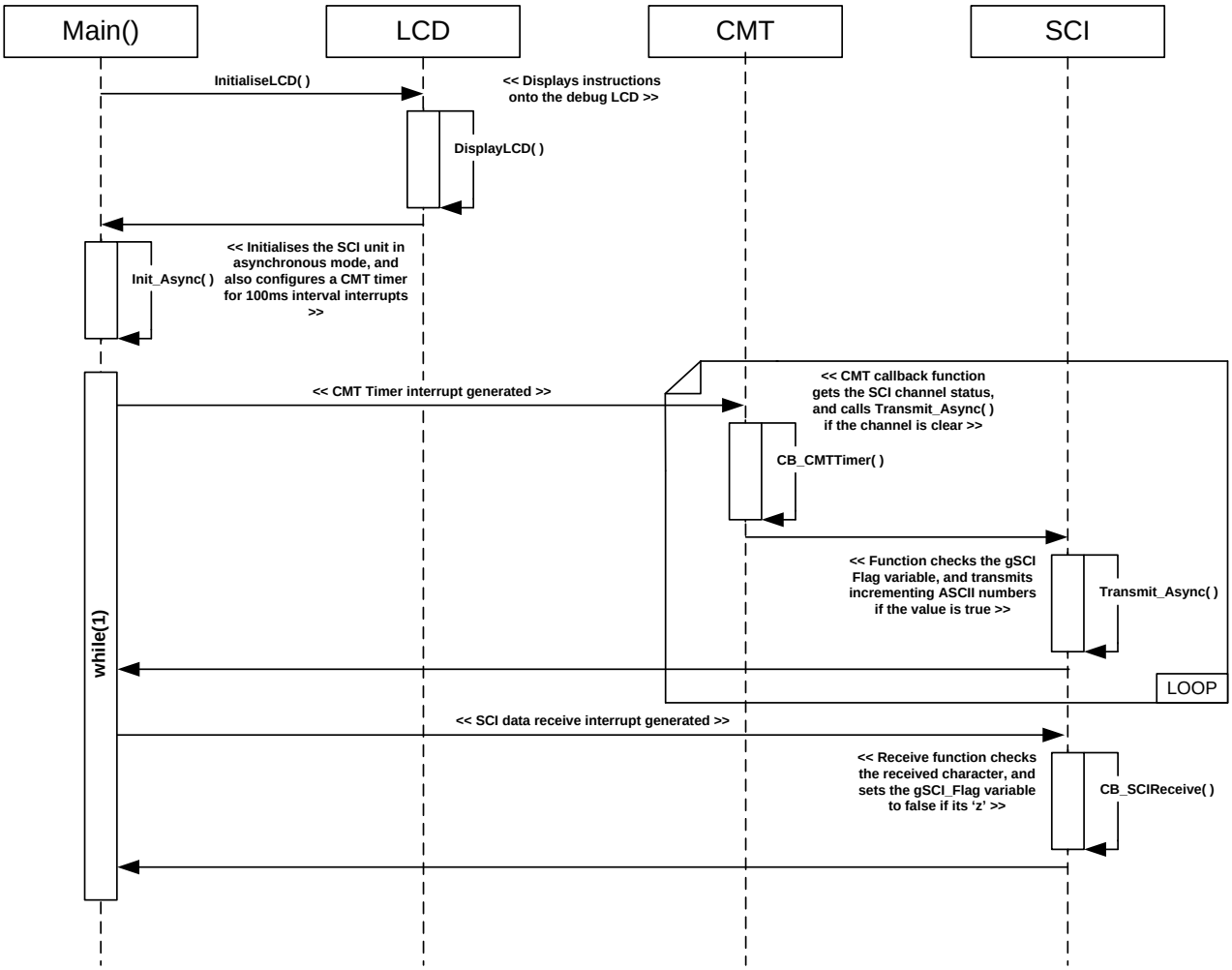
4.7.2 Operation

1. Before the sample begins, the user should connect the RSK to a PC via an RS-232 cable and start the terminal program (refer to the instructions in the sample code comments).
2. The sample initialises the LCD module, and displays ‘Async’ on the first line and ‘Serial’ on the second.
3. The sample configures the SCI channel and a CMT channel with the function Init_Async, and transmits instructions to the terminal display. The program then returns to an infinite while(1) – the rest of the sample’s functionality is achieved through interrupts.
4. The CMT channel generates a periodic interrupt every 100ms, and calls the callback function CB_CMTTimer.
5. The function CB_CMTTimer fetches the SCI channel status, and calls the function Transmit_Async if the channel is clear. If the channel is busy, the function returns to the while(1) loop.
6. The Transmit_Async function checks the global flag gSCI_Flag, and transmits an incrementing ASCII number (loops back to 0 after 9) to the terminal display if gSCI_Flag is true. If the flag is false, the function returns without writing to the terminal.

4.7.3 Sequence Diagram

Figure 4-7 below shows the program execution flow of the Async_Serial sample.

Figure 4-7: Async_Serial Sequence Diagram



4.7.4 RPDL Integration

Table 4-7 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_Async	R_SCI_Create
	R_SCI_Receive
	R_SCI_Send
	R_CMT_Create
CB_CMTTimer	R_SCI_GetStatus
Transmit_Async	R_SCI_Send
CB_SCIReceive	R_SCI_Receive

Table 4-7: Async_Serial Sample RPDL Integration

4.8 Sync_Serial

4.8.1 Description

This sample code demonstrates usage of serial communications interface (SCI), configured in synchronous mode. The SCI module is setup to perform loop back communication between two SCI channels, using a 3-wire interface.

4.8.2 Operation

1. The sample initialises the LCD module, and displays the sample name.
2. The sample then calls the Init_Sync function which clears the reserved SCI data buffers.
3. The sample then calls the function SCI2toSCI0Transfer_Sync, which configures the SCI channels and then transfers a data string from channel 2 to channel 0. The callback function CB_SCI0Receive is called when the data is received, and it checks the data received is correct.
4. The sample then calls the function SCI0toSCI2Transfer_Sync, which transfer data from channel 0 to channel 2. The callback function CB_SCI2Receive is called when the data has been received.
5. The function CB_SCI2Receive checks if both transfers were successful, and displays “Success” on the debug LCD. If any of the transfers failed, the function reports “Failure” on the LCD.
6. The sample then enters an infinite while loop.

4.8.3 Sequence Diagram

Figure 4-8 below shows the program execution flow of the Sync_Serial sample.

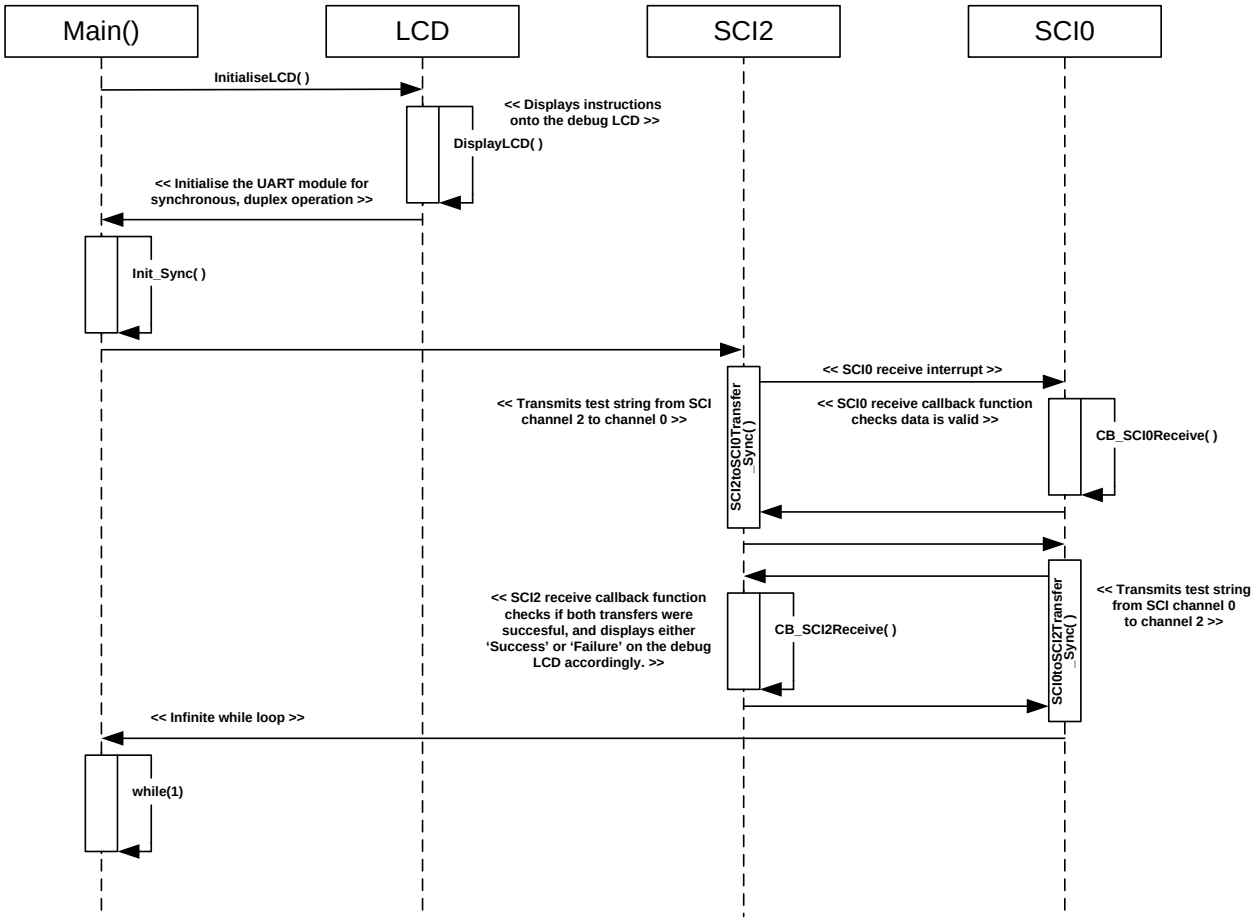


Figure 4-8: Sync_Serial Sequence Diagram

4.8.4 RPDL Integration

Table 4-8 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
SCI0toSCI2Transfer_Sync	R_SCI_Create
	R_SCI_Receive
	R_SCI_Send
SCI2toSCI0Transfer_Sync	R_SCI_Create
	R_SCI_Receive
	R_SCI_Send

Table 4-8: Sync_Serial Sample RPDL Integration

4.9 Power_Down

4.9.1 Description

In this sample, the LPC (Low Power Consumption) registers are configured to enter the MCU into standby mode by pressing a switch, and wake again from an interrupt.

4.9.2 Operation

1. The sample initialises the LCD module, and displays ‘Pwr Mode’ (Power Mode) on the first line of the LCD, and the current power mode, ‘Active’ on the second line.
2. The sample then calls the function Init_PowerDown which configures the standby registers.
3. The sample then calls the Flash_LEDs function, which uses a CMT one-shot timer to create a delay to flash all the user LEDs. The function waits in a while loop, polling the switch flag variable, gSwitchFlag.
4. When a user presses switch SW1, Flash_LEDs calls the function Standby_PowerDown.
5. The function Standby_PowerDown sets the second line of the LCD to ‘Standby’, and turns off the user LEDs and prepares the MCU for entering standby by polling the variable gSwitchStandbyReady, and waits until it is true. If a user is still holding down one of the switches, the function lights LED3 to indicate it is waiting to enter standby. When all switches are released, all LEDs are turned off and the MCU enters standby mode.
6. Pressing any of the switches will wake the MCU from standby. The second line of the LCD changes to ‘Active’ and the LEDs are switched on.

4.9.3 Sequence Diagram

Figure 4-9 below shows the program execution flow of the Power_Down sample.

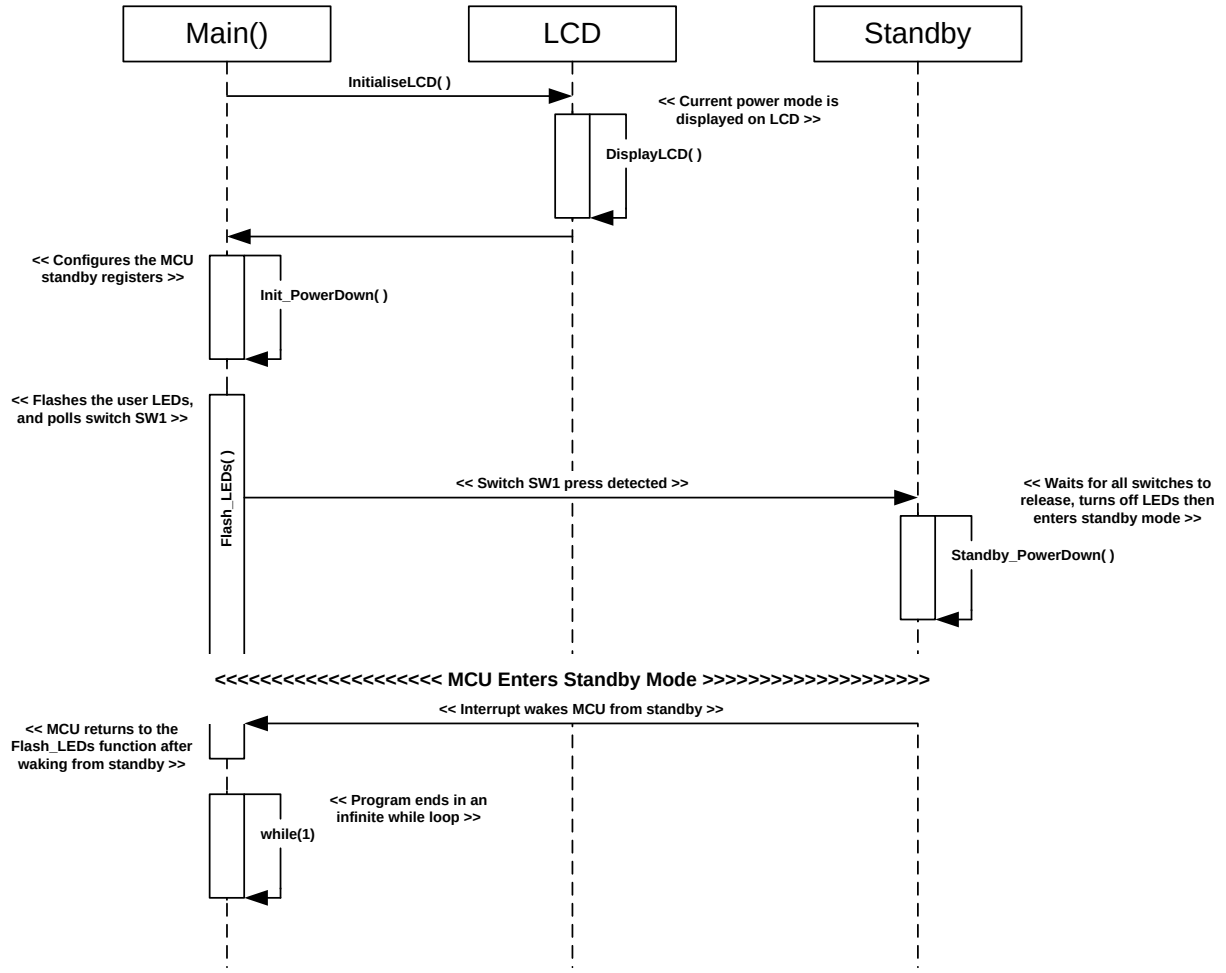


Figure 4-9: Power_Down Sequence Diagram

4.9.4 RPDL Integration

Table 4-9 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_PowerDown	R_LPC_Create
Flash_LEDs	R_CMT_CreateOneShot
	R_IO_PORT_Modify
	R_IO_PORT_Write
Standby_PowerMode	R_IO_PORT_Write
	R_LPC_Control

Table 4-9: Power_Down Sample RPDL Integration

4.10 LVD

4.10.1 Description

In this sample, the LVD (Low Voltage Detection) circuit is configured to generate an interrupt when the power supply equals or falls below the detection level.

4.10.2 Operation

1. The sample calls the function Init_LVD which configures the LVD1 circuit to generate an interrupt each time the power supply falls below the set detection level.
2. The LVD interrupt is non-maskable and handled differently from maskable interrupts. The LVD interrupt is configured as an NMI with falling edge detection along with the callback function CB_LVD1. CB_LVD1 turns off LED0-LED2 and turns on LED3 each time it is called. A periodic timer, with callback function CB_CMTimer is also configured to generate periodic interrupts every 500ms, used to synchronise the toggling of LEDs.
3. The sample then enters an infinite while loop which is interrupted on every low voltage detection and CMT period timeouts.

4.10.3 Sequence Diagram

Figure 4-10 below shows the program execution flow of the Low Voltage Detection sample.

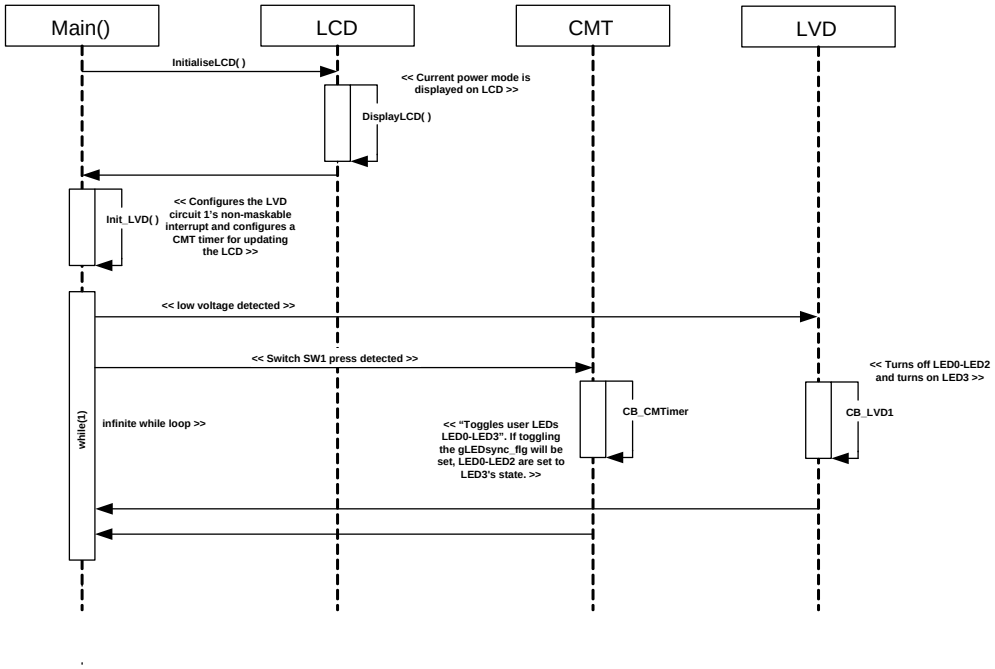


Figure 4-10: LVD Sequence Diagram

4.10.4 RPDL Integration

Table 4-10 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_LVD	R_LVD_Control
	R_INTC_CreateExtInterrupt
	R_CMT_Create
CB_CMTimer	R_IO_PORT_Modify
CB_LVD1	R_CMT_Control
	R_IO_PORT_Write

Table 4-10: Low Voltage Detection Sample RPDL Integration

4.11 IIC_Master

4.11.1 Description

This sample demonstrates usage of the IIC unit in master mode, by performing read and write operations to an EEPROM memory device. The sample is configured to work with following Renesas devices:

- HN58X24512I, 1Mbit EEPROM, 1MHz

4.11.2 Operation

1. The sample initialises the LCD module, and displays the sample name on the screen.
2. The sample then enters the main IIC master sequence loop, where it first calls the Init_EEPROM_Master function which configures the IIC unit to operate in master mode.
3. The master sequence then waits in an infinite while loop, polling the user switches.
4. When switch is pressed, the switch callback function executes, and identifies which switch has been pressed.
5. When switch SW2 is pressed, an EEPROM write operation is executed using the Write_EEPROM_Master function. The write operation writes the string “XXRenesas IIC ”, where XX is replaced with an ASCII data identifier, which increments with every write operation.
6. The write operation always starts from the EEPROM’s first memory address and after every successful write, displays the data identifier of the written string on the debug LCD. If the write operation fails, the debug LCD displays “Error W.”
7. When switch SW3 is pressed, an EEPROM read operation is executed using the Read_EEPROM_Master function. The read operation starts from zero and increments to the next 16byte location after every successful read.
8. If the read data matches the expected data string, “XXRenesas IIC ”, the data identifier (XX) is displayed on the debug LCD to indicate a successful read operation. If the read operation fails, the debug LCD displays “Error R.”.
9. LED1 remains lit whilst IIC acitivity is in progress – during a successful transfer, the light should blink. If the light remains on constantly, the IIC bus has locked up. Restart both the master and slave devices to free the bus again.

4.11.3 Sequence Diagram

Figure 4-11 below shows the program execution flow of the IIC_Master sample.

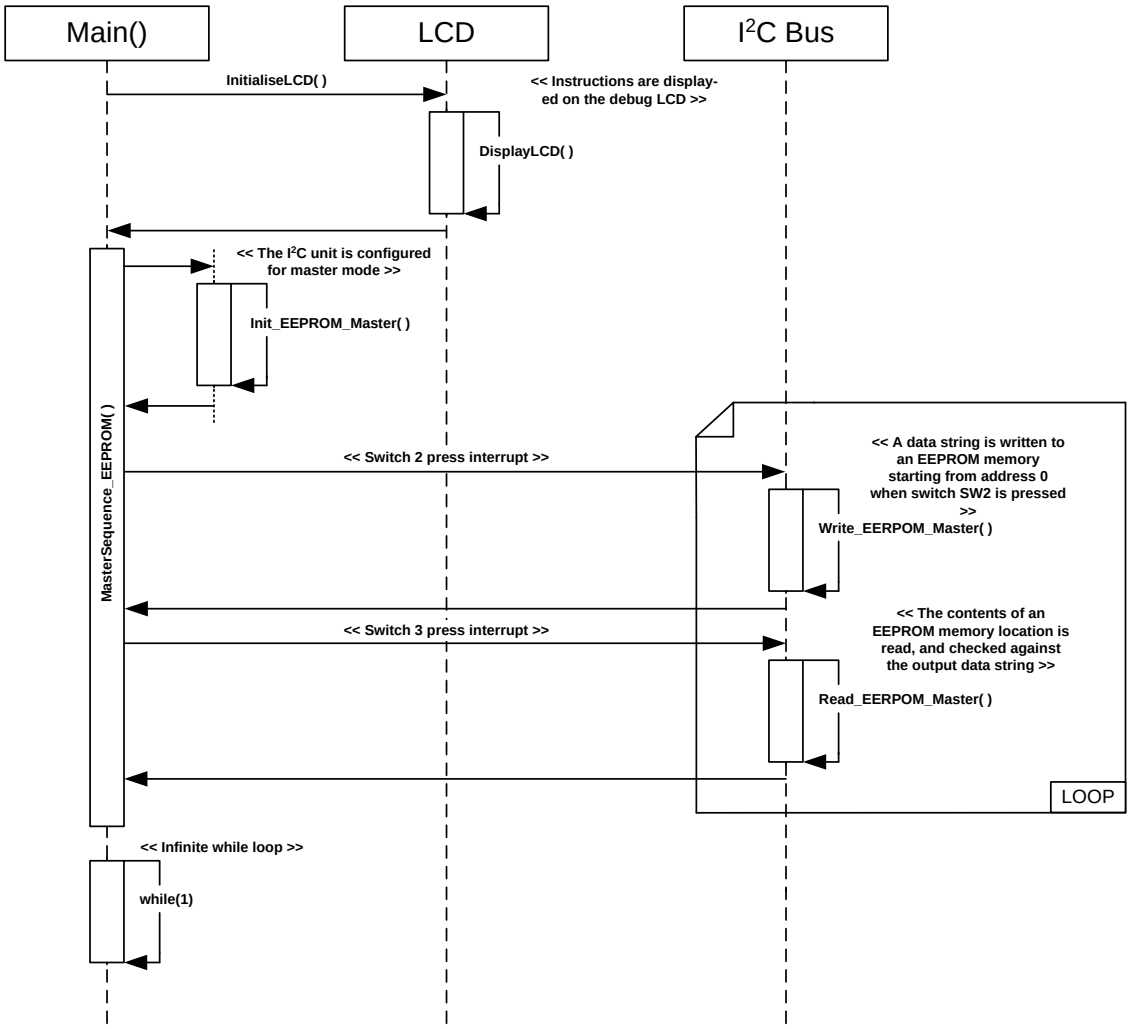


Figure 4-11: IIC_Master Sequence Diagram

4.11.4 RPDL Integration

Table 4-11 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_EEPROM_Master	R_IIC_Create
Write_EEPROM_Master	R_IIC_MasterSend
	R_CMT_CreateOneShot
Read_EEPROM_Master	R_IIC_MasterSend
	R_IIC_MasterReceive
	R_CMT_CreateOneShot

Table 4-11: IIC_Master Sample RPDL Integration

4.12 IIC_Slave

4.12.1 Description

This sample demonstrates usage of the IIC unit in slave mode, by performing simulating a 2k byte EEPROM memory device.

4.12.2 IIC Slave Commands

(1) Write Operation

To write to the simulated EEPROM, the master should send a start condition followed by the EEPROM device address (default address: 0xA0), and wait for an ACK (acknowledgement) signal from the slave.
The master should proceed by sending the 8bit EEPROM memory address (valid from 0x0000 to 0x0800) and then wait for an ACK response from the slave.
The master should then proceed to transmit the write data in bytes, with an ACK response after each data byte. The maximum number of byte writes the slave can support in a single write operation is 16.
Once the final byte has been sent, the master should send a stop signal to end the transaction.
The simulated EEPROM's internal address pointer will auto increment with each byte written.

(2) Read Operation

The read operation will always start from the current internal simulated EEPROM memory pointer, and auto increment to the next byte until the address reaches the maximum value or a stop condition is detected.
To read from an arbitrary memory location, the master should first send a dummy write to the required EEPROM memory address. The dummy write should consist of a start condition, device address and a memory address.
To read from the current internal memory location, the master should send a start condition followed by the EEPROM device address.
The EEPROM slave should then reply with an ACK signal, and send the data located at the current memory location and auto increment the internal pointer to the next byte location.
In order to read another byte, the master should send an ACK signal. The master should repeat this until the required number of bytes has been read, and should end the operation with a stop condition.
The read operation can be used to read data from the entire address range (0x0000 to 0x0800), and will stop only when a stop condition is detected or the last memory address is reached.

4.12.3 Operation

1. The sample initialises the LCD module, and displays the sample name on the screen.
2. The sample then initialises the IIC unit to operate in slave mode, and prepares the simulated EEPROM memory area.
3. The sample then calls the BusMonitor_EEPROM_Slave function, which polls the IIC unit and waits until data is received from a master device on the IIC bus.
4. When data is received from a master device via the IIC bus, the function BusReply_EEPROM is called. This function determines whether the master sent a valid read or write request.
5. If a valid write request is detected, the BusReply_EEPROM function calls the Write_EEPROM_Slave function, which writes the received data to the simulated EEPROM. The internal pointer is set to the address received from the master, and is auto incremented with each byte write.

6. If a valid read request is detected, the BusReply_EEPROM function calls the Read_EEPROM_Slave function which sends contents of the simulated EEPROM memory to the master device until either the end of the simulated memory is reached or a valid stop condition is detected.

7. The program then returns to the bus reply function, which reports the read/write operation's success or failure to the debug LCD. If an invalid request is sent from the master, the function reports a error on the debug LCD.
8. The program then returns back to the BusMonitor_EEPROM_Slave function, where it waits for another master command over the IIC bus.

4.12.4 Sequence Diagram

Figure 4-12 below shows the program execution flow of the IIC_Slave sample.

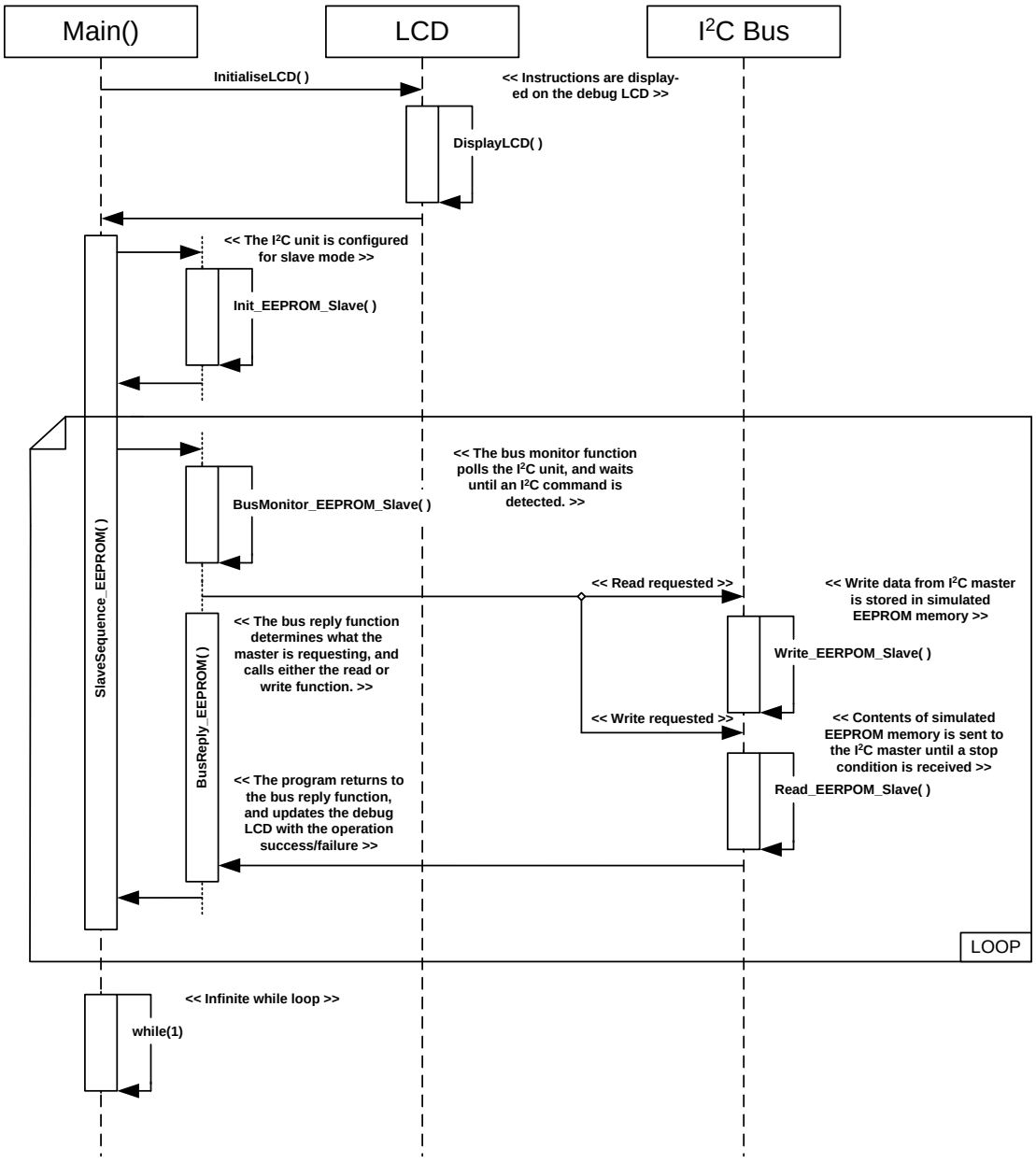


Figure 4-12: IIC_Slave Sequence Diagram

4.12.5 RPDL Integration

Table 4-12 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_EEPROM_Slave	R_IIC_Create
Read_EEPROM_Slave	R_IIC_SlaveSend
BusMonitor_EEPROM_Slave	R_IIC_SlaveMonitor

Table 4-12: IIC_Slave Sample RPDL Integration

4.13 CRC

4.13.1 Description

This sample demonstrates the CRC unit, by echoing typed characters from the SCI terminal with a corresponding checksum.

4.13.2 Operation

1. Before starting this sample, the user should connect the RSK to the PC via an RS232 serial cable and run a suitable terminal program (see instructions in sample code comments).
2. The sample displays “CRC”, “Calc” on the first and second lines of the debug LCD.
3. The sample then calls the function Init_CRC, which configures the CRC unit to produce 16bit ANSI checksums, and the SCI unit for asynchronous operation to the PC terminal.
4. The function also configures an interrupt to be generated when data is received from the terminal, and displays instructions in the terminal window.
5. The sample then enters an infinite while loop, and the rest of the samples functionality is performed in interrupts.
6. When the user presses a key in the terminal, the SCI interrupt executes the callback function CB_SCIReceive. This function takes the received character and calls the function Calculate_CRC to generate a checksum.
7. The sample returns from the Calculate_CRC function to the callback function and writes a string containing the received character and its checksum to the terminal.
8. The sample then returns to the infinite while loop and waits until a key is entered into the terminal again.

4.13.3 Sequence Diagram

Figure 4-13 below shows the program execution flow of the CRC sample.

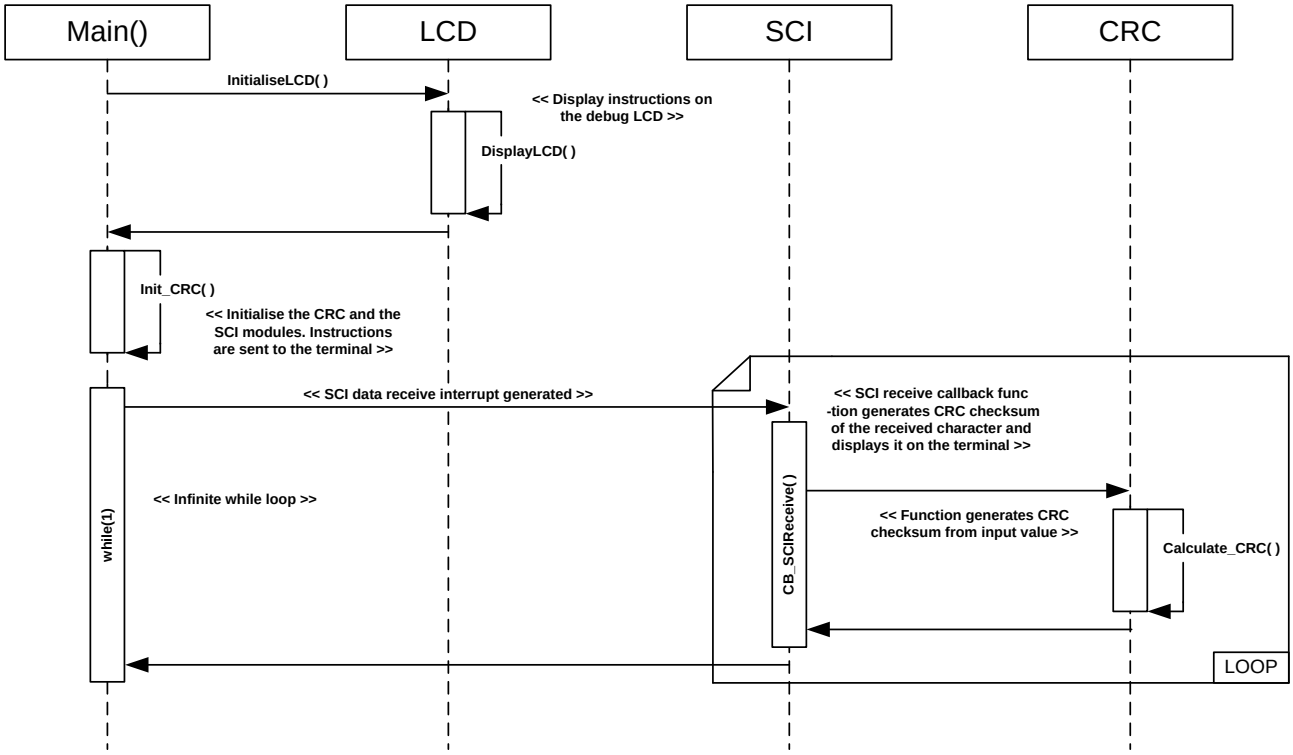


Figure 4-13: CRC Sequence Diagram

4.13.4 RPDL Integration

Table 4-13 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_CRC	R_CRC_Create
	R_SCI_Create
	R_SCI_Receive
	R_SCI_Send
CB_SCIReceive	R_SCI_GetStatus
	R_SCI_Send
	R_SCI_Receive
Calculate_CRC	R_CRC_Write
	R_CRC_Read

Table 4-13: CRC Sample RPDL Integration

4.14 Timer_Capture

4.14.1 Description

This sample configures the timer to run whilst the switch SW1 is held down. Once SW1 is released, the period of time in which the switch was held down is displayed on the debug LCD, in milliseconds.

In order to ensure a clean switch on/off transition, the MCU is insensitive to switch changes for 10ms after each edge change is detected. This time period is to prevent switch 'bounce' from incorrectly triggering switch interrupts. This means the minimum switch press time will always be greater than 20ms. The sample is for demo purposes only, and does not represent the timer accuracy of the MCU.

4.14.2 Operation

1. The sample initialises the LCD module, and displays "Capture" on the first line and "Push SW1" on the second.
2. The sample then configures a CMT timer to generate interrupts every millisecond. The CMT timer is stopped and the count value is reset to 0. Switch press and release callback functions are also set.
3. The sample then enters the main while loop, and the rest of the functionality is performed at interrupt level.
4. When the user presses down switch SW1, an interrupt is generated which calls the function CB_SwitchPress, which resets the count variable usCapture_value and starts the CMT timer.
5. Whilst the user still has SW1 held down, the callback function CB_TimerClockTick will execute every millisecond by CMT timer interrupt. The function increments the variable usCapture_value with each execution.
6. When the user releases SW1, an interrupt is generated which calls the callback function CB_SwitchRelease. The function stops the CMT timer. It then calls the function Update_TimerCapture which checks if the usCapture_value is less than 10 seconds. The value is converted to a string and displayed if it is less than 10 seconds otherwise a message is displayed indicating that the value was greater than 10 seconds.

4.14.3 Sequence Diagram

Figure 4-14 below shows the program execution flow of the Timer_Capture sample.

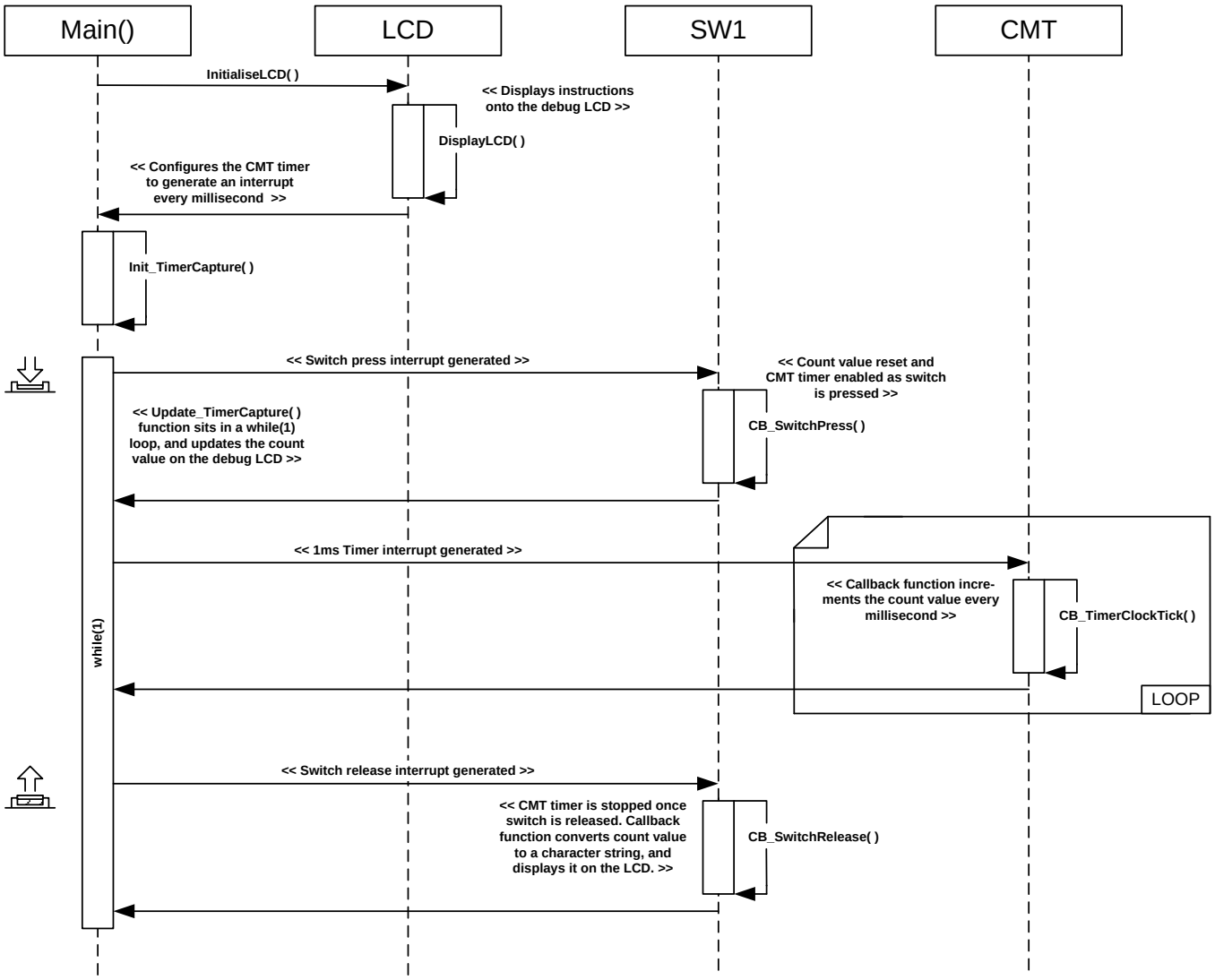


Figure 4-14: Timer_Capture Sequence Diagram

4.14.4 RPDL Integration

Table 4-14 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_TimerCapture	R_CMT_Create
	R_CMT_Control
CB_SwitchPress	R_CMT_Control
CB_SwitchRelease	R_CMT_Control

Table 4-14: Timer_Capture Sample RPDL Integration

4.15 Timer Compare

4.15.1 Description

This sample configures the CMT timer to run, and execute a callback function every time a compare match interrupt occurs. The callback function toggles the state of the user LEDs, making them blink.

4.15.2 Operation

1. The sample initialises the LCD module, and displays “Timer” on the first line and “Compare” on the second.
2. The sample then configures a CMT timer to generate interrupts every 100ms.
3. The sample then enters the main while loop, and is interrupted every 100ms when the CMT timer interrupt calls the CB_CompareMatch callback function.
4. The CB_CompareMatch callback function toggles the state of the user LEDs, making them blink.

4.15.3 Sequence Diagram

Figure 4-15 below shows the program execution flow of the Timer_Compare sample.

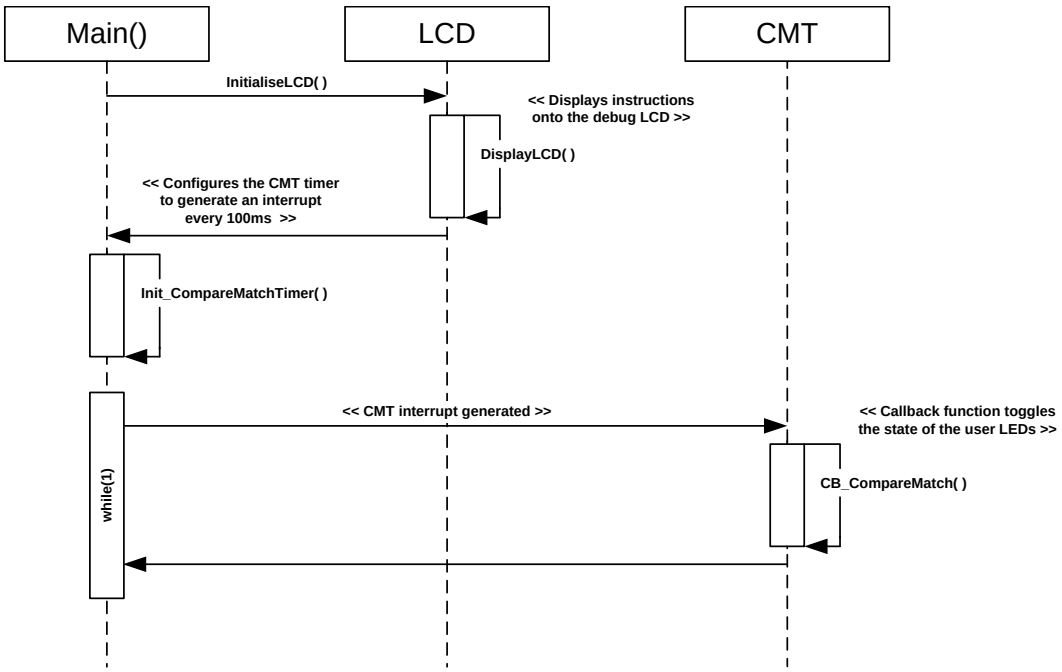


Figure 4-15: Timer_Compare Sequence Diagram

4.15.4 RPDL Integration

Table 4-15 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_CompareMatchTimer	R_IO_PORT_Set
	R_CMT_Create
CB_CompareMatch	R_IO_PORT_Modify

Table 4-15: Timer_Compare Sample RPDL Integration

4.16 Timer Event

4.16.1 Description

This sample configures MTU3 timer 2, to be clocked from negative pulses from SW2 switch actions. The CMT is also configured to update the MTU3 timer's counter display value every 100ms, through the callback function CB_CMT_UpdateLCD.

4.16.2 Operation

1. The sample initialises the LCD module, and displays "TR Event" on the first line and "Push SW2" on the second.
2. The sample then configures MTU3 unit's timer 2 to be clocked from switch SW2's falling edge signals and a CMT timer is configured to generate interrupts every 100ms for updating the debug LCD with the count.
3. The sample then enters the main while loop, and is interrupted every 100ms when the CMT timer interrupt calls the CB_CompareMatch callback function.
4. When called, the CB_CompareMatch callback function stops the timer's counting, reads the count value, converts the integer value to a string then displays it on the debug LCD before exiting the function.

4.16.3 Sequence Diagram

Figure 4-16 below shows the program execution flow of the Timer_Compare sample.

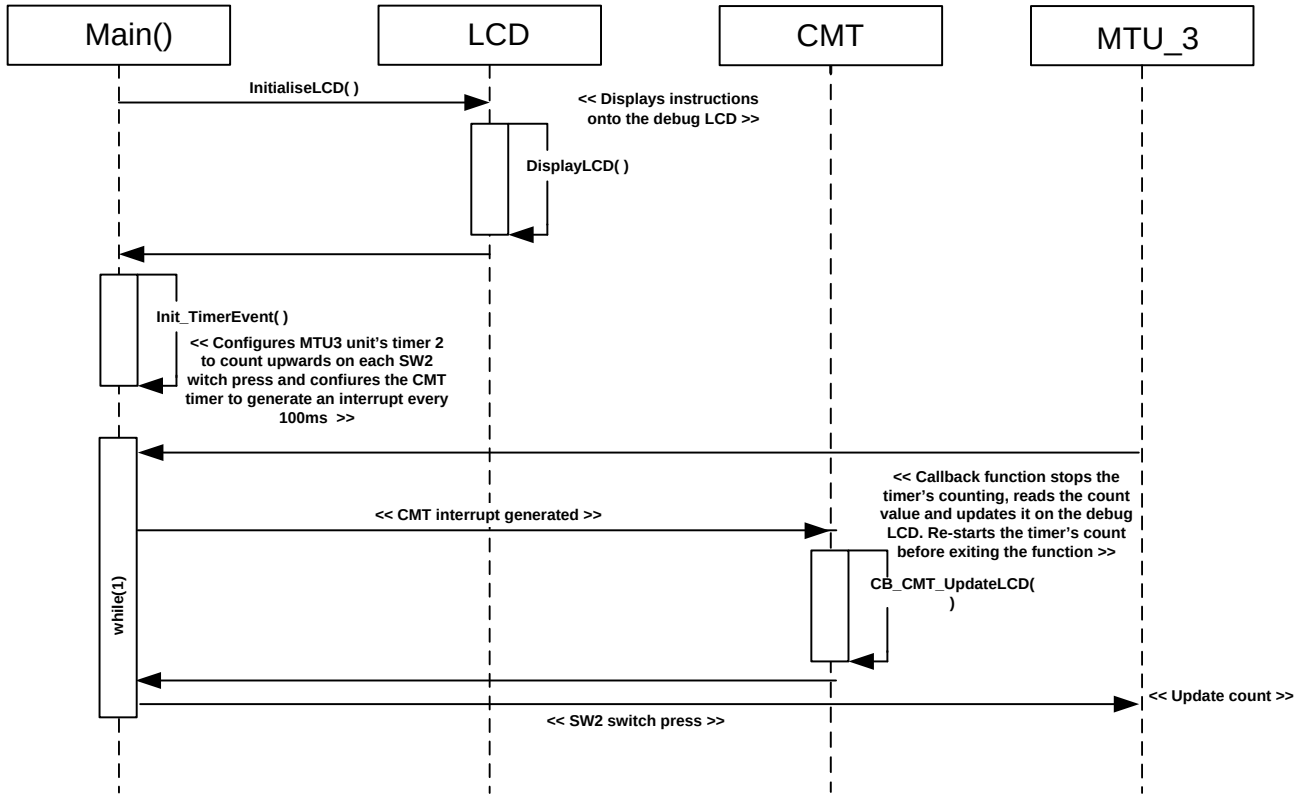


Figure 4-16: Timer_Event Sequence Diagram

4.16.4 RPDL Integration

Table 4-16 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_CompareMatchTimer	R_MTU3_Set
	R_MTU3_Create
	R_CMT_Create
CB_CMT_UpdateLCD	R_MTU3_ControlChannel
	R_MTU3_ReadChannel

Table 4-16: Timer_Event Sample RPDL Integration

4.17 Timer Mode

4.17.1 Description

This sample configures MTU3 unit's timer 2 to run, and toggle the timer output pin every time a compare match occurs. No interrupts are used since the timer is configured to automatically change state on each compare match occurrences.

4.17.2 Operation

1. The sample initialises the LCD module, and displays "1KHz" on the first line and "J2-pin13" on the second.
2. The sample then configures MTU3 unit's timer 2 to output a 1 KHz periodic square wave. The timer toggles a specific output pin, MTIOC2A on every compare match.
3. The sample then enters an infinite while loop.

4.17.3 Sequence Diagram

Figure 4-17 below shows the program execution flow of the Timer_Mode sample.

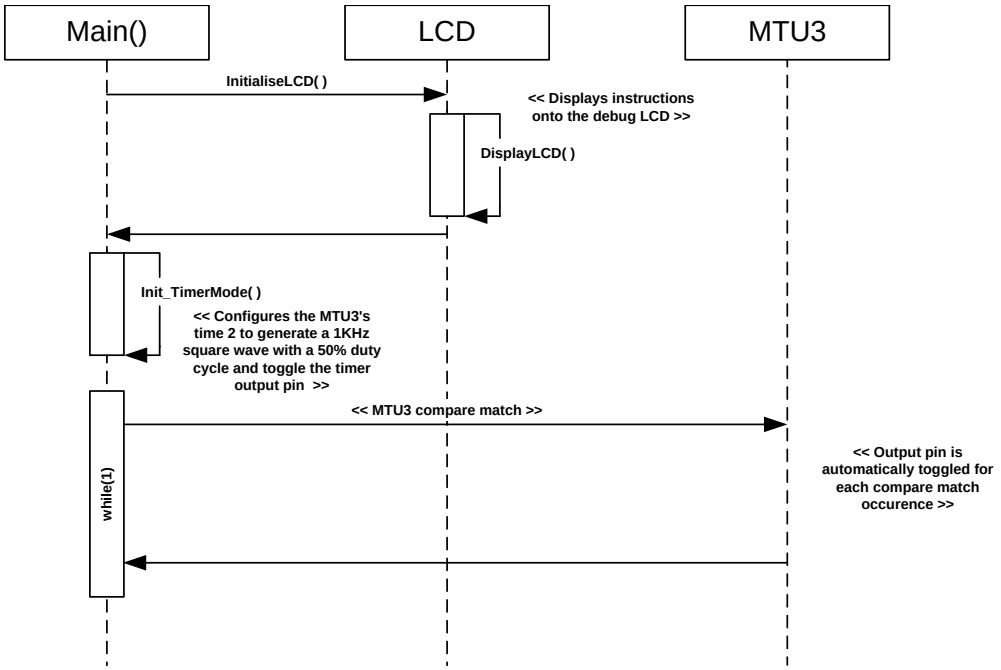


Figure 4-17: Timer_Mode Sequence Diagram

4.17.4 RPDL Integration

Table 4-17 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_TimerMode	R_MTU3_Set
	R_MTU3_ControlChannel

Table 4-17: Timer_Mode Sample RPDL Integration

4.18 Flash_Data

4.18.1 Description

This sample demonstrates usage of the data flash memory, by writing the results of an ADC conversions to an incrementing data flash address every time a switch is pressed.

4.18.2 Operation

1. The sample initialises the LCD module, displays instructions on the screen and calls the function Init_FlashData.
2. The function Init_FlashData then configures the FCU unit and the data flash memory area followed by the initialisation of the 10-bit ADC. A function call to erase to erase the flash memory is called before the sample proceeds. The sample also configures the ADC unit, and switch interrupts.
3. The sample then waits in an infinite while loop for a user to press a switch.
4. When a user presses switch SW1, the sample triggers an ADC conversion and then writes the value into flash memory. The flash data and address it was written to is displayed on the debug LCD.
5. When the user presses switch SW1 again, the sample writes the updated ADC value to the next free memory location and displays the data and address on the debug LCD.
6. When the user presses switch SW3, the contents of the data flash block are erased, and the flash writes start from the beginning of the data flash block again.
7. To perform a new write (from the start of the data flash block), press SW1.

4.18.3 Sequence Diagram

Figure 4-18 below shows the program execution flow of the Flash_Data sample.

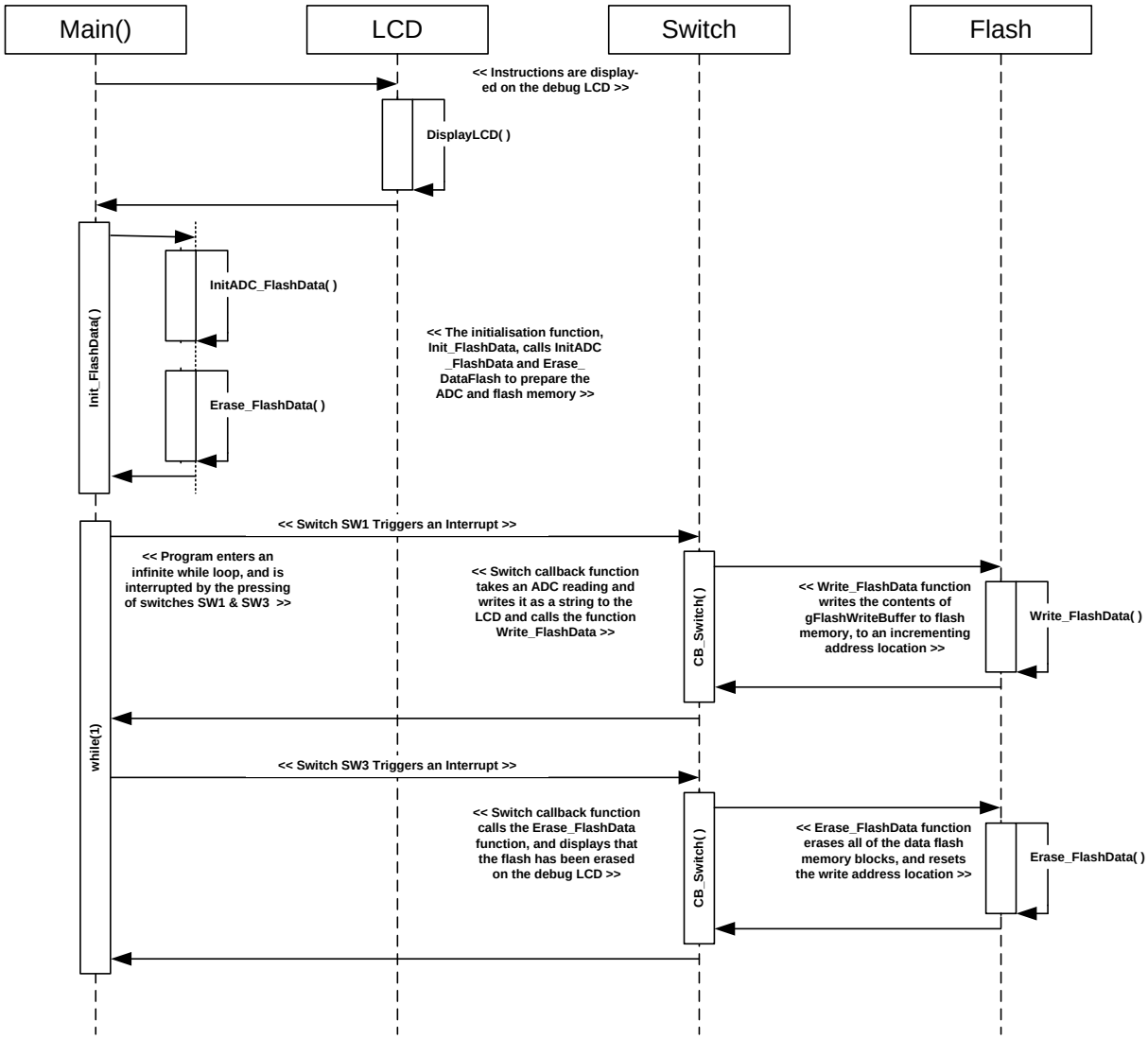


Figure 4-18: Flash_Data Sequence Diagram

4.18.4 RPDL Integration

Table 4-18 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
InitADC_FlashData	R_ADC_10_Create
CB_Switch	R_ADC_10_Control
	R_ADC_10_Read

Table 4-18: Timer_Compare Sample RPDL Integration

The sample makes extensive use of the Renesas RX600 Flash API, details of which can be found in the Simple Flash API for RX600 Application Note (R01AN0544EU-0200).

4.19 DTC

4.19.1 Description

This sample demonstrates usage of the DTC unit, by perform a DTC transfer of an ADC result to an incrementing location in an array when a switch is pressed.

4.19.2 Operation

1. The sample initialises the LCD module, and displays instructions on the screen.
2. The sample calls the Init_DTC function, which configures the DTC unit and also configures an ADC unit which triggers the DTC transfer after a successful conversion. The DTC transfer is configured to transfer the contents of the ADC result register to incrementing locations in the global array, gDTC_Destination.
3. The sample then enters an infinite while loop, with the rest of the sample's functionality completed in interrupts.
4. When the switch SW3 is pressed, the callback function CB_Switch is executed. The callback function checks the number of remaining transfers, and triggers an AD conversion. If there are no more remaining transfers, the function clears the contents of the gDestination array and reconfigures the DTC transfer to start from the beginning.

4.19.3 Sequence Diagram

Figure 4-19 below shows the program execution flow of the DTC sample.

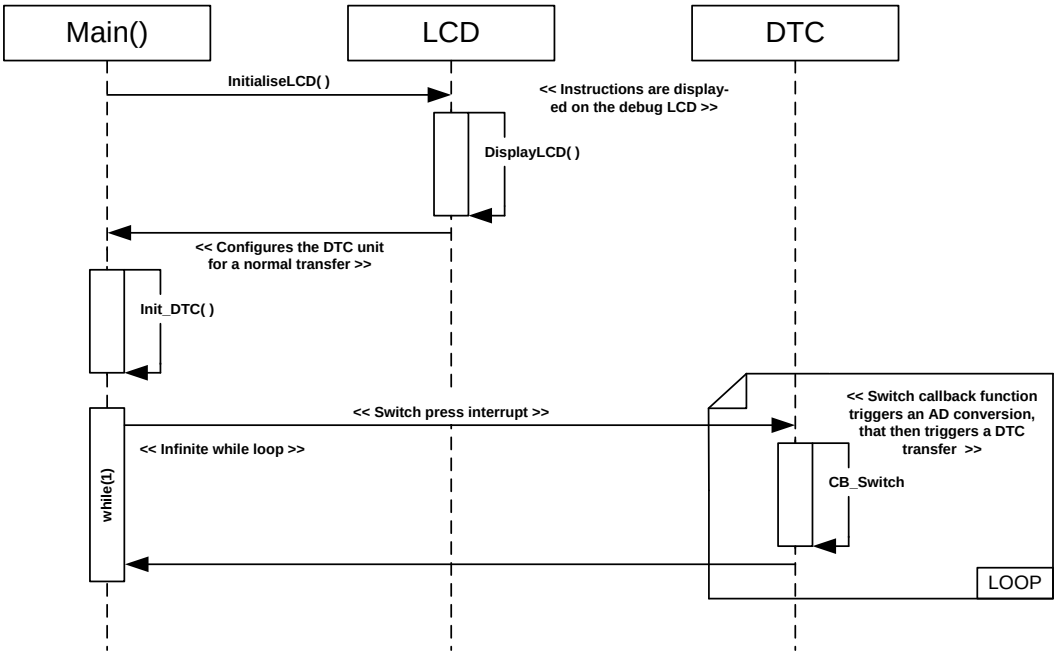


Figure 4-19: DTC Sequence Diagram

4.19.4 RPDL Integration

Table 4-19 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_DTC	R_DTC_Set
	R_DTC_Create
	R_ADC_10_Create
	R_DTC_Control
CB_Switch	R_DTC_GetStatus
	R_DTC_Control
	R_INTC_Write
	R_ADC_10_Control

Table 4-19: DTC Sample RPDL Integration

4.20 PWM

4.20.1 Description

This sample demonstrates usage of the general purpose timer (GPT) channels by outputting a varying duty cycle from 10% to 90% of the duty period. The output is observed on a dedicated timer I/O pin, MTIOC2A.

4.20.2 Operation

1. The sample initialises the LCD module, and displays instructions on the screen.
2. The sample then calls the Init_PWM function which configures the GPT channel 0's timer to output a pwm signal with a varying duty from 10% to 90%. Two compare registers, GTCCRA and GTCCRB, to respectively set the increasing duty and reset the duty to 10% once it has reached 90%. This is done using interrupts.
3. Pressing SW1 stops the duty from varying and the duty percentage is displayed on the debug LCD.

4.20.3 Sequence Diagram

Figure 4-20 below shows the program execution flow of the PWM sample.

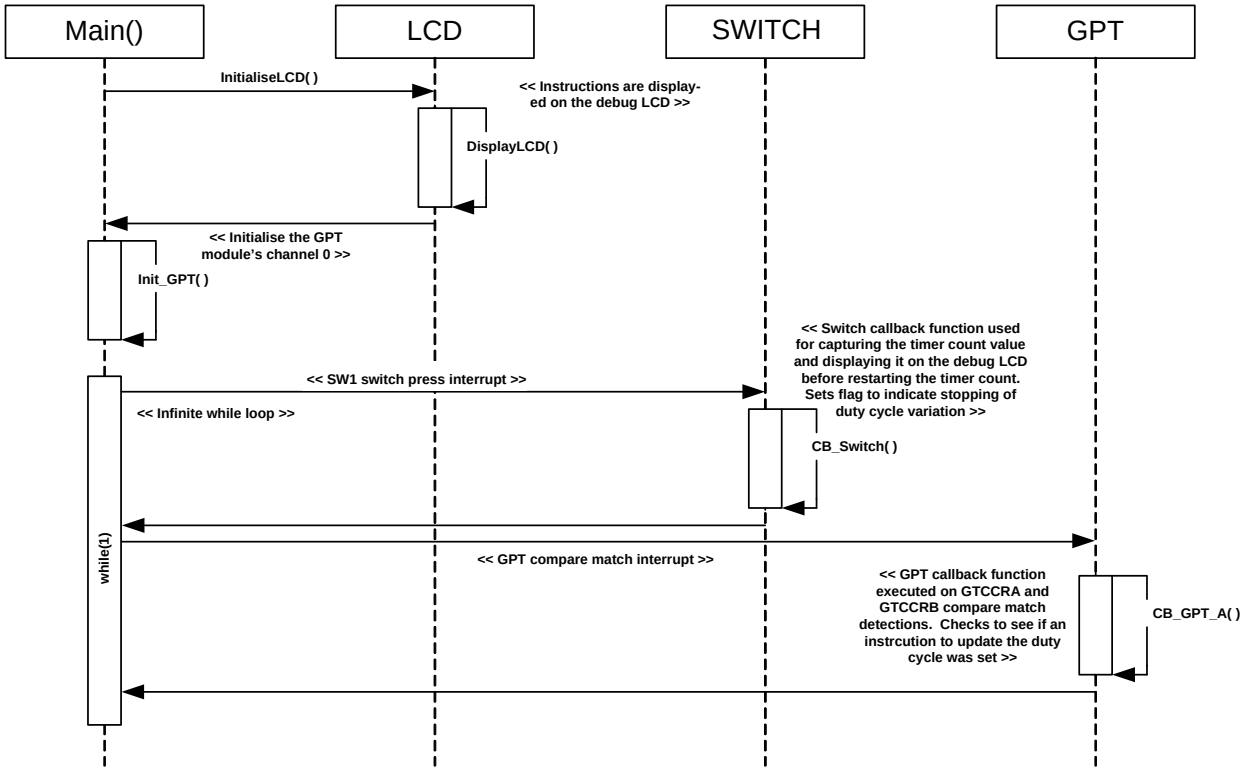


Figure 4-20: PWM Sequence Diagram

4.20.4 RPDL Integration

Table 4-20 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_PWM	R_GPT_Set
	R_GPT_Create
	R_GPT_ControlChannel
CB_GPT_A	R_GPT_ReadChannel
	R_GPT_ControlChannel
CB_Switch	R_GPT_ControlChannel

Table 4-20: PWM Sample RPDL Integration

4.21 WDT

4.21.1 Description

This sample demonstrates usage of the watch timer unit, by creating a timer overflow interrupt and a periodic CMT interrupt used to reset the timer. The period of the CMT interrupt is varied with the potentiometer, allowing it to be reduced until the WDT overflow interrupt occurs.

4.21.2 Operation

1. The sample initialises the LCD module, and displays instructions on the screen.
2. The sample then calls the Init_WDT function which configures the WDT and ADC unit, as well as the CMT to create a periodic interrupt. Watchdog timer overflow period is set to ~700ms, and is set to execute the callback function CB_WDTOverflow when the WDT overflows.
3. The sample then enters an infinite while loop. When the timer period elapses the callback function, CB_CMTPeriod, is executed.
4. The function CB_CMTPeriod resets the WDT count, toggles the user LEDs, and triggers an AD conversion. The function also updates the timer period with value set in the global variable gCMT_Period.
5. The callback function CB_ADConversion is executed when the AD conversion is complete. The function fetches the ADC result and uses it to calculate a new timer period which is stored in gCMT_Period.
6. When the timer period duration is greater than 700ms, the watchdog timer overflows generating an overflow interrupt.
7. The WDT overflow interrupt executes the function, CB_WDTOverflow, which sets the user LEDs to static ON and displays “Watchdog Overflow” on the debug LCD. The function then waits in an infinite while loop.

4.21.3 Sequence Diagram

Figure 4-21 below shows the program execution flow of the WDT sample.

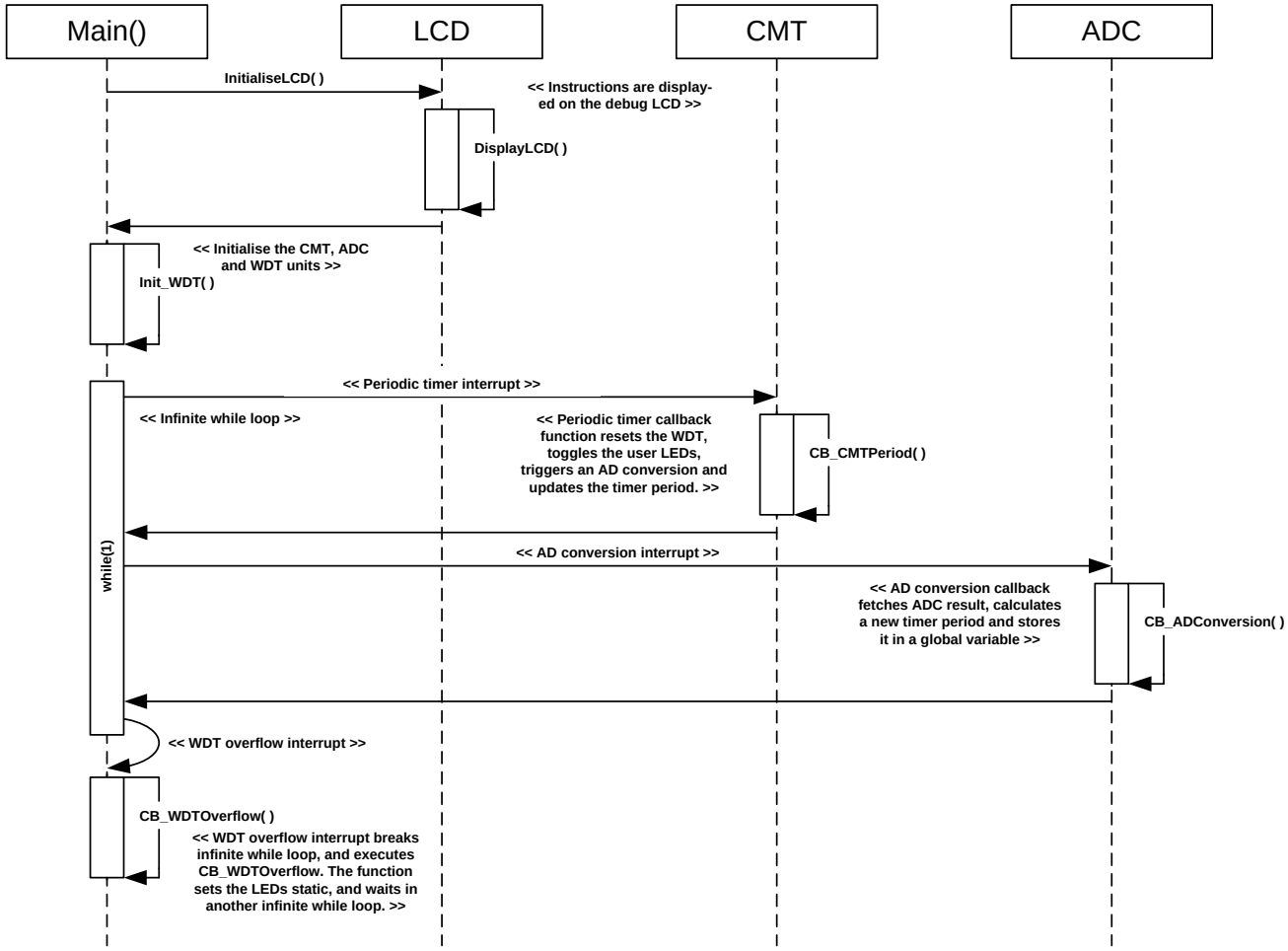


Figure 4-21: WDT Sequence Diagram

4.21.4 RPDL Integration

Table 4-21 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_WDT	R_ADC_10_Create
	R_WDT_Create
	R_CMT_Create
CB_CMTPeriod	R_WDT_Control
	R_IO_PORT_Modify
	R_ADC_10_Control
	R_CMT_Control
CB_ADConversion	R_ADC_10_Read
CB_WDTOverflow	R_IO_PORT_Write

Table 4-21: WDT Sample RPDL Integration

4.22 IWDT

4.22.1 Description

This sample demonstrates usage of the independent watchdog timer unit, by creating a timer underflow interrupt and a periodic CMT interrupt used to refresh the timer. The period of the CMT interrupt varies with the potentiometer, allowing it to be reduced until the IWDT underflow interrupt occurs.

4.22.2 Operation

1. The sample initialises the LCD module, and displays instructions on the screen.
2. The sample then calls the Init_IWDT function which checks to see if the last reset was due to the IWDT underflowing and halts the program if this is the case. Otherwise, it configures the IWDT and ADC unit, as well as the CMT to create a periodic interrupt. Watchdog timer underflow period is set to ~130ms, and it resets the microcontroller when the IWDT underflows.
3. The sample then enters an infinite while loop. When the timer period elapses the callback function, CB_CMTPeriod, is executed and updated with the ADC value multiplied by a factor.
4. The function CB_CMTPeriod refreshes the IWDT count, toggles the user LEDs, and triggers an AD conversion. The function also updates the timer period with value set in the global variable gCMT_Period.
5. The callback function CB_ADConversion is executed when the AD conversion is complete. The function fetches the ADC result and uses it to calculate a new timer period which is stored in gCMT_Period.
6. When the timer period duration is greater than ~130ms, the watchdog timer underflows generating a device reset.
7. The IWDT resets the device, turning all LEDs to static ON and displays “Watchdog Underflow” on the debug LCD. The function then waits in an infinite while loop.

4.22.3 Sequence Diagram

Figure 4-22 below shows the program execution flow of the IWDT sample.

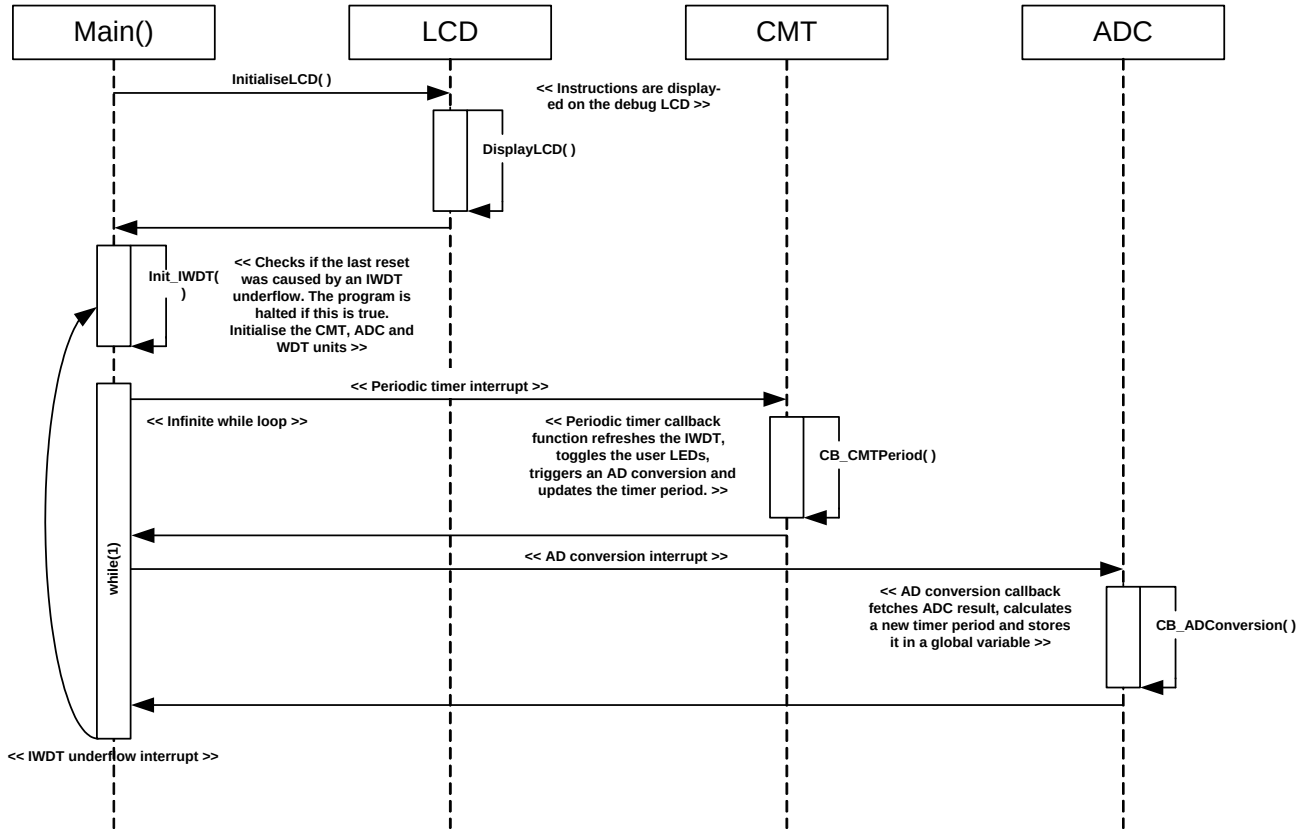


Figure 4-22: IWDT Sequence Diagram

4.22.4 RPDL Integration

Table 4-22 below details the RPDL functions used in each sample code function shown in the sequence diagram.

Function	RPDL Function
Init_IWDT	R_ADC_10_Create
	R_IWDT_Set
	R_INTC_CreateExtInterrupt
	R_CMT_Create
CB_CMTPeriod	R_WDT_Control
	R_IO_PORT_Modify
	R_ADC_10_Control
	R_CMT_Control
CB_ADConversion	R_ADC_10_Read

Table 4-22: IWDT Sample RPDL Integration

5. Additional Information

Technical Support
For details on how to use High-performance Embedded Workshop (HEW), refer to the HEW manual available on the CD or from the web site.

For information about the RX62G series microcontrollers refer to the RX62G Group hardware manual.

For information about the RX62G assembly language, refer to the RX600 Series Software Manual.

Technical Contact Details

***Please refer to the contact details listed in section 7 of the* "quick start guide"**

General information on Renesas Microcontrollers can be found on the Renesas website at:
http://www.renesas.com/

Trademarks
All brand or product names used in this manual are trademarks or registered trademarks of their respective companies or organisations.

REVISION HISTORY	RSKRX62G Software Manual

Rev.	Date	Description	
		Page	Summary
1.00	Sep 14, 2012	¾	First Edition issued

Renesas Starter Kit Software Manual

Publication Date: Rev.1.00 Sep 14, 2012

Published by: Renesas Electronics Corporation

Renesas Electronics Corporation

SALES OFFICES http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130
Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220
Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804
Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327
Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679
Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898
Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044
Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670
Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300
Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510
Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.3

RX62G Group

Renesas Electronics Corporation

R20UT2246EG0100