

Document information

Info	Content
Keywords	LPC1788, OEM-LPC1788
Abstract	Describe the examples for testing each peripheral functions of LPC177x_8x in LPC1780CMSIS Library package.



Revision history

Rev	Date	Description
02	20110602	Update descriptions for new testing boards
01	20110325	Initial version.

Contact information

For additional information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

This document describes the example code of LPC177x_8xCMSIS library package that used to test each peripheral function of LPC177x_8x.

- Its main description is for the basic information about the peripheral(s), and their area(s) that will be tested by the example(s).
- It shows the instructions to configure hardware (jumpers, connections...) in requirements for every example.
- It also points out the true steps to run examples properly. This will help the tests done more easily.

2. General information

2.1 Microcontrollers

LPC177x_8xCMSIS examples were written and tested with LPC177x_8x microcontroller. But they are also available with LPC177x_8x family chip.

Type number ^[1]	Flash (kB)	Total SRAM (kB)	EEPROM (kB)	Ethernet	USB	UART	Ex. Mem Bus ^[2]	LCD	QEI	DAC	I ² S	SD
LPC178x												
LPC1788	512	96 ^[5]	4	Y	H/O/D	5	32-bit/ 16-bit ^[3] / 8-bit ^[4]	Y	Y	Y	Y	Y
LPC1787	512	96 ^[5]	4	N	H/O/D	5	32-bit	Y	Y	Y	Y	Y
LPC1786	256	80 ^[6]	4	Y	H/O/D	5	32-bit	Y	Y	Y	Y	Y
LPC1785	256	80 ^[6]	4	N	H/O/D	5	32-bit	Y	N	Y	Y	Y
LPC177x												
LPC1778	512	96 ^[5]	4	Y	H/O/D	5	32-bit/ 16-bit ^[3] / 8-bit ^[4]	N	Y	Y	Y	Y
LPC1777	512	96 ^[5]	4	N	H/O/D	5	32-bit	N	Y	Y	Y	Y
LPC1776	256	80 ^[6]	4	Y	H/O/D	5	32-bit/ 16-bit ^[3]	N	Y	Y	Y	Y
LPC1774	128	40 ^[7]	2	N	D	5/4 ^[4]	32-bit/ 8-bit ^[4]	N	N	Y/N ^[4]	N	N
LPC1772	64	24 ^[8]	2	N	D	5/4 ^[4]	32-bit/ 8-bit ^[4]	N	N	Y/N ^[4]	N	N

Fig 1. Options for LPC178x/7x parts

2.2 Evaluation boards

LPC177x_8xCMSIS examples can be working well on:

- LPC1788 OEM board rev A connects with OEM base board rev A.
- LPC1788 IAR Start Kit Rev.B.

2.2.1 LPC1788 OEM board rev A connects with OEM base board rev A

This board is designed by Embedded Artist. Since, it should be called as OEM board later on in this document.

A captured picture of this board is as below

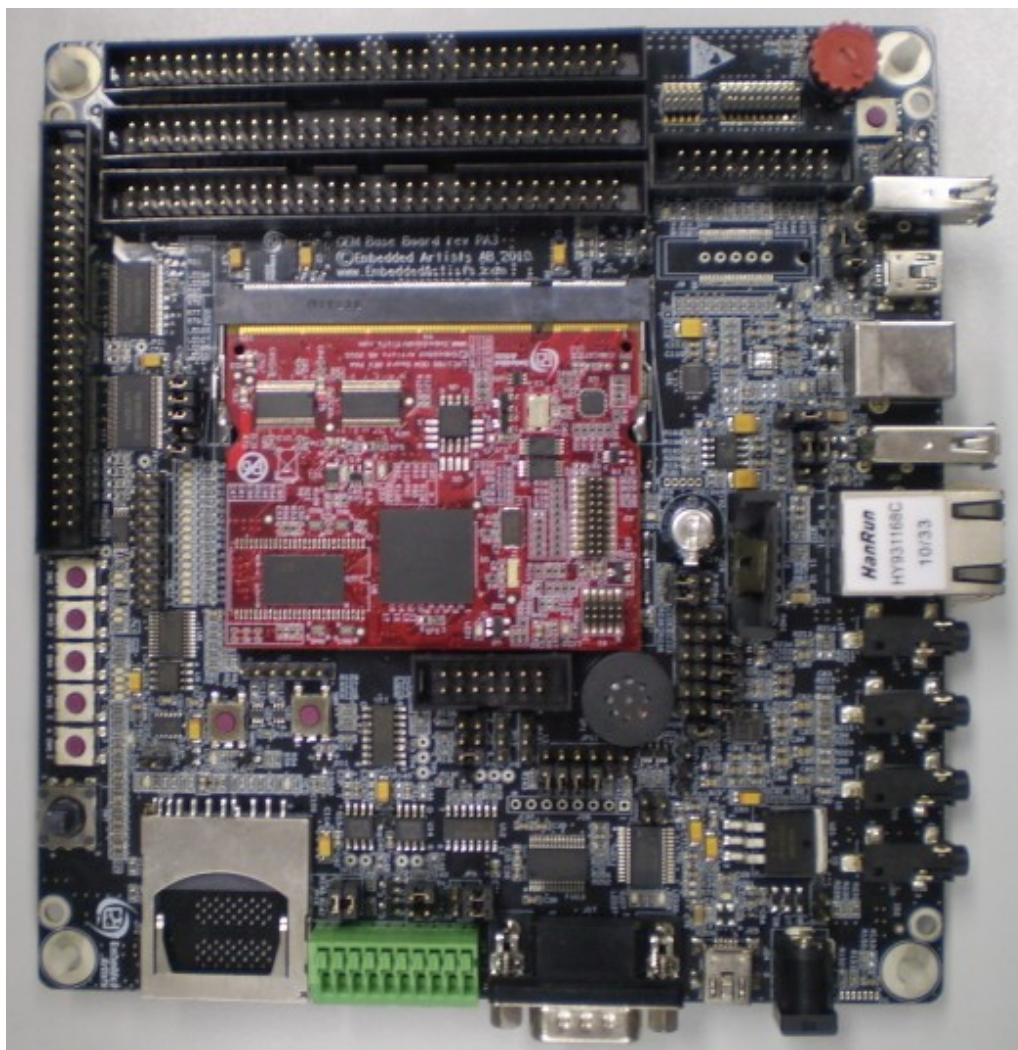


Fig 2. LPC1788 OEM board rev A is mounted OEM base board rev A

2.2.2 LPC1788 IAR Start Kit Rev.B

This board can be called as IAR board from now on in this document

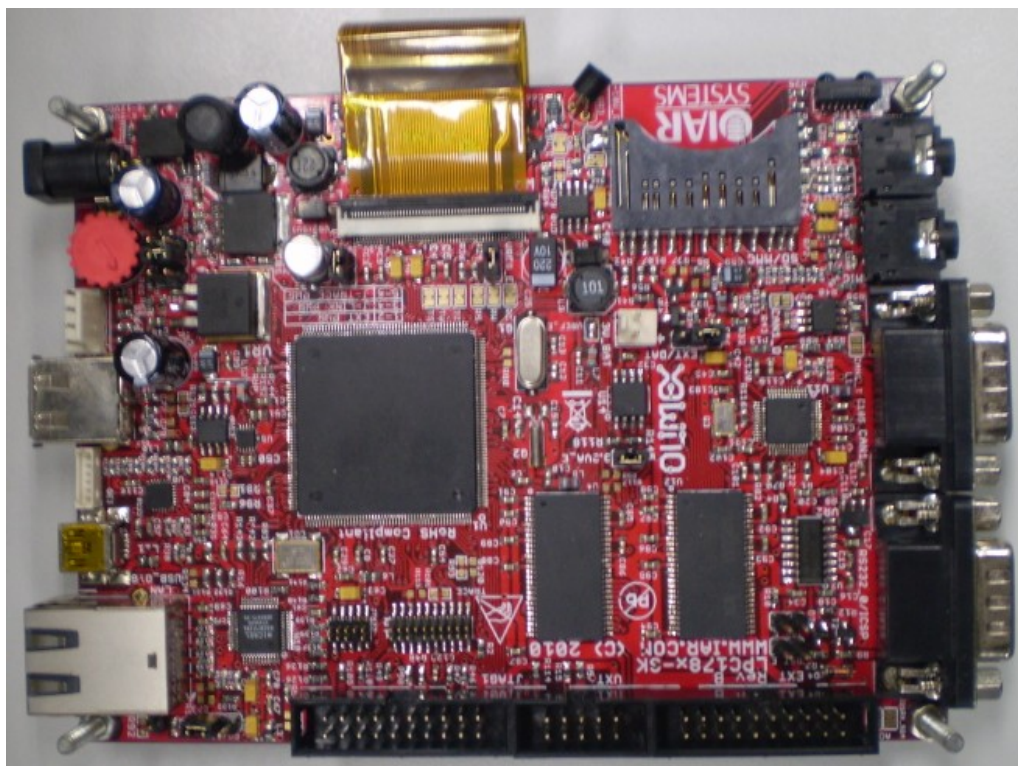


Fig 3. LPC1788 IAR Start Kit Rev B Board

2.2.3 Reference to the board that using to test by the code

Because these boards listed above (OEM board and IAR board) have differences about schematic, jumpers, pin-layout, and pins that ready for connection... it's required some changes in the code to apply these differences.

The examples will distinguish these board by using `#define _CURR_USING_BRD`. The assigned value for this definement is the number that represented for the board as below.

```
#define _EA_PA_BOARD                (2)
#define _IAR_OLIMEX_BOARD          (3)

#define CURR_USING_BRD              (_IAR_OLIMEX_BOARD)
```

Fig 4. Selecting the board that will be tested by the code in "bsp.h" file

2.3 Toolchains

LPC177x_8xCMSIS package supports these toolchains:

- ✓ Keil – RealView Microcontroller Development Kit (RVMDK) version 4.13
- ✓ IAR – Embedded Workbench for (ARM EWARM) version 5.5



Fig 5. IAR and Keil

2.4 Serial messages display

Almost examples allow displaying serial data or message via UART communication with a terminal on PC. In this case, UART component must be configured with the following condition.

The serial interface in a terminal on PC should be configured as below parameters:

- 115200 bps
- 8 data bit
- None parity
- 1 stop bit
- No flow control

2.4.1 UART handling for debug messages in the software:

❖ Files including:

In order to use serial communication to display data and messages, a call of `#include "debug_frmwrk.h"` is required in application code.

- `debug_frmwrk.h`: debug framework header file, placed at `..\Drivers\include`
- `debug_frmwrk.c`: debug framework source file, placed at `..\Drivers\source`

❖ On-chip UART settings

Choose expected UART port by setting `#define USED_UART_DEBUG_PORT` in `debug_frmwrk.h` header file. Its value is represented the UART number peripheral that is used to show debug messages.

```
#define DEBUG_FRMWRK_H_
#include "lpc177x_8x_uart.h"

#define USED_UART_DEBUG_PORT 0
```

Fig 6. Setting for UART in the code to show serial data on PC terminal

The serial parameters above (115200 bps, 8 data bits, none parity, 1 stop bit, no flow control) are setting in `debug_frmwrk_init()` function for corresponding UART

❖ Supported functions

Debug framework support display functions:

- `_DBG(x)`: display a string of characters
- `_DBG_(x)`: display s string of characters and move to new line
- `_DBC(x)`: display a character

- `_DBD(x)` : display a 8-bits decimal number
- `_DBD16(x)` : display a 16-bits decimal number
- `_DBD32(x)` : display a 32-bits decimal number
- `_DBH(x)` : display a 8-bits hex number
- `_DBH16(x)` : display a 16-bits hex number
- `_DBH32(x)` : display a 32-bits hex number
- `_DG()` : get character value from the UART
- `_DGV(option, numCh, val)` : returned value in `val` from UART with the base `option` (hex or dec) and number `numCh` of character(s) to be received

❖ Hardware configuration

➤ OEM Board (LPC1788 OEM Board rev A and OEM Base Board rev A)

- UART0: USB serial port
 - ✓ All jumpers: Default
- UART1:
 - ✓ P0.15 (@ J5.19) - JP12.2
 - ✓ P0.16 (@ J3.24) - JP13.2
- UART2:
 - ✓ JP6: 1-2: OFF
 - ✓ 3-4: OFF
 - ✓ 5-6: ON
 - ✓ JP12: 1-2
 - ✓ JP13: 1-2
 - ✓ Other jumpers: Default
- UART3:
 - ✓ P0.2 (@J5.13) - JP12.2
 - ✓ P0.3 (@J5.14) - JP13.2
- UART4:
 - ✓ P0.22 (@J3.25) - JP12.2
 - ✓ P2.9 (@J3.15) - JP13.2

➤ IAR Board (LPC1788 IAR Start Kit Rev.B)

Only UART0 is ready for using to connect to other sites as PC or other board.

- All jumpers: Default

2.4.2 Serial Terminal Program on PC:

COM Port on Windows PC should be configured as [Fig 7](#) picture of Hyper Terminal COM Properties with all parameters mentioned above.

It's advised that the Hyper Terminal should not be used on Windows in order to avoid the error shown in [Fig 8](#) if the data content to display is larger than the space of hyper terminal window (white space), otherwise some characters will be lost (this is not an error of the UART driver).

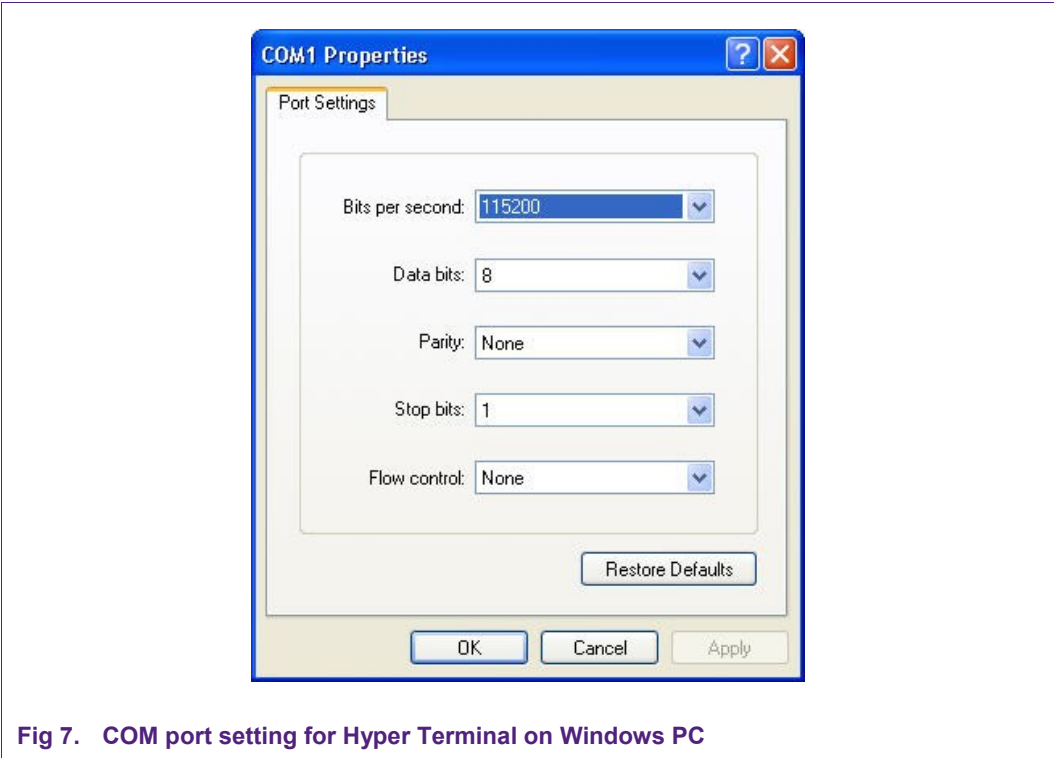


Fig 7. COM port setting for Hyper Terminal on Windows PC

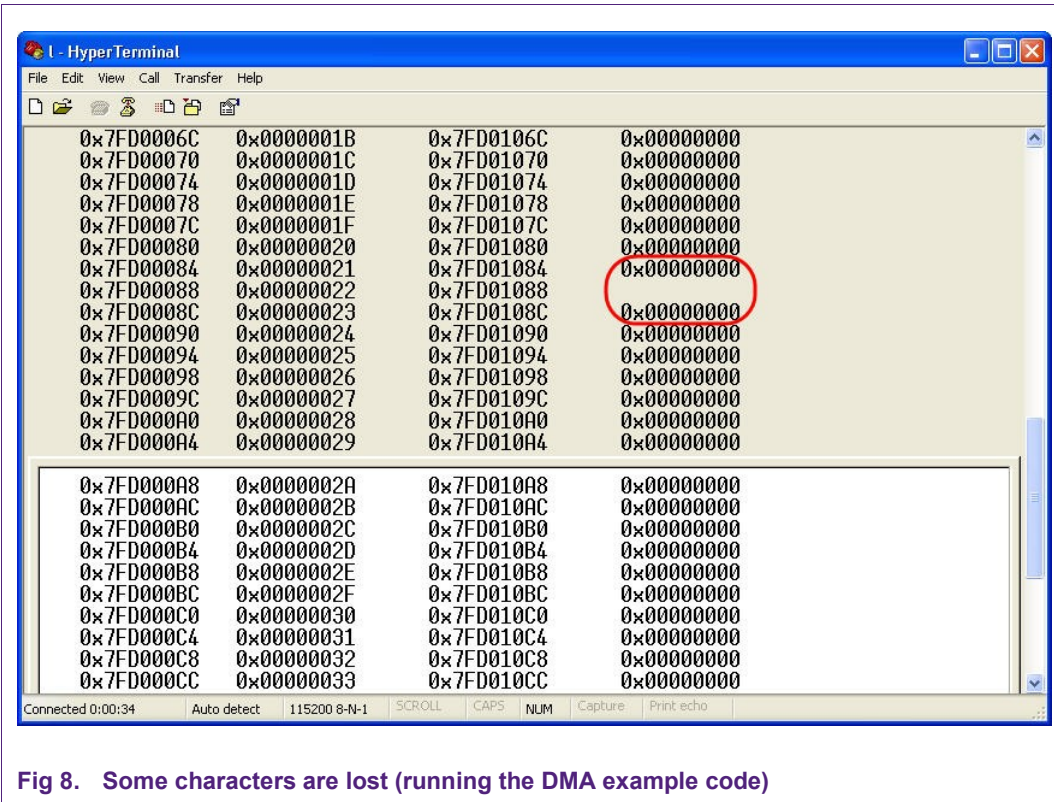


Fig 8. Some characters are lost (running the DMA example code)

The solution for this problem is other communication PC software, such as Flash magic, or TeraTerm.

Note: With Flash Magic, configure newlines: CR

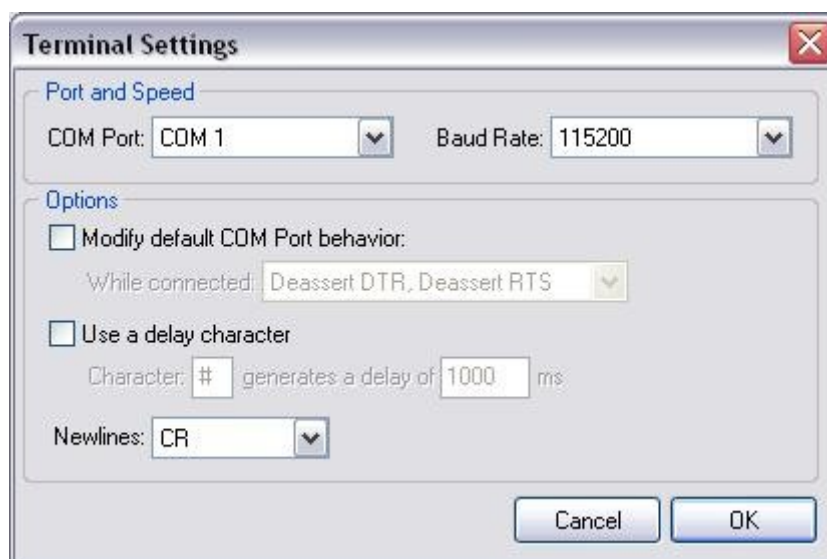


Fig 9. COM configuration on Flash Magic

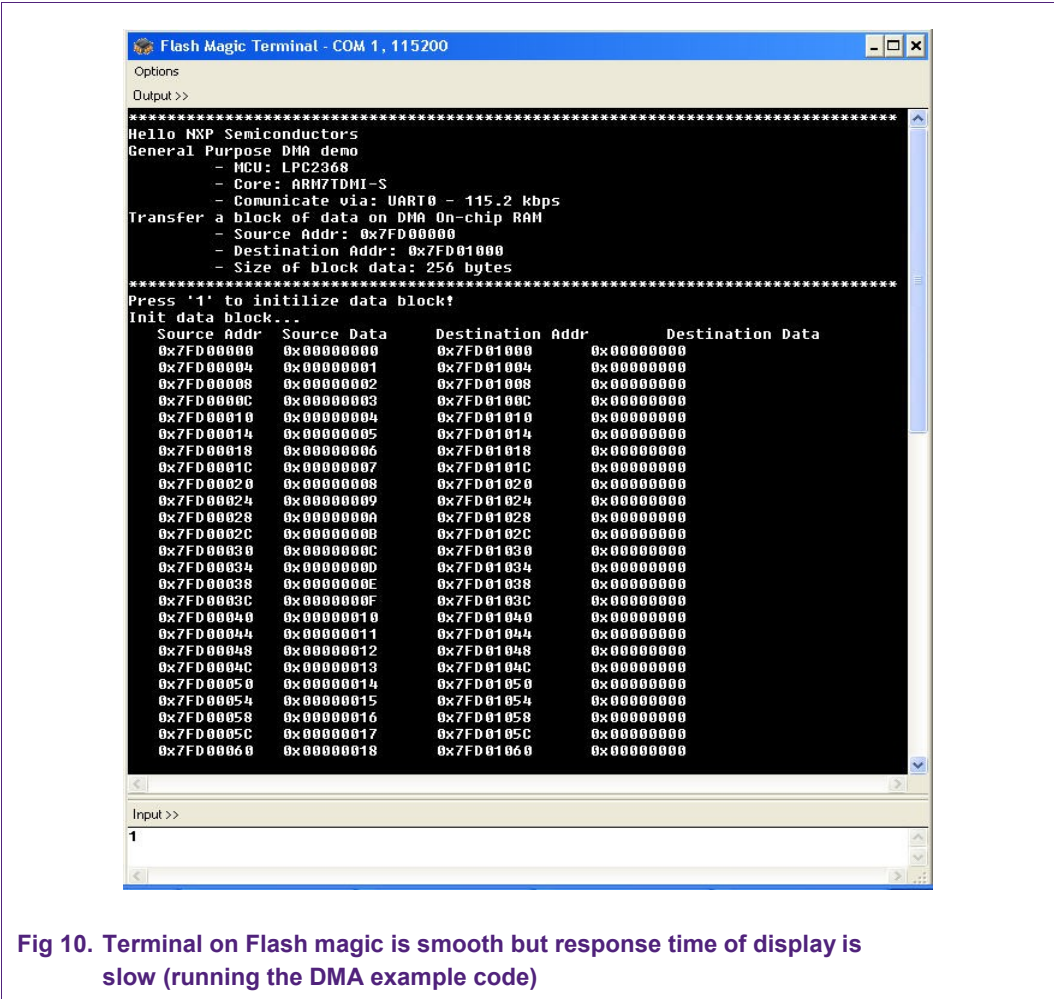


Fig 10. Terminal on Flash magic is smooth but response time of display is slow (running the DMA example code)

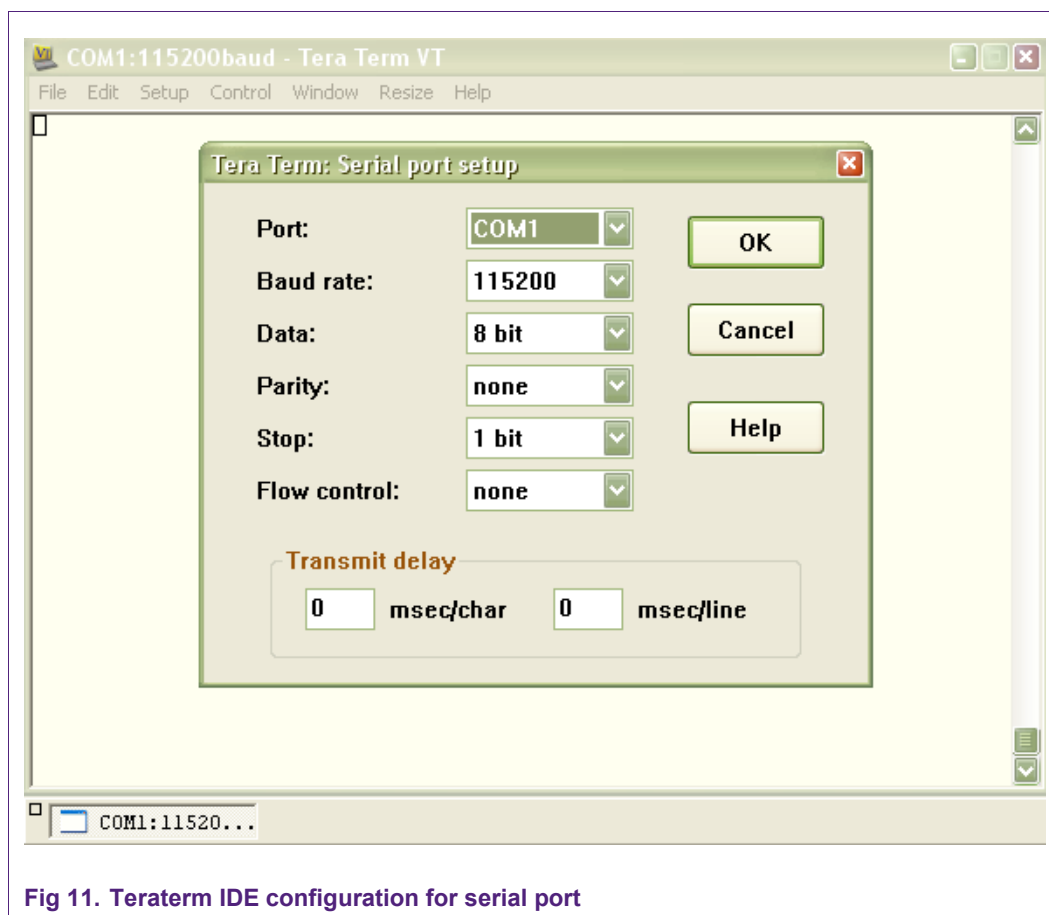


Fig 11. Teraterm IDE configuration for serial port

2.5 Initial working condition

2.5.1 Power the board:

It's advised to check the below hardware part for correct power on the board

➤ LPC1788 OEM board rev A connects with OEM base board rev A

- None

➤ LPC1788 IAR Start Kit Rev.B

- 3.3VA_E: ON
- VREG: ON
- IO_E: ON
- PWR_SEL: depends on power source selection

2.6 Running mode

Each example can be run in RAM mode, ROM (Flash) mode or both. Please reference 'Running mode' section in the examples' description to know which mode that example can run.

❖ Burn built image to chip by using Flash Magic tool

According to the board that currently using, these jumpers' positions must be taken care:

- **OEM Board (LPC1788 OEM Board rev A and OEM Base Board rev A):**
 - At **JP20** jumpers:
 - **Link 1-2: ON**
 - **Link 3-4: ON**

Because UART0 of the chip is used with USB connection to PC, if using this USB serial port to flash the code, it needs to ensure choosing correct COM port number on Windows PC. Reference for this COM number is:

“Device Manager” -> “Ports(COM & LPT)” -> “USB Serial Port (COMx)”

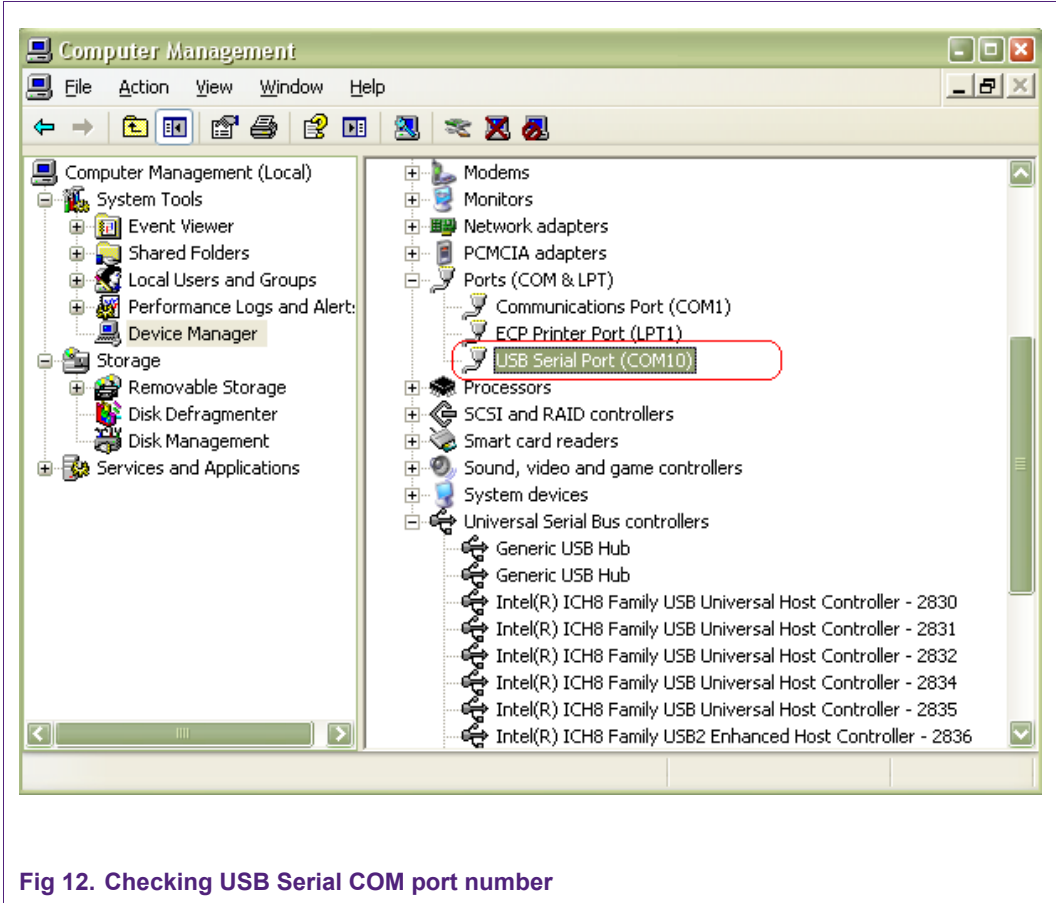


Fig 12. Checking USB Serial COM port number

- **IAR Board (LPC1788 IAR Start Kit Rev.B):**
 - Jumper **RST_E**: **ON**
 - Jumper **ISP_E**: **ON**

2.7 Flag definitions

This table lists flags that used in LPC1780CMSIS Lib

Flag	Value	Description
_EA_PA_BOARD	2	Defined for using LPC1788 OEM Board rev A and OEM Base Board rev A

<code>_IAR_OLIMEX_BOARD</code>	3	Defined for using LPC1788 IAR Start Kit Rev.B
<code>_CURR_USING_BRD</code>		Assigned to above Flags (<code>_EA_PA_BOARD</code> or <code>_IAR_OLIMEX_BOARD</code>) for the board currently using with the code
<code>__RAM_MODE__</code>		Defined for running in RAM mode Not defined when running in ROM/Flash mode

3. Examples

3.1 ADC (ANALOG – TO – DIGITAL CONVERTER)

3.1.1 Adc_Polling

3.1.1.1 Example description

Purpose

This example describes how to use ADC conversion in polling mode.

Knowledge and Process

ADC is configured to convert signal input from channel 2 (if on OEM board) or from channel 7 (if on IAR board).

The ADC conversion rate is 400KHz. A fully accurate conversion requires 31 of these clocks.

So ADC clock = 400KHz * 31 = 12.4MHz.

After starting ADC operation, it's continuously polling "DONE" bit in ADC Data Register `AD0DRx`. If this "DONE" bit is set means the conversion is completed, ADC converted result is saved to `adc_value` variable and display this data via serial interface and restart ADC for next conversion.

Turn potentiometer to change ADC signal input.

3.1.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Adc_Polling.c: Main program file

3.1.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

3.1.1.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.1.1.5 Steps to run

- (1) Choose correctly the working board by setting `#define _CURR_USING_BRD` in header file `.\.\.\BoardSupport\bsp.h`
- (2) Build example
- (3) Burn built image file (in hex) into the board (in case of ROM mode running)
- (4) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (5) Do hardware configuration exactly following the above instructions.
- (6) Reset the board (in case of ROM mode running) and Run the example on the working board
- (7) Testing actions and results:
 - Turn the potentiometer left or right
 - The content on PC serial terminal should be like this

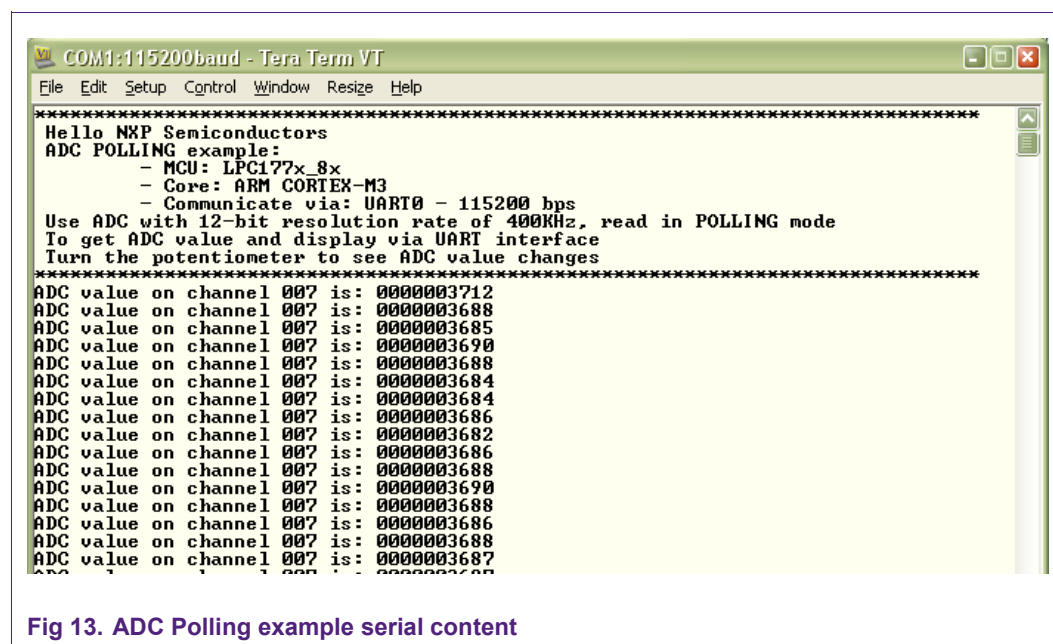


Fig 13. ADC Polling example serial content

3.1.2 Adc_Interrupt

3.1.2.1 Example description

Purpose

This example describes how to use ADC conversion in interrupt mode.

Knowledge and Process

ADC is configured to convert signal input from channel 2 (if on OEM board) or from channel 7 (if on IAR board).

The ADC conversion rate is 400KHz. A fully accurate conversion requires 31 of these clocks.

So ADC clock = 400KHz * 31 = 12.4MHz.

ADC will generate an interrupt once current ADC conversion is ended. ADC Interrupt Service Routine will be invoked to check ADC status, if "DONE" bit in ADC Data Register AD0DRx is set, ADC converted data will be stored in `adc_value`.

ADC interrupt is disabled and re-enabled for the next conversion.

ADC converted data are displayed by serial communication with PC.

Turn potentiometer to change ADC signal input.

3.1.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Adc_Interrupt.c: Main program file

3.1.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

3.1.2.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.1.2.5 Steps to run

- (1) Choose correctly the working board by setting `#define _CURR_USING_BRD` in header file `.\.\.\BoardSupport\bsp.h`
- (2) Build example
- (3) Burn built image file (in hex) into the board (in case of ROM mode running)
- (4) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (5) Do hardware configuration exactly following the above instructions.
- (6) Reset the board (in case of ROM mode running) and Run the example on the working board
- (7) Testing actions and results:
 - Turn the potentiometer left or right
 - The content on PC serial terminal should be like this

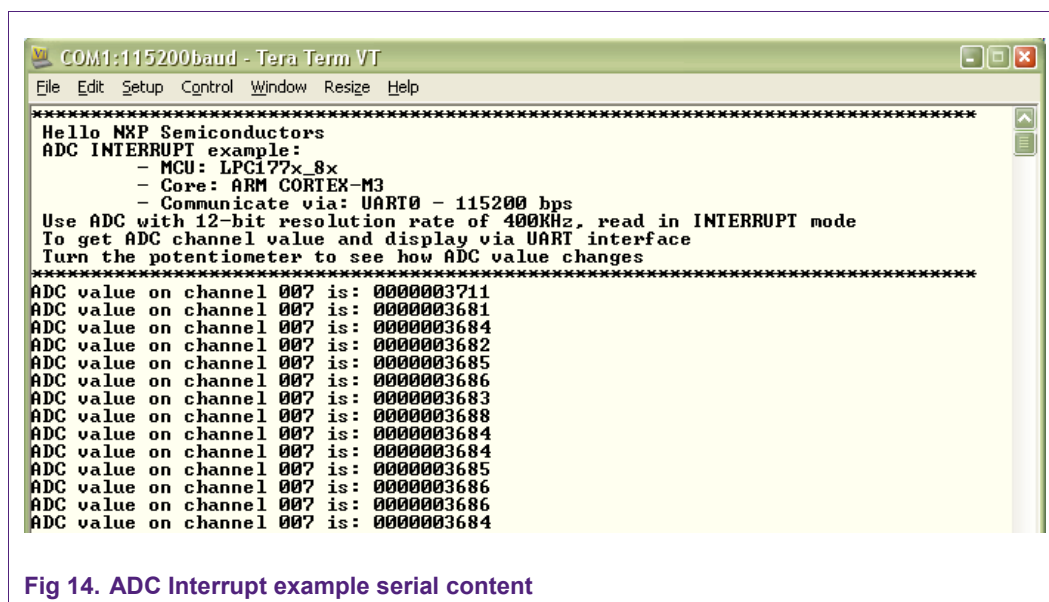


Fig 14. ADC Interrupt example serial content

3.1.3 Adc_Dma

3.1.3.1 Example description

Purpose

This example illustrates how to use ADC converter and transfer converted results by using DMA.

Knowledge and Process

ADC is configured to convert signal input from channel 2 (if on OEM board) or from channel 7 (if on IAR board).

The ADC conversion rate is 400KHz. A fully accurate conversion requires 31 of these clocks.

So ADC clock = 400KHz * 31 = 12.4MHz.

ADC will generate interrupt at the end of each conversion. It will make a request for DMA source to transfer the converted data from ADC data register `AD0DR` to `adc_value` variable.

After sending this converted result to PC terminal via serial communication and display, the DMA component will re-setup for next transfer.

Turn potentiometer to change ADC signal input.

3.1.3.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Adc_Dma.c: Main program file

3.1.3.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

3.1.3.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.1.3.5 Steps to run

- (1) Choose correctly the working board by setting `#define _CURR_USING_BRD` in header file `.\.\.\BoardSupport\bsp.h`
- (2) Build example
- (3) Burn built image file (in hex) into the board (in case of ROM mode running)
- (4) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (5) Do hardware configuration exactly following the above instructions.
- (6) Reset the board (in case of ROM mode running) and Run the example on the working board
- (7) Testing actions and results:
 - Turn the potentiometer left or right
 - The content on PC serial terminal should be like this

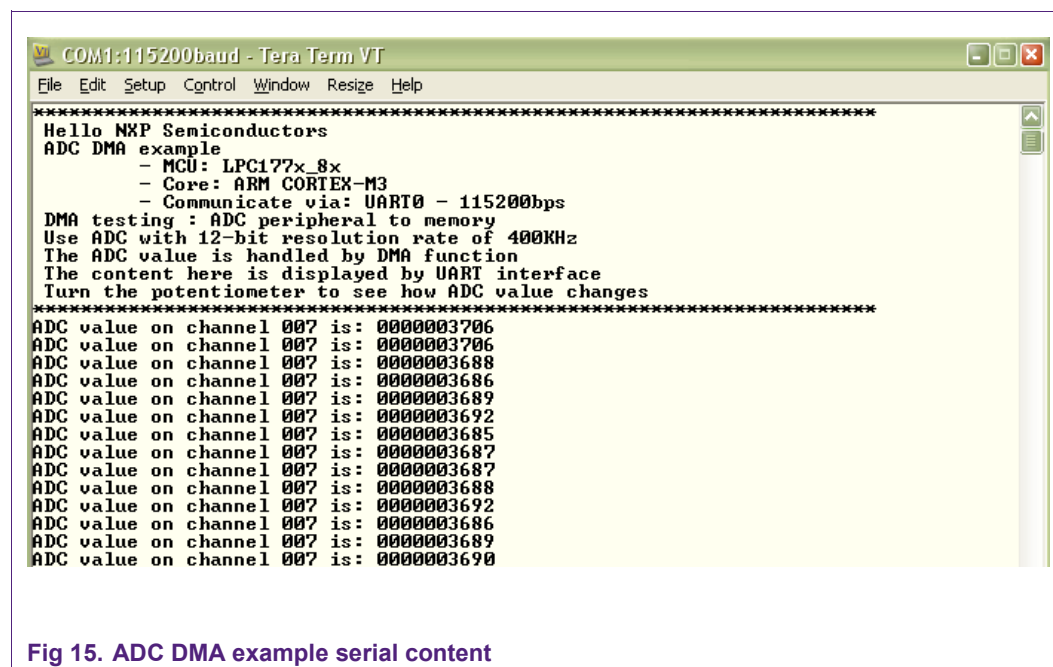


Fig 15. ADC DMA example serial content

3.1.4 Adc_Burst

3.1.4.1 Example description

Purpose

This example describes how to use ADC component in burst mode with single or multiple inputs and how to inject an ADC conversion channel while the others are running.

Knowledge and Process

The ADC conversion rate is 400 KHz. A fully accurate conversion requires 31 of these clocks.

So ADC clock = 400KHz * 31 = 12.4MHz.

Note that maximum ADC clock input is 12.4MHz.

Burst ADC will repeatedly sample-convert the voltage on ADC0.x pin(s) then update the channel data register(s). So we just need to read the ADC channel data register(s) and display the value via serial connection.

Only in burst mode, ADC Converter can sample multiple channels, then we can inject by configuring this mode.

Turn potentiometer to change ADC signal input.

In this example, other ADC channel is used is ADC channel 3.

This example uses external interrupt EXTINT0 on P2.10 to trigger the injection function, and uses LED P0.13 (LED USB Host) to toggle the LED indicating that an ADC channel has just been injected or removed.

- LED on: ADC channel 3 is activated.
- LED off: ADC channel 3 is inactivated.

Generation an external interrupt:

➤ LPC1788 OEM Board rev A and OEM Base Board rev A:

- ✓ Press **SW6** button

➤ LPC1788 IAR Start Kit Rev.B

- ✓ Connect **ISP_E[2]** with **GND**

3.1.4.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Adc_Burst.c: Main program file

3.1.4.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

ADC hardware connection

▪ For multiple inputs:

- The ADCx sampling pins must not left floating, otherwise, the read-out value of these pins will same as the value of the pin which connected to a real voltage.
- This example uses an addition ADCx pin: AD0.3 (P0.26).
- Try to make an external 10K, 3 pins vari-resistor, 1 terminal connects to 3.3V or Vrefp, the other connects to GND, and the rest (middle pin) connects to P0.26

- **For injection test:**

- LED jumper need to be ON to blinking LED P0.13.

3.1.4.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.1.4.5 Steps to run

- (1) Choose correctly the working board by setting `#define _CURR_USING_BRD` in header file `.\.\.\BoardSupport\bsp.h`

Testing purpose(s): do the changes in `Adc_Burst.c` file

- If test multiple input: Do uncomment `#define LPC177x_8x_ADC_BURST_MULTI` as below:

```
///#define LPC177x_8x_ADC_INJECT_TEST  
#define LPC177x_8x_ADC_BURST_MULTI
```

Code portion for testing ADC Multiple Inputs

- If test injection: Do uncomment `#define LPC177x_8x_ADC_INJECT_TEST` as below:

```
#define LPC177x_8x_ADC_INJECT_TEST  
///#define LPC177x_8x_ADC_BURST_MULTI
```

Code portion for testing ADC Injection

- (2) Build example
- (3) Burn built image file (in hex) into the board (in case of ROM mode running)
- (4) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (5) Do hardware configuration exactly following the above instructions.
- (6) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (7) Testing actions and results:
 - Turn the potentiometer left or right
 - If in injection test, it's need press a button to generate an external interrupt (see [above](#) to understand about this), accordingly to the state of LED *P0.13* (LED UST Host), it's known:
 - *P0.13* is ON → ADC channel 3 has just been included
 - *P0.13* is OFF → ADC channel 3 has just been excluded
 - On PC terminal, it keeps update the messages from the board. And in case of injection test,
 - If *P0.13* is ON (ADC channel 3 inclusion), channel 3 value is updated

- If P0.13 is OFF (ADC channel 3 exclusion), channel 3 value is unchanged from the last update
- The content on the serial terminal should be like this

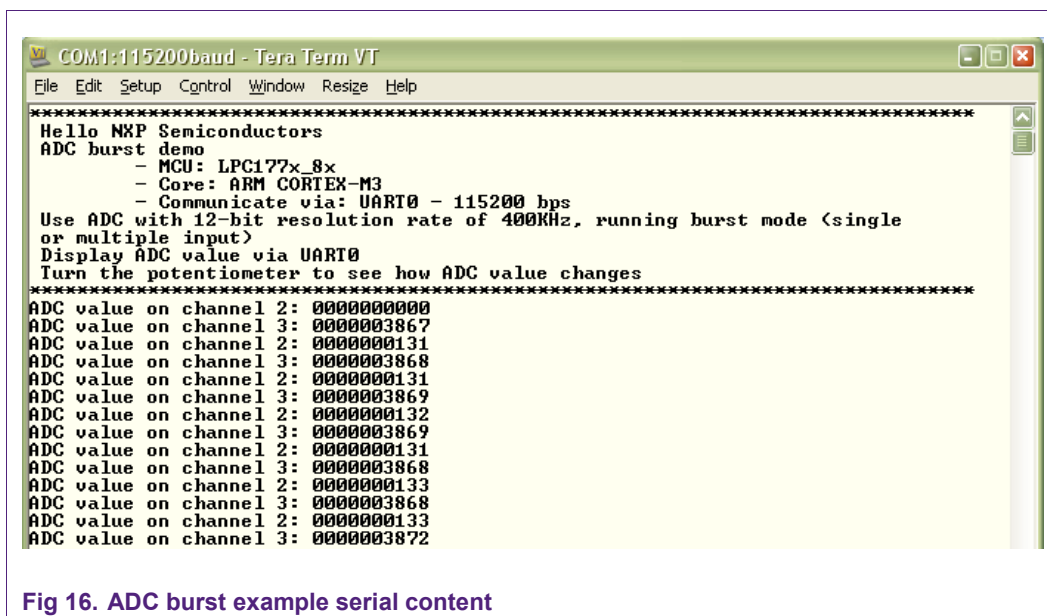


Fig 16. ADC burst example serial content

3.2 CAN (CONTROLLER AREA NETWORK)

3.2.1 Can_Selftest

3.2.1.1 Example description

Purpose

This example describes how to test CAN self-test mode

Knowledge and Process

Self-test mode implemented in this example is local self-test. It fits for single node tests.

In Self-test mode, the transmitted message is also received and stored in the received buffer. Bypass mode is enabled, so this received is legal.

After receive message, it will be compared with transmitted message:

- If they are same, self-test mode is successful.
- If not, self-test mode is fail.

Please observe process via serial display.

3.2.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Can_Selftest.c: Main program file

3.2.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

This example does not require any connection on hardware.

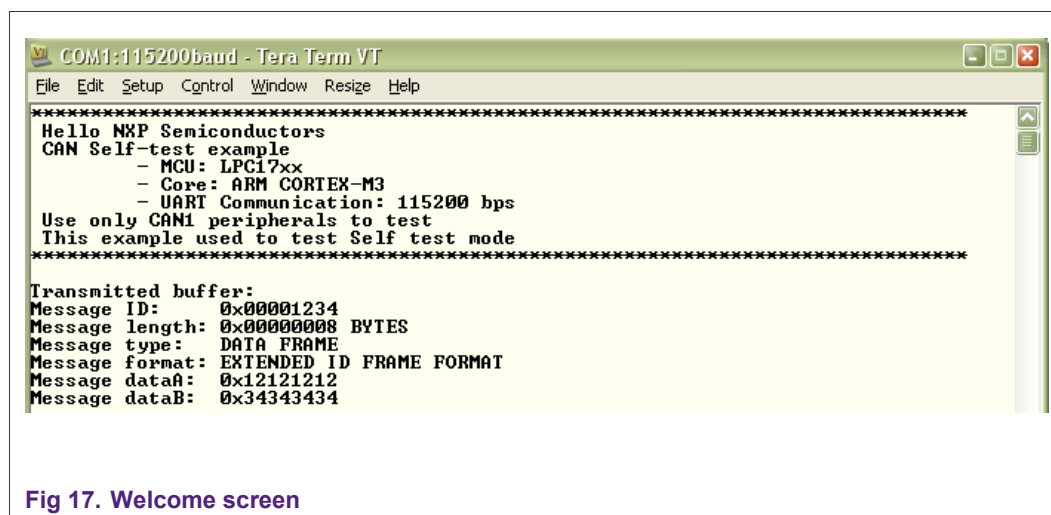
3.2.1.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.2.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board. *Make sure that the OEM board is currently using unless its running is not ok.*
- (6) Testing actions and results:
 - After reset, welcome screen appears like this:



3.2.2 Can_Bypass

3.2.2.1 Example description

Purpose

This example describes how to test CAN operation by using bypass mode

Knowledge and Process

Using 2 CAN peripheral CAN1 and CAN2 in same eval board to test CAN operation.

This example just supports extended ID format.

Both CAN1 and CAN2 are set baudrate at 125KHz.

In bypass mode, AFULT will be disabled, all messages could be received.

One transmit message is initialized with ID = 0x00001234 and data = 0x00.

CAN1 will send this message to CAN2

Whenever CAN2 receive message, CAN interrupt service routine `CAN_IRQHandler` will be invoked to receive message, print message's data into serial display via UART0 port, increase message's ID and data for next transmission by CAN1. This process is an endless loop.

Open serial display to observe CAN transfer processing.

3.2.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Can_Bypass.c: Main program file

3.2.2.3 Hardware configuration

This example just runs on OEM board only.

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

These jumpers must be configured as follows:

➤ LPC1788 OEM board rev A connects with OEM base board rev A

- ✓ **TP6** connects to P0.4-RD2 (**J5.16**)
- ✓ **TP8** connects to P0.5-TD2 (**J5.15**)
- ✓ At **JP14: Link 1-3: ON**
- ✓ At **JP14: Link 2-4: ON**

3.2.2.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.2.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board. *Make sure that the OEM board is currently using unless its running is not ok.*
- (6) Testing actions and results:
 - Press "1" to initialize sending messages
 - Press "2" to start CAN operation
 - The content in PC serial terminal after all running:

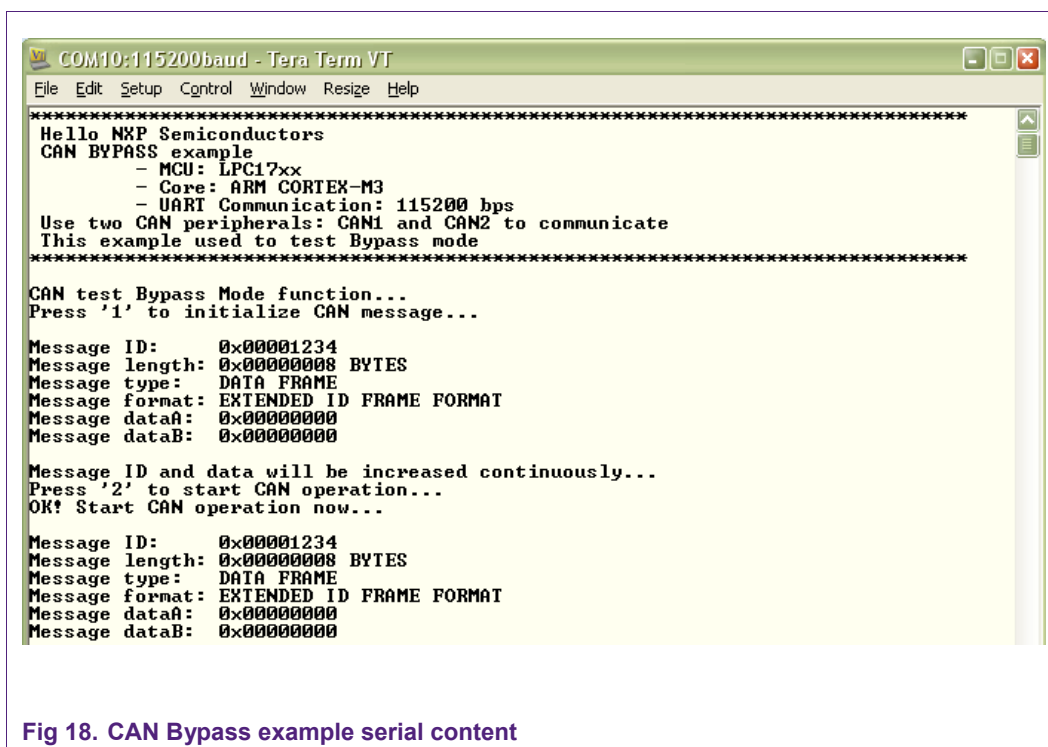


Fig 18. CAN Bypass example serial content

3.2.3 Can_Aflut

3.2.3.1 Example description

Purpose

This example describes how to use CAN driver functions for setup and change AFLUT (Acceptance Filter Look – Up Table) table dynamically.

Knowledge and Process

Using 2 CAN peripheral CAN1 and CAN2 to test CAN operation.

This example supports all kind of identifier: FullCAN, explicit or group format.

Both CAN1 and CAN2 are set baudrate at 125KHz.

First, setup AF look-up table with 5 sections:

- 6 entries for FullCAN section
- 6 entries for Standard Frame Format (SFF) section
- 6 entries for Group Standard Frame Format (SFF_GRP) section
- 6 entries for Extended Frame Format (EFF) section
- 6 entries for Group Extended Frame Format (EFF_GRP) section

Initialize 10 messages:

- 1st message with 11-bit ID which exit in AF Look-up Table in FullCAN Section
- 2nd message with 11-bit ID which not exit in AF Look-up Table
- 3th message with 11-bit ID which exit in AF Look-up Table in SFF Section
- 4th message with 11-bit ID which not exit in AF Look-up Table
- 5th message with 11-bit ID which exit in AF Look-up Table in Group SFF Section

- 6th message with 11-bit ID which not exit in AF Look-up Table
- 7th message with 29-bit ID which exit in AF Look-up Table in EFF Section
- 8th message with 29-bit ID which not exit in AF Look-up Table
- 9th message with 29-bit ID which exit in AF Look-up Table in Group of EFF Section
- 10th message with 29-bit ID which not exit in AF Look-up Table

Then, send 10 messages from CAN1 to CAN2, whenever CAN2 receive message that has ID exit in its AFLUT, CAN receive interrupt occurs, CAN interrupt service routine `CAN_IRQHandler()` will be invoked to receive message and save it in array `"AFRxBuf[]"`.

In this case, message 1, 3, 5, 7, 9 will be received.

After that, `CAN_ChangeAFTable()` function will be called to load and remove entries in AFLUT in such a way as to receive messages 2, 4, 6, 8, 10 instead of 1, 3, 5, 7, 9.

Re-send 10 messages and re-received messages to check if AFLUT operation is correct or not.

Open serial display window to observe CAN transfer processing.

3.2.3.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Can_Aflut.c: Main program file

3.2.3.3 Hardware configuration

This example just run on OEM board only.

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

These jumpers must be configured as follows:

➤ **LPC1788 OEM board rev A connects with OEM base board rev A**

- ✓ **TP6** connects to P0.4-RD2 (**J5.16**)
- ✓ **TP8** connects to P0.5-TD2 (**J5.15**)
- ✓ At **JP14: Link 1-3: ON**
- ✓ At **JP14: Link 2-4: ON**

3.2.3.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.2.3.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.

- (5) Reset the board (in case of ROM mode running) and Run the example on the working board. *Make sure that the OEM board is currently using unless its running is not ok.*
- (6) Testing actions and results:
- Press "1" to initialize message and AFLUT
 - Press "2" to start CAN operation

```
COM10:115200baud - Tera Term VT
File Edit Setup Control Window Resize Help
*****
Hello NXP Semiconductors
CAN AFLUT example:
  - MCU: LPC17xx
  - Core: ARM CORTEX-M3
  - UART Communication: 115200 bps
Use 2 CAN peripherals: CAN1&CAN2 to transfer data
This example tests full Acceptance Filter operation
and load/remove AFLUT entry dynamically functions
*****
Test Acceptance Filter function...
Press '1' to initialize message and AF Loop-up Table...

Init message finished!!!
Setup AF: SUCCESSFUL!!!

Press '2' to start CAN transferring operation...

Message ID:      0x00000001
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  STANDARD ID FRAME FORMAT
Message dataA:   0x78787878
Message dataB:   0x21212121

Message ID:      0x00000004
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  STANDARD ID FRAME FORMAT
Message dataA:   0x23232323
Message dataB:   0x45454545
```

Fig 19. After pressing '2' for transferring data

- Press "3" to display received messages

```
Message ID:      0x00005001
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  EXTENDED ID FRAME FORMAT
Message dataA:   0x85858585
Message dataB:   0x27272727

Sending finished !!!

Press '3' to display received messages...

Message ID:      0x00000001
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  STANDARD ID FRAME FORMAT
Message dataA:   0x78787878
Message dataB:   0x21212121

Message ID:      0x00000008
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  STANDARD ID FRAME FORMAT
Message dataA:   0x15151515
Message dataB:   0x36363636

Message ID:      0x00000015
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  STANDARD ID FRAME FORMAT
Message dataA:   0x65656565
Message dataB:   0x37373737
```

Fig 20. Receiving messages after pressing '3'

- Press "4" to change AFLUT
- Press "5" to re-send message

```
Message ID:      0x00004801
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  EXTENDED ID FRAME FORMAT
Message dataA:   0x52525252
Message dataB:   0x06060606

Press '4' to change AF look-up table...

Change AFLUT: FINISHED!!!
Press '5' to re-send messages...

Message ID:      0x00000001
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  STANDARD ID FRAME FORMAT
Message dataA:   0x78787878
Message dataB:   0x21212121

Message ID:      0x00000004
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  STANDARD ID FRAME FORMAT
Message dataA:   0x23232323
Message dataB:   0x45454545
```

Fig 21. Re-send messages after modifying the AF look-up table

- Press "6" to display received messages


```

Message ID:      0x00005001
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  EXTENDED ID FRAME FORMAT
Message dataA:   0x85858585
Message dataB:   0x27272727

Re-Sending finished !!!

Press '6' to display received messages...

Message ID:      0x00000004
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  STANDARD ID FRAME FORMAT
Message dataA:   0x23232323
Message dataB:   0x45454545

Message ID:      0x00000008
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  STANDARD ID FRAME FORMAT
Message dataA:   0x15151515
Message dataB:   0x36363636

Message ID:      0x00000026
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  STANDARD ID FRAME FORMAT
Message dataA:   0x76767676
Message dataB:   0x32323232

Message ID:      0x00001805
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  EXTENDED ID FRAME FORMAT
Message dataA:   0x78787878
Message dataB:   0x21212121

Message ID:      0x00005001
Message length:  0x00000008 BYTES
Message type:    DATA FRAME
Message format:  EXTENDED ID FRAME FORMAT
Message dataA:   0x85858585
Message dataB:   0x27272727

Demo terminal !!!

```

Fig 22. Press '6' to show new messages received and terminal example

3.3 CRC (CYCLIC REDUNDANCY CHECK)

3.3.1 Crc_Dma

3.3.1.1 Example description

Purpose

This example describes how to use CRC engine with DMA support.

Knowledge and Process

CRC use CRC-32 polynomial for demo.

DMA channel 0 will be used to transfer a block data to CRC WR_DATA register.

After initialize, CRC engine on the chip will calculate CRC-checksum for a block data with `BLOCK_SIZE` bytes and display result by serial communication with PC

3.3.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Crc_Dma.c: Main program file

3.3.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

This application is not required any hardware configuration.

3.3.1.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.3.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - After reset, welcome screen appears like this:

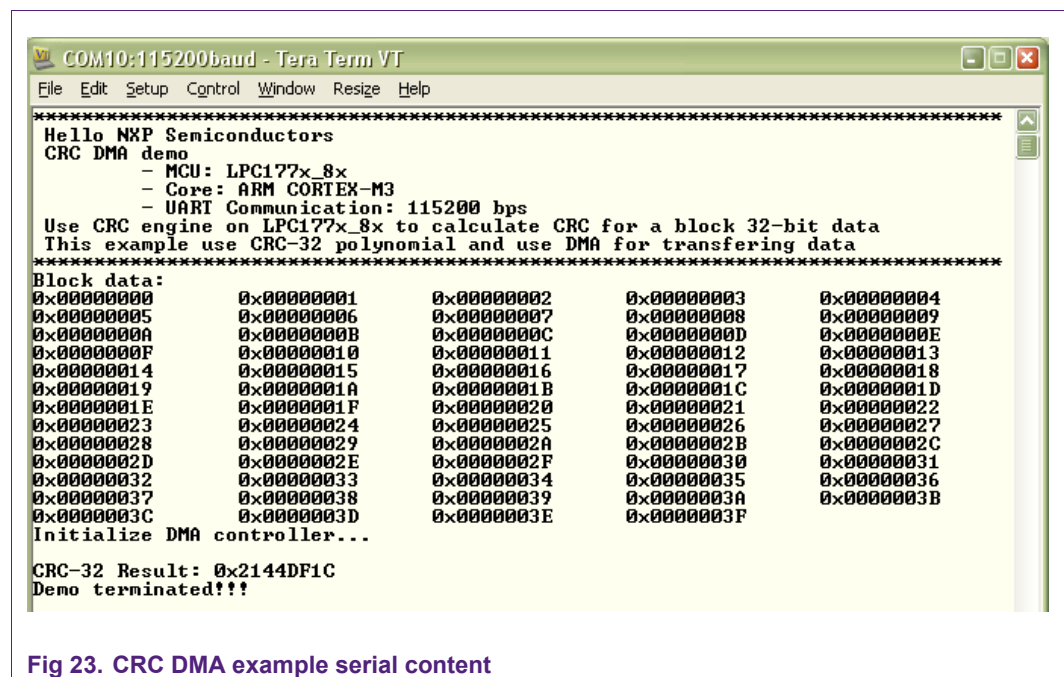


Fig 23. CRC DMA example serial content

3.3.2 Crc_Demo

3.3.2.1 Example description

Purpose

This example describes how to use CRC engine on LPC177x_8x.

Knowledge and Process

CRC can use one of three polynomials, include:

- CRC – CCITT:
- CRC – 16
- CRC – 31

By serial data communication from a PC's terminal, it's allowed to choose one of 3 polynomials above to calculation CRC value.

After initialize, CRC engine on the chip will calculate CRC-checksum for a block data with `BLOCK_SIZE` bytes and display result by serial communication with PC

3.3.2.2 Directory contents

`\EWARM:` includes EWARM (IAR) project and configuration files

`\Keil:` includes RVMDK (Keil) project and configuration files

`Crc_Demo.c:` Main program file

3.3.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

This application is not required any hardware configuration.

3.3.2.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.3.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - After reset, welcome screen appears like this:

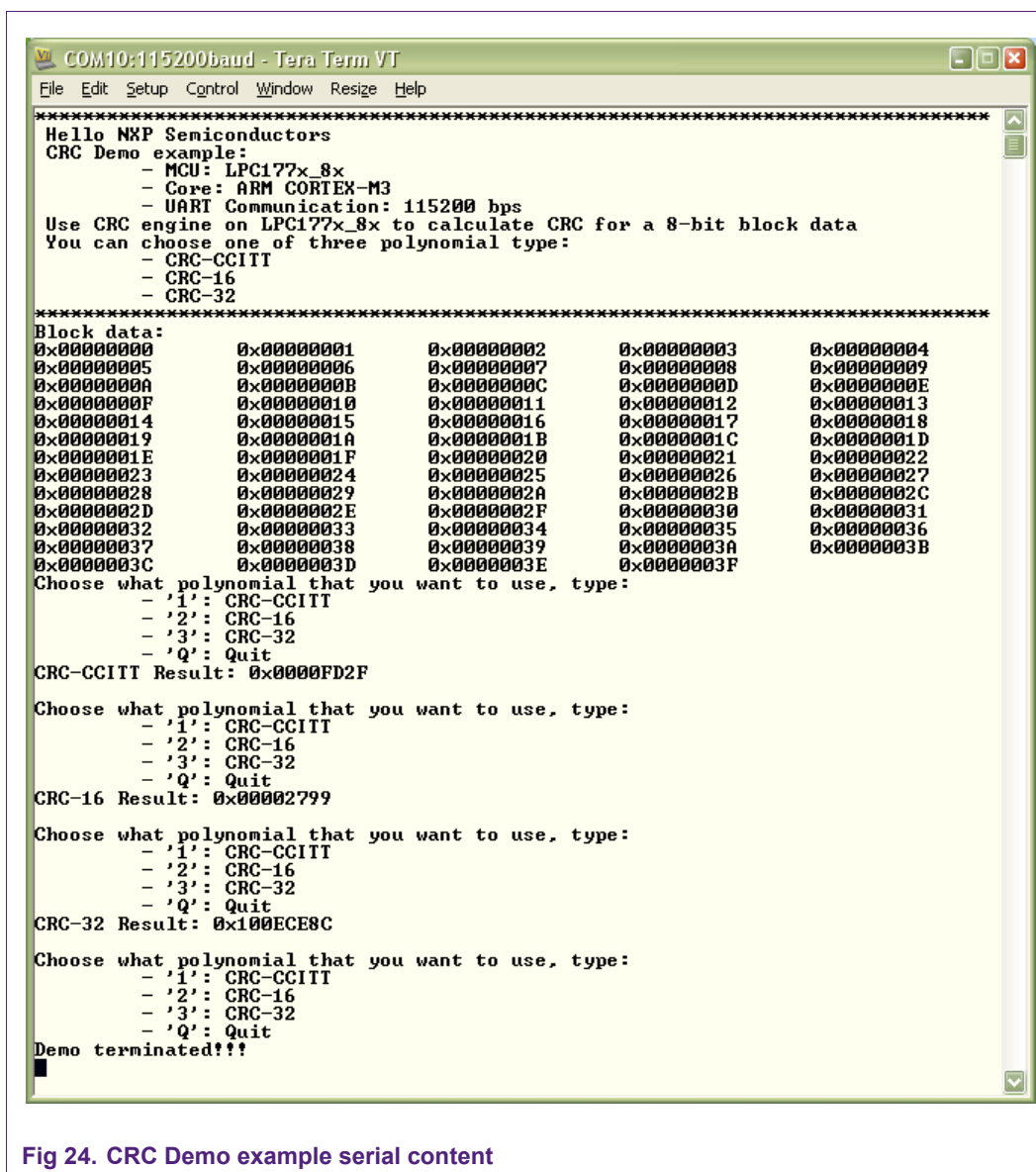


Fig 24. CRC Demo example serial content

3.4 DAC (DIGITAL – TO – ANALOG CONVERTER)

3.4.1 Dac_Dma

3.4.1.1 Example description

Purpose

This example describes how to use DMA to transmit and receive result data with DAC converter.

Knowledge and Process

DAC converter will be initialized with maximum current is 700uA. This allows a maximum update rate of 1Mhz

Formula for output voltage on Analog Output pin (AOUT) is:

$$VAOUT = VALUE \times ((Vrefp - Vrefn) / 1024) + Vrefn$$

in which:

- Vrefp: tied to VDD(3.3V)
- Vrefn: tied to Vss
- VALUE: a number (in Digital form) that expected to convert to Analog form of voltage at output
- VAOUT: Analog voltage at output pin (AOUT)

GPDMA channel 0 is used to configure in this example to work with DAC component.

GPDMA channel 0 will transfer a `dac_value` to DAC peripheral. DAC updated values have range from 0 to 0x3FF (10 – bit DAC). So AOUT output voltage will change from: Vss to VDD.

Observe AOUT (P0.26) signal by oscilloscope, it's a ramp wave form.

3.4.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Dac_Dma.c: Main program file

3.4.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Pin map

	<i>OEM Board</i>	<i>IAR Board</i>
<i>P0.26</i>	J3.28	EXT-11

3.4.1.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.4.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results: observe the signal output from DAC converter at P0.26 pin (AOUT) on the oscilloscope
 - After reset, the output is in ramp wave form as below:

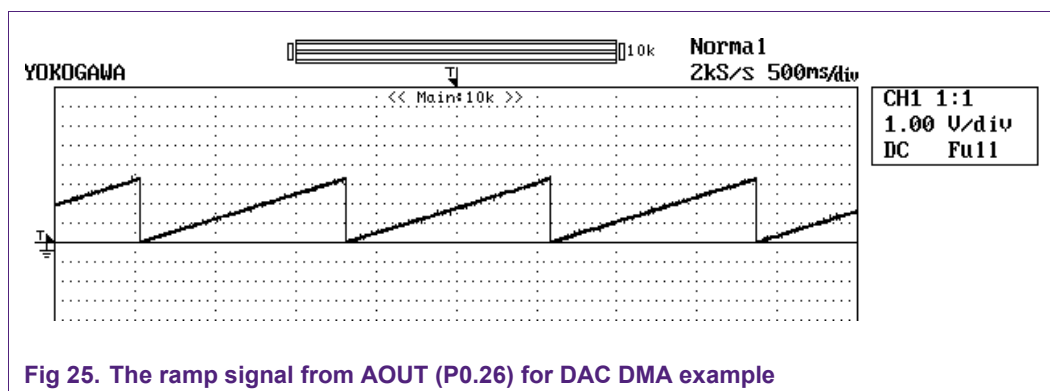


Fig 25. The ramp signal from AOUT (P0.26) for DAC DMA example

3.4.2 Dac_SineWave

3.4.2.1 Example description

Purpose

This example describes how to use DMA to generate a sine wave signal

Process

DAC converter will be initialized with maximum current is 700uA. This allows a maximum update rate of 1Mhz

Formula for output voltage on Analog Output pin (AOUT) is:

$$VAOUT = VALUE \times ((Vrefp - Vrefn) / 1024) + Vrefn$$

in which:

- Vrefp: tied to VDD(3.3V)
- Vrefn: tied to Vss
- VALUE: a number (in Digital form) that expected to convert to Analog form of voltage at output
- VAOUT: Analog voltage at output pin (AOUT)

DAC will generate a sine wave form with peak to peak is within *Vrefp* and *Vrefn*. We need to prepare a look up table with 60 items (*dac_sine_lut[]*), each item is the value to update AOUT voltage, and it's correspondent to a sample point of 1 circle sine wave signal. The formula is below:

```
for(i=0;i<NUM_SINE_SAMPLE;i++) //NUM_SINE_SAMPLE = 60
{
    dac_sine_lut[i] = OFFSET + AMPLITUDE * sin(i);
    dac_sine_lut[i] = (dac_sine_lut[i] << 6);
}
```

Piece of code to prepare the look – up table

GPDMA channel 0 is configured and used to transfer *dac_sine_lut[i]* to DAC peripheral. When the last item of *dac_sine_lut* is transferred, GPDMA will roll back to transfer the first item.

DAC is configured to use time out for each DAC value update and trigger GPDMA to fill DAC value register.

This time out value can also be used to calculate the sine wave frequency:

timeout=(PCLK_DAC_IN_MHZ*1000000)/(SINE_FREQ_IN_HZ*NUM_SINE_SAMPLE);

Where:

- PCLK_DAC_IN_MHZ = 25
- SINE_FREQ_IN_HZ = 60
- NUM_SINE_SAMPLE = 60

Observe AOUT (P0.26) signal by oscilloscope.

3.4.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Dac_SineWave.c: Main program file

3.4.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Pin map

	<i>OEM Board</i>	<i>IAR Board</i>
<i>P0.26</i>	J3.28	EXT-11

3.4.2.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.4.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results: observe the signal output from DAC converter at *P0.26* pin (AOUT) on the oscilloscope
 - After reset, the output is in sine wave form as below:

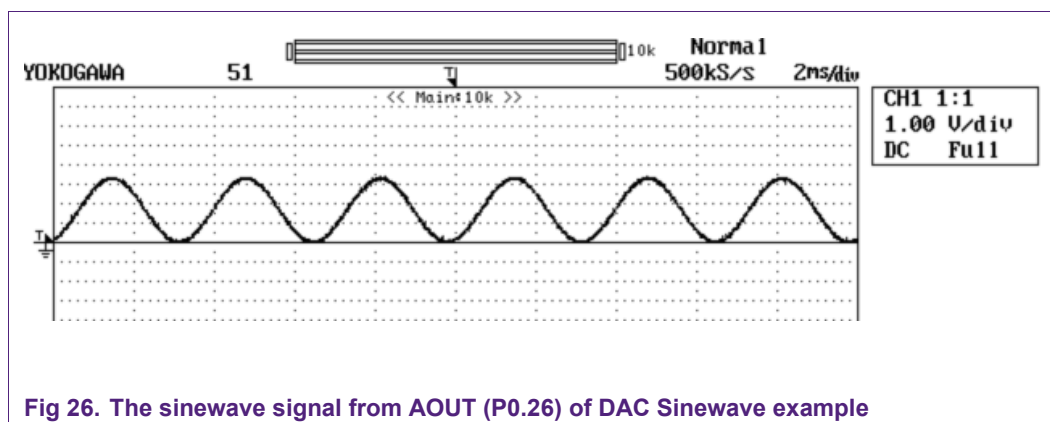


Fig 26. The sinewave signal from AOUT (P0.26) of DAC Sinewave example

- If modify this flag: `_ONE_POSITIVE_HALF` as:

```
#define _ONE_POSITIVE_HALF (1)
```

We will have another form of sinewave as captured:

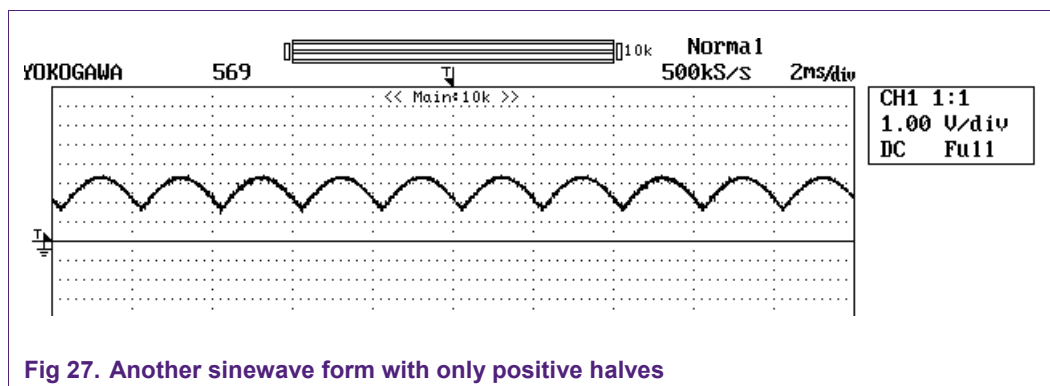


Fig 27. Another sinewave form with only positive halves

3.5 DMA (DIRECT MEMORY ACCESS)

3.5.1 Dma_Flash2Ram

3.5.1.1 Example description

Purpose

This example describes how to test GPDMA (General Purpose Direct Memory Access) function by transferring data from Flash to Ram memory

Process

This example will transfer a block of data from Flash memory to Ram memory.

Transferred block is initialized by defining constant array `DMA_Scr_Buffer`. This buffer will be burned into flash when compile. Received block data is `DMA_Dest_Buffer` will be stored in ram when compiling.

GPDMA channel 0 is configured in this example.

Transfer size is 16 words.

After transferring completed, `Buffer_Verify()` will be called to compare data block in memory source and destination. If not similar, program will enter infinite loop.

Open serial display to see DMA transfer result.

3.5.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Dma_Flash2Ram.c: Main program file

3.5.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

3.5.1.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.5.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - After reset, then see the serial terminal, it should be like this:

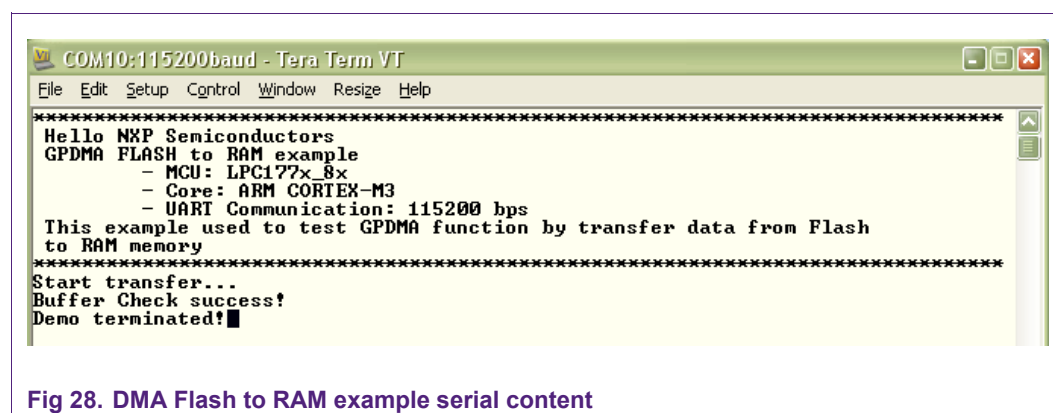


Fig 28. DMA Flash to RAM example serial content

3.6 EEPROM (ELECTRICALLY ERASABLE PROGRAMMABLE READ – ONLY MEMORY)

3.6.1 Eeprom_Demo

3.6.1.1 Example description

Purpose

This example describes how to work with EEPROM (Electrically Erasable Programmable Read-Only Memory) memory on LPC177x_8x.

Knowledge and Process

This example will be use `MODE_8_BIT` to write/read data from EEPROM memory.

First, data will be written at the position `PAGE_OFFSET` of `PAGE_ADDR`.

Then, using `EEPROM_Read()` to read back data from this address.

Data will be displayed via serial screen.

3.6.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Eeprom_Demo.c: Main program file

3.6.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

3.6.1.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.6.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board. *Make sure that the OEM board is currently using unless its running is not ok.*
- (6) Testing actions and results:
 - After reset, this example should display a "HELLO WORLD" phrase as below:

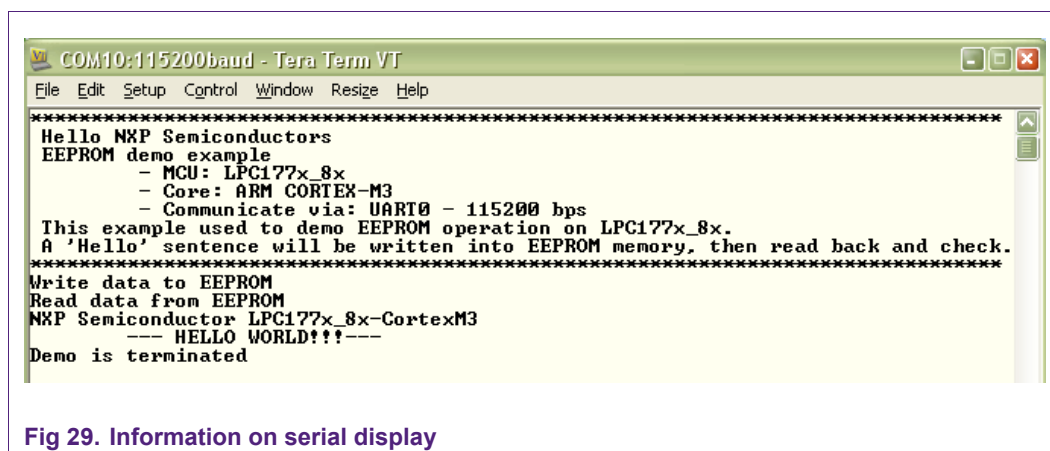


Fig 29. Information on serial display

3.7 EMAC (ETHERNET MEDIA ACCESS CONTROLLER)

3.7.1 Emac_EasyWeb

3.7.1.1 Example description

Purpose

This example describes how to implement a simple web application

Process

This tiny web server was taken from the 'Design & Elektronik' extra issue 'Embedded Internet'. It can be downloaded from the following web site:

www.elektroniknet.de/extraheft.

Note that modifications are not optimal, because ARM is a 32 – bit machine while the original software was written for 16-bit CPU.

The web page shows the values of two analog inputs (AN0 and AN1).

This tiny webserver needs very little resources and therefore has some restrictions:

- Only one active TCP session at any one time
- No support for fragmented IP datagrams
- No buffer for TCP datagrams received in wrong order
- Only one web page. No GIF/JPG graphics possible.

The IP address can be modified in the module `tcpip.h` to fit into your existing LAN (see `MYIP_x`).

The default IP address of this server is: 192.168.0.100

3.7.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

ADC.h/.c: ADC low level functions

easyweb.c/.h: easy web application (Main program)

EMAC.h/.c: LPC1768 EMAC hardware driver functions

Retarget.c: target-dependent low level functions

tcpip.h/.c: implement TCP/IP stack functions

webpage.h: webpage html source

3.7.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

These jumpers must be configured as follows:

➤ **LPC1788 OEM board rev A connects with OEM base board rev A**

- ✓ Remain jumpers: default

➤ **LPC1788 IAR Start Kit Rev.B**

- ✓ Jumper **PD/CTRL** must be at **2 – 3**: to power PHY LAN chip

This application need to use a **CrossOver internet cable** to connect from PC to ETH port on evaluation board

3.7.1.4 Running mode

This example can run only on ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.7.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions. Remember to use a Crossover cable connection.
- (5) Modify configuration for IP (Internet Protocol) at PC site with below value as shown by the picture captured:
 - IP address: 192.168.0.x (x != 100)
 - Subnet mask: 255.255.255.0

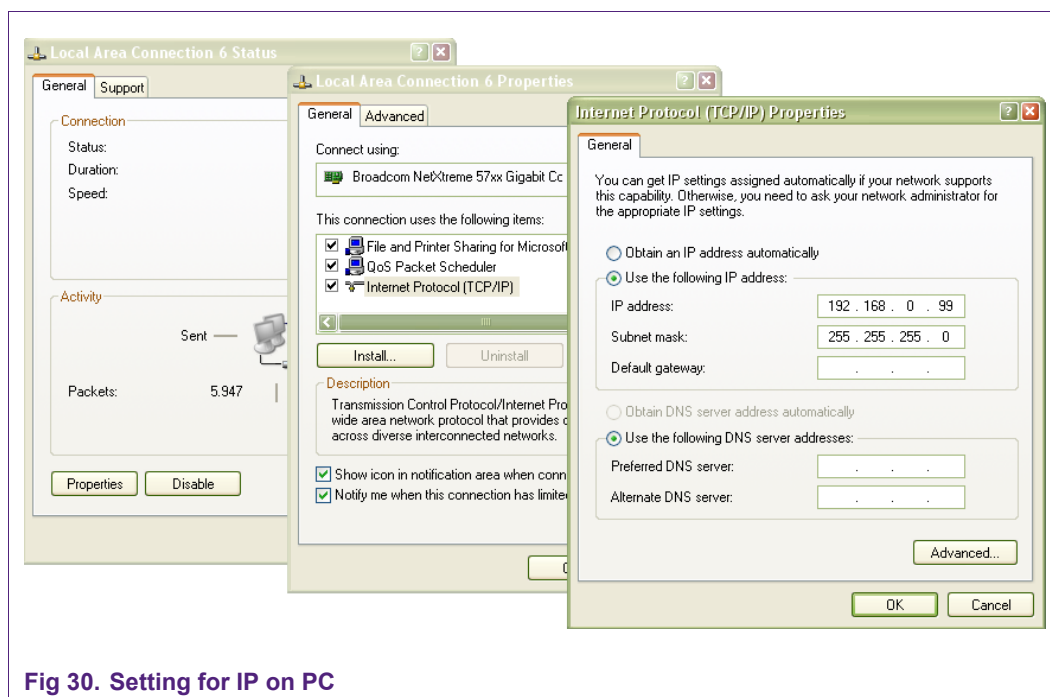


Fig 30. Setting for IP on PC

- (6) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (7) Testing actions and results:
- After the evaluation board initialization, open command prompt on window PC and execute "ping 192.168.0.100" command.

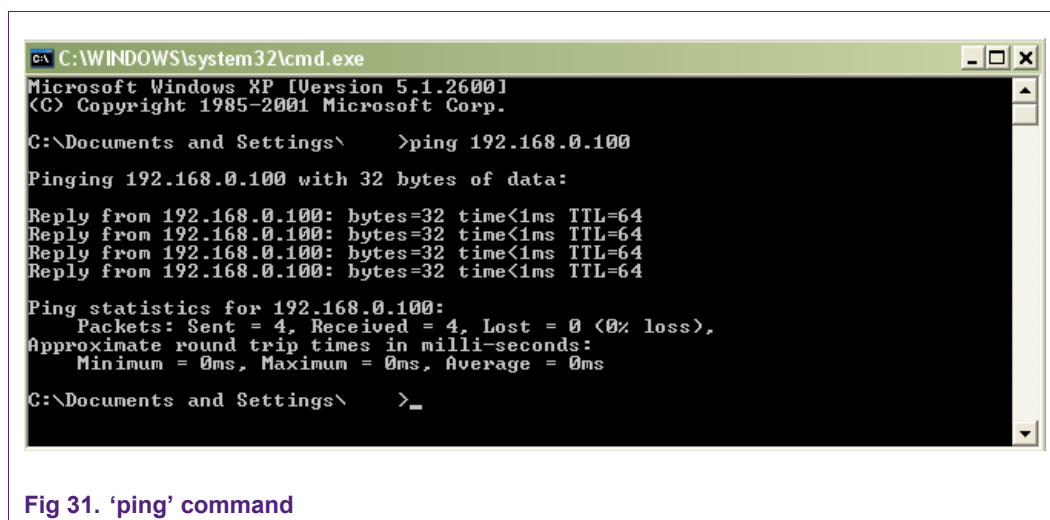


Fig 31. 'ping' command

- Open web browser, access to address "http://192.168.0.100" to display the content of webserver:

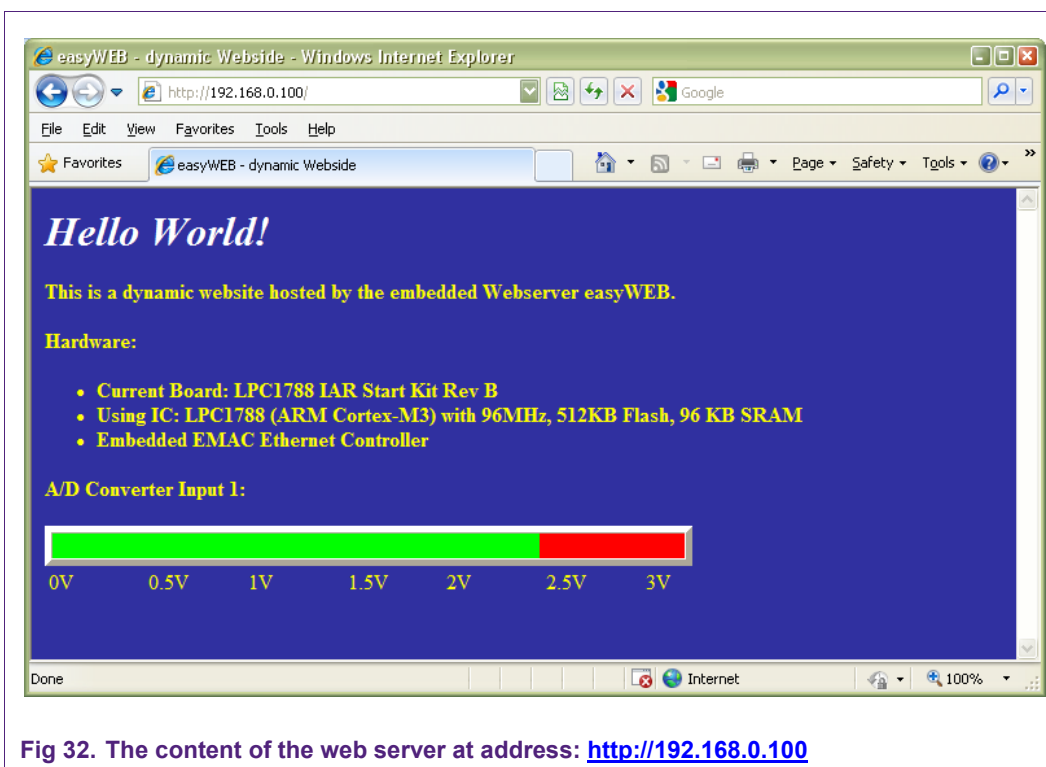


Fig 32. The content of the web server at address: <http://192.168.0.100>

- Turn potentiometer and see the web update for ADC value on “**A/D Converter Input 1:**” field

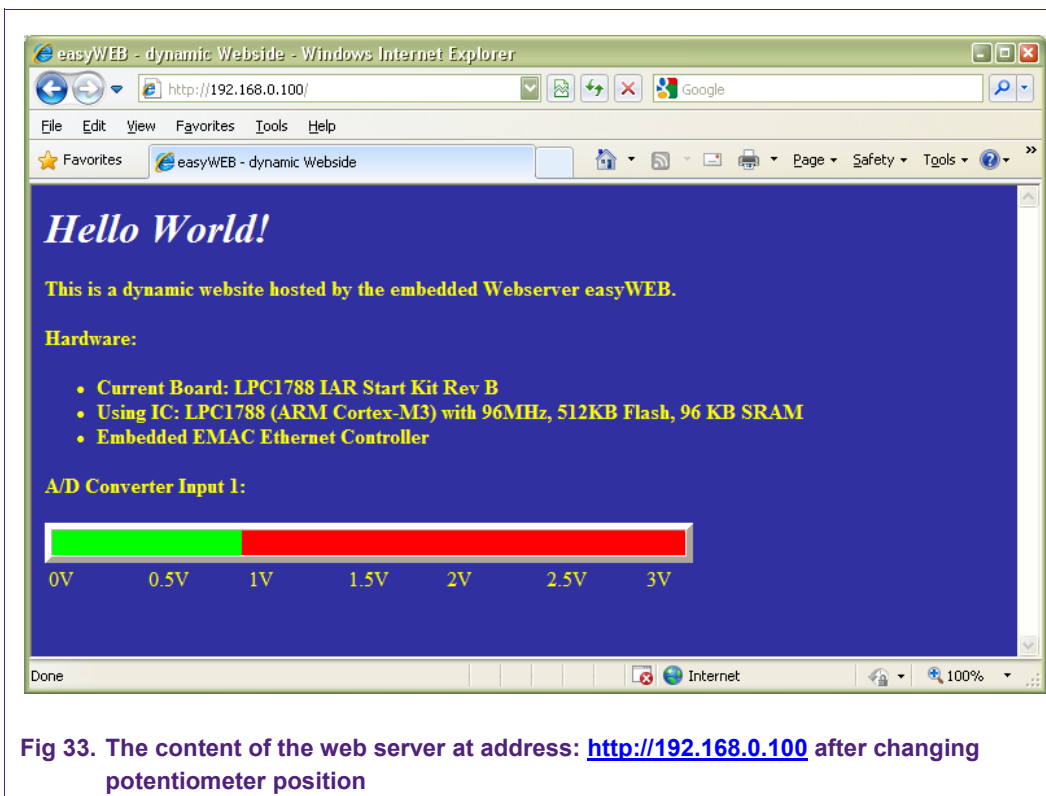


Fig 33. The content of the web server at address: <http://192.168.0.100> after changing potentiometer position

3.7.2 Emac_Raw

3.7.2.1 Example description

Purpose

This example describes how to test EMAC driver with raw packet frame format that is not related with any upper-layer (i.e. TCP/IP...).

Process

There are two ways to test:

- `TX_ONLY` and `BOUNCE_RX` flags can be set one at a time, not both. When `TX_ONLY` is set to 1, it's a `TX_ONLY` packet from the MCB1700 board to the LAN. Use the traffic analyzer such as ethereal, once the program is running, the packets can be monitored on the traffic analyzer.
- When `BOUNCE_RX` is set to 1 (`TX_ONLY` needs to reset to 0), it's to test both TX and RX, use the traffic generator/analyzer, you can create a packet with the destination address as that on the MCB1700 board, use the traffic generator to send packets, as long as the destination address matches, MCB1700 will reverse the source and destination address and send the packets back on the network.

`ENABLE_WOL` flag is used to test power down and WOL functionality.

`BOUNCE_RX` flag needs to be set to 1 when WOL is being tested.

3.7.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

crc32c.h/.c: Ethernet CRC module

Emac_Raw.c: main program

libnosys.gnu.c: Definitions for OS interface, stub function required by newlibc used by Codesourcery GNU compiler.

3.7.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

These jumpers must be configured as follows:

➤ **LPC1788 OEM board rev A connects with OEM base board rev A**

- ✓ Remain jumpers: Default

➤ **LPC1788 IAR Start Kit Rev.B**

- ✓ Jumper **PD/CTRL** must be at **2 – 3**: to power PHY LAN chip

This application need to use a **CrossOver internet cable** to connect from PC to ETH port on evaluation board

3.7.2.4 Running mode

This example can run only on ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.7.2.5 Step to run

(1) Build examples: this test requires 2 evaluation boards for TX and RX functions. These functions are decided by the flags below in file `emactest.c`

- TX site:
 - `TX_ONLY = 1`
 - `BOUNCE_RX = 0`
- RX site:
 - `TX_ONLY = 0`
 - `BOUNCE_RX = 1`

→ Start building and creating 2 images after 2 configurations above.

(2) Burn 2 built image files (in hex) into the board (in case of ROM mode running)

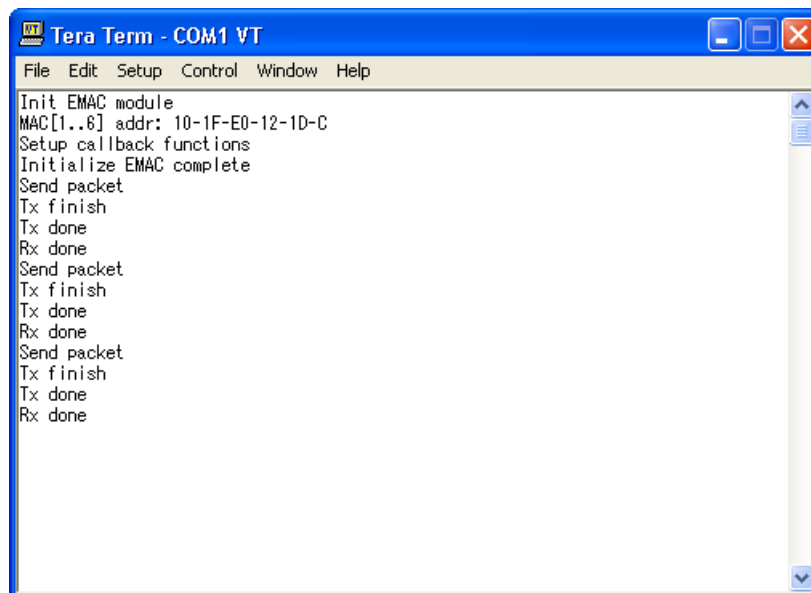
(3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)

(4) Do hardware configuration exactly following the above instructions. Remember to use a Crossover cable connection.

(5) Reset 2 boards (in case of ROM mode running) and Run the example on the working boards.

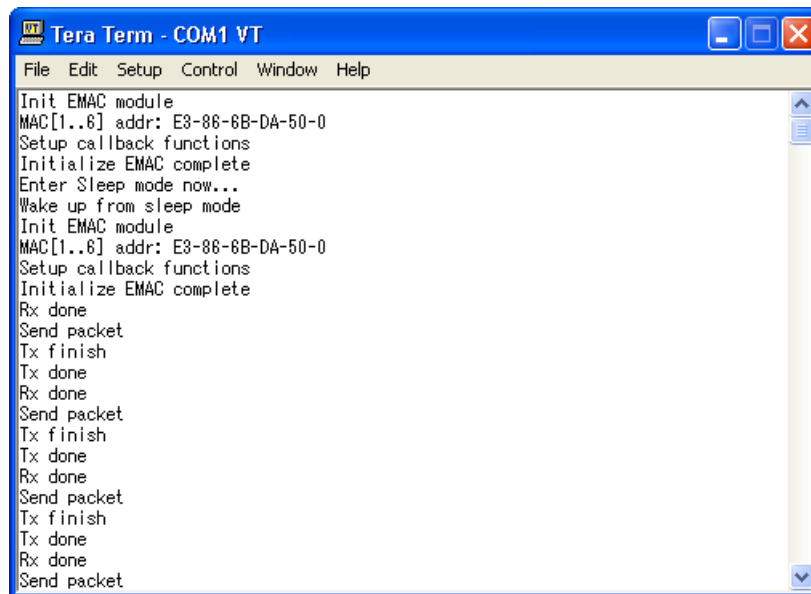
(6) Testing actions and results:

- Serial terminal on PC will show the status messages from 2 boards
- In case the `ENABLE_WOL` is enabled on the RX site,
 - RX site will enter sleep mode after its initialization success. Then it will wait for Waked – up o LAN (WoL) from TX site.
 - If hitting `INT0` on TX site, it will send a frame through LAN to RX site. This message will wake up the RX site.
 - Once RX site receive the sending frame from TX site, it will wake up then work as normal.



```
File Edit Setup Control Window Help
Init EMAC module
MAC[1..8] addr: 10-1F-E0-12-1D-C
Setup callback functions
Initialize EMAC complete
Send packet
Tx finish
Tx done
Rx done
Send packet
Tx finish
Tx done
Rx done
Send packet
Tx finish
Tx done
Rx done
```

Fig 34. Status on 'TX_ONLY' side



```
File Edit Setup Control Window Help
Init EMAC module
MAC[1..8] addr: E3-86-6B-DA-50-0
Setup callback functions
Initialize EMAC complete
Enter Sleep mode now...
Wake up from sleep mode
Init EMAC module
MAC[1..8] addr: E3-86-6B-DA-50-0
Setup callback functions
Initialize EMAC complete
Rx done
Send packet
Tx finish
Tx done
Rx done
Send packet
Tx finish
Tx done
Rx done
Send packet
Tx finish
Tx done
Rx done
Send packet
```

Fig 35. Status on 'BOUNCE_RX' side

3.7.3 Emac_ulp

3.7.3.1 Example description

Purpose

This example describes how to handle a single network interface and contains the IP, ICMP, UDP and TCP protocols.

Process

The uIP TCP/IP stack is an extremely small implementation of the TCP/IP protocol suite intended for embedded systems running low-end 8 or 16-bit microcontrollers. The code size and RAM requirements of uIP is an order of magnitude smaller than other generic TCP/IP stacks today.

The uip_webserver implements WEB server.

The default IP address is:

192.168.0.100

The default router's IP address is:

192.168.0.1

The subnet mask is:

255.255.255.0

IP address on your PC should not be one of these IP addresses above

3.7.3.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

\apps: contains a few example applications

- **\dhcpc:** Implementation of DHCP protocol
- **\hello-world:** A small example showing how to write applications with sockets.
- **\resolv:** DNS resolver
- **\smtp:** SMTP E-mail sender
- **\telnetd:** Implementation of TELNET network protocol
- **\webclient:** Implementation of the HTTP client.
- **\webserver:** Implementation of an HTTP server

\common: implement some supported standard functions (printf, serial..)

\uip: contains files that implement uIP stack

\lpc178x_port: include main program

3.7.3.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

These jumpers must be configured as follows:

➤ **LPC1788 OEM board rev A connects with OEM base board rev A**

- ✓ Remain jumpers: Default

➤ **LPC1788 IAR Start Kit Rev.B**

- ✓ Jumper **PD/CTRL** must be at **2 – 3**: to power PHY LAN chip

This application need to use a **Crossover internet cable** to connect from PC to ETH port on evaluation board

3.7.3.4 Running mode

This example can run only on ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.7.3.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions. Remember to use a Crossover cable connection.
- (5) Modify configuration for IP (Internet Protocol) at PC site with below value as shown by the picture captured:
 - IP address: 192.168.0.x (x != 100)
 - Subnet mask: 255.255.255.0

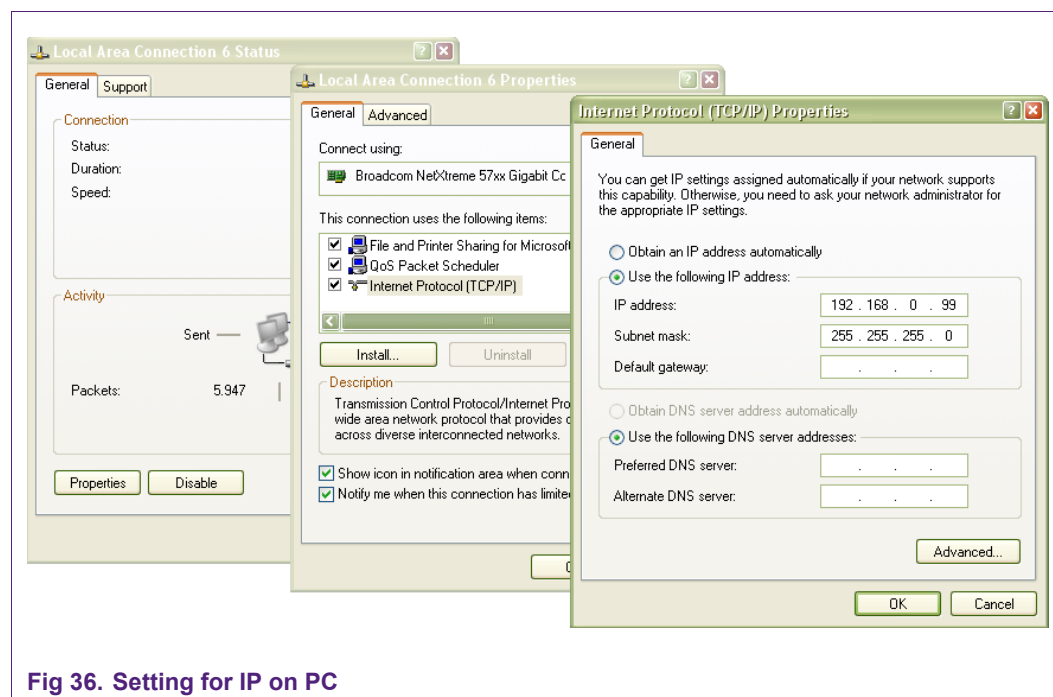


Fig 36. Setting for IP on PC

- (6) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (7) Testing actions and results:
 - After the evaluation board initialization, open command prompt on window PC and execute "ping 192.168.0.100" command.

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ >ping 192.168.0.100

Pinging 192.168.0.100 with 32 bytes of data:

Reply from 192.168.0.100: bytes=32 time<1ms TTL=64
Reply from 192.168.0.100: bytes=32 time<1ms TTL=64
Reply from 192.168.0.100: bytes=32 time<1ms TTL=64
Reply from 192.168.0.100: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.0.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\ >_

```

Fig 37. 'ping' command

- Open web browser, access to address "http://192.168.0.100" to display the content of the webpage:

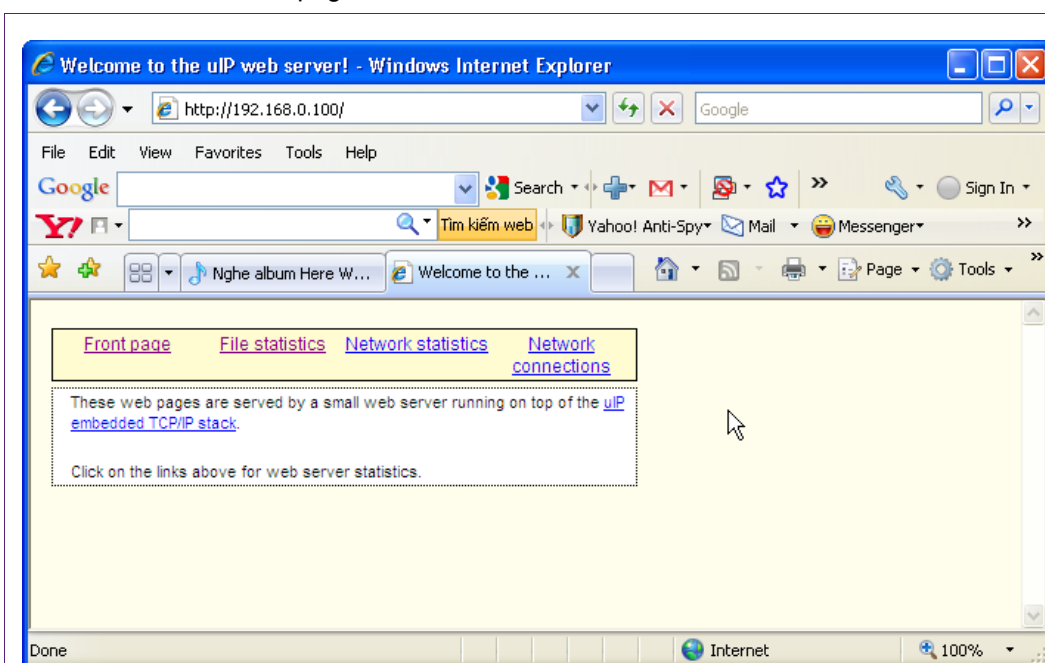


Fig 38. Work with Web browser

3.8 EMC (EXTERNAL MEMORY CONTROLLER)

3.8.1 Emc_NandFlashDemo

3.8.1.1 Example description

Purpose

This example describe how to use EMC (External Memory Controller) connects with On-board NAND FLASH on OEM LPC1788 board

Knowledge and Process

NAND flash use this case is Samsung 128Byte K9F1G08U0A.

NAND flash memory range will be mounted at address: 0x8000 0000 – 0x800FF FFFF.

First, EMC will be initialized and the application does set up timer for compatibleness with this NAND flash chip being used.

Then check Device ID. If not correct, process will enter into infinite loop and display error via UART.

If ID device checking is Okie, entire Flash memory will be erased, 2K block data will be written and read back for verify. Verify process result will be display on terminal screen.

3.8.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Emc_NandFlashDemo.c: Main program

3.8.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD in .\.\.\BoardSupport\bsp.h` file for current working board.

Depending on the board, the NandFlash IC that is current being mounted on is different; it's needed to change the code followingly.

➤ LPC1788 OEM board rev A connects with OEM base board rev A

This board is currently using K9F1G08U0C NandFlash IC with EMC Controller of LPC1788x IC

➤ LPC1788 IAR Start Kit Rev.B

It's unable to run this example because there's no NandFlash that mounted on this board.

3.8.1.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.8.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - After reset, then see the serial terminal, it should be like this:

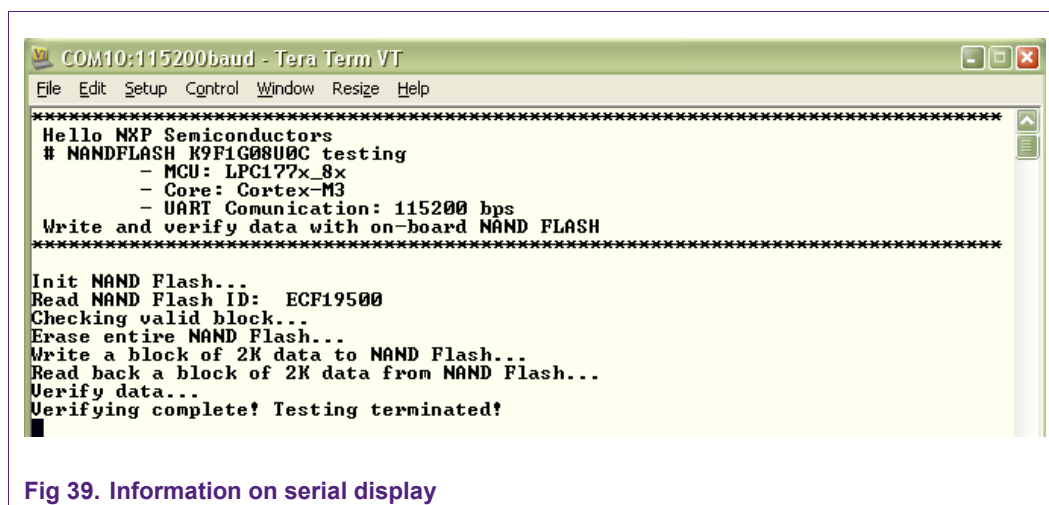


Fig 39. Information on serial display

3.8.2 Emc_NorFlashDemo

3.8.2.1 Example description

Purpose

This example describes how to use EMC connects with On-board NOR FLASH on OEM LPC1788 board

Process

NOR flash use this case is 4M Bytes SST39VF3201.

NOR flash memory range will be mounted at address: 0x8100 0000 - 0x81FFF FFFF.

First, EMC will be initialized and the application does setup timer for compatibility with this NOR flash chip being used.

Then check Device ID. If not correct, process will enter into infinite loop and display error via UART.

If ID device checking is ok, entire Flash memory will be erased, 2K block data will be written and read back for verify. Verify process result will be display on terminal screen.

3.8.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Emc_NorFlashDemo.c: Main program

3.8.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Depending on the board, the NorFlash IC that is current being mounted on is different; it's needed to change the code followingly.

➤ LPC1788 OEM board rev A connects with OEM base board rev A

This board is currently using SST39VF3201 NorFlash chip with EMC controller of LPC1788 IC.

➤ LPC1788 IAR Start Kit Rev.B

It's unable to run this example because there's no NorFlash chip that mounted on this board.

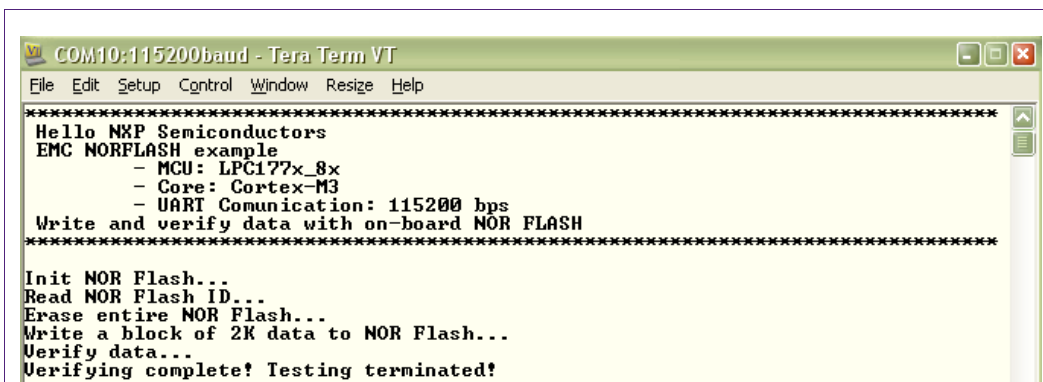
3.8.2.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.8.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - After reset, then see the serial terminal, it should be like this:



```
COM10:115200baud - Tera Term VT
File Edit Setup Control Window Resize Help
*****
Hello NXP Semiconductors
EMC NORFLASH example
- MCU: LPC177x_8x
- Core: Cortex-M3
- UART Communication: 115200 bps
Write and verify data with on-board NOR FLASH
*****
Init NOR Flash...
Read NOR Flash ID...
Erase entire NOR Flash...
Write a block of 2K data to NOR Flash...
Verify data...
Verifying complete! Testing terminated!
```

Fig 40. Information on serial display

3.8.3 Emc_SdramDemo

3.8.3.1 Example description

Purpose

This example describe how to use EMC connects with On-board SDRAM on LPC1788 board

Process

NOR flash use this case is 256Mbit SDRAM Micron MT48LC8M32LFB5 (on QVGA board), or K4S561632J (on OEM board).

NOR flash memory range will be mounted at address: 0xA000 0000 – 0xAFFF FFFF.

First, EMC will be initialized and the application does setup timer for compatiblens with this SDRAM chip being used. Entire content on SDRAM will be cleared.

Then continue write data into SDRAM in 8-bits mode and 16-bits mode sequentially.

After each mode writing, data in SDRAM will be verified. If verify process fail, program will enter into infinite loop.

Please observe terminal screen for more information.

3.8.3.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Emc_SdramDemo.c: Main program

3.8.3.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

Depending on the board, the SDRAM IC that is current be mounted on is different; it needs to change the code as following.

➤ **LPC1788 OEM board rev A connects with OEM base board rev A**

MT48LC8M32LFB5 SDRAM chip is mounted on this board

➤ **LPC1788 IAR Start Kit Rev.B**

K4S561632J SDRAM chip is mounted on this board

3.8.3.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.8.3.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - After reset, then see the serial terminal, it should be like this:

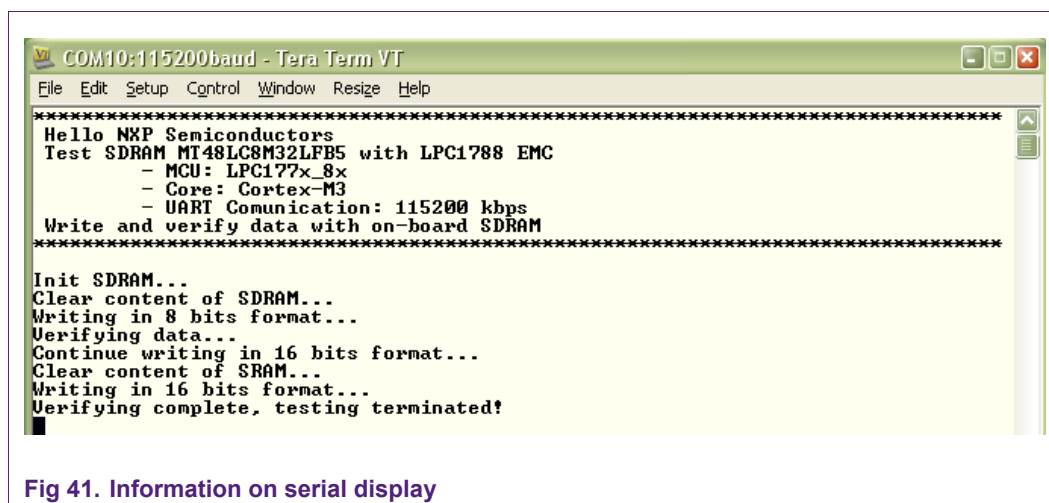


Fig 41. Information on serial display

3.9 GPIO (GENERAL PURPOSE INPUT/OUTPUT)

3.9.1 Gpio_Interrupt

3.9.1.1 Example description

Purpose

This example describes how to use GPIO interrupt function..

Knowledge and Process

Interrupt is configured to use *P0.25* pin (on OEM board) or *P0.13* (on IAR board), because they are linked to ADC potentiometer.

Turn the ADC potentiometer according to clock direction until interrupt happen.

3.9.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Gpio_Interrupt.c: Main program

3.9.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

3.9.1.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.9.1.5 Step to run

(1) Build example

- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - Turn potentiometer until the GPIO interrupt occurs, the LED *P0.13* (on OEM board) *P1.18* (on IAR board) is blinked 10 times.

3.9.2 Gpio_LedBlinky

3.9.2.1 Example description

Purpose

This simple program used to test GPIO interrupt functionality to drive LED

Process

After reset, the LED connected to pin P0.13 will be blinking

3.9.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Gpio_LedBlinky.c: Main program

3.9.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It doesn't required any jumper configuration or link.

3.9.2.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.9.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:

- After the board is reset, the LED *P1.18* (on OEM board) *P1.13* (on IAR board) is blinked 10 times.

3.10 I2C (INTER – IC CONTROL BUS)

3.10.1 I2c_Pa9532Drv

3.10.1.1 Example description

Purpose

This example describes how to use I2C to drive with PCA9532.

Knowledge and Process

I2C Clock Rate is set at 200K.

PCA8581 Slave address is 0xC0 (8-bit format)

After initialize and enable I2C, PCA9532 should be configured to blink

LEDs connected at specific frequencies, always OFF or ON.

By this example, it's configured to:

- LED1 and LED7: blink at 30% of their brightness
- LED2 and LED6: OFF always
- LED3 and LED8: ON always
- LED4 and LED5: blink at 2Hz and duty-cycle 50%

3.10.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

I2c_Pca9532Drv.c: Main program

3.10.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

➤ LPC1788 OEM board rev A connects with OEM base board rev A

It doesn't required any jumper configuration or link.

➤ LPC1788 IAR Start Kit Rev.B

There is no PCA9532 chip mounted on this board. This example does not work on this board

3.10.1.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.10.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results: See LED show:
 - LED1 and LED7: blink at 30% of their brightness
 - LED2 and LED6: OFF always
 - LED3 and LED8: ON always
 - LED4 and LED5: blink at 2Hz and duty-cycle 50%

3.11 I2S (INTER – IC SOUND BUS)

3.11.1 I2s_Interrupt

3.11.1.1 Example description

Purpose

This example describes how to use I2S to transfer data in interrupt mode.

Knowledge and Process

I2S setup:

- wordwidth: 16 bits
- stereo mode
- master mode for TX and slave mode for RX
- `ws_halfword = 31`
- frequency = 44.1Khz (maximum is 96kHz)

This setup can be changed to test with other configurations.

Source data are initialized at `I2S_BUFFER_SRC` and will be copy to `I2S_BUFFER_DST` by I2S peripheral.

- Transfer size = 0x400 bytes
- `rx_depth_irq = 1`
- `tx_depth_irq = 8`

First, I2S transmit channel will send data to I2S receive channel, when I2S receive data, it generate interrupt, `I2S_IRQHandler()` Service Routine will be called to receive data and save it into `I2SRXBuffer`.

After transmission finished, `Buffer_Verify()` will check RX and TX buffer data. The result will be printed out serial display.

Please note that because I2S is the protocol for audio data transfer, so sometime it has dummy data while FIFO transmit is empty. These data are not importance and they can be ignored when verify.

3.11.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

I2s_Interrupt.c: Main program

3.11.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

I2S connection: I2S-RX connects to I2S-TX as following:

- ✓ **P0.4-I2SRX_CLK** connects to **P0.7-I2STX_CLK**
- ✓ **P0.5-I2SRX_WS** connects to **P0.8-I2STX_WS**
- ✓ **P0.6-I2SRX_SDA** connects to **P0.9-I2STX_SDA**

Pin-map:

	OEM board	IAR board
<i>I2SRX_CLK</i>	J5.15	EXT-4
<i>I2SRX_WS</i>	J5.16	EXT-5
<i>I2SRX_SDA</i>	J3.18	EXT-6
<i>I2STX_CLK</i>	J5.17	EXT-7
<i>I2STX_WS</i>	J3.19	EXT-8
<i>I2STX_SDA</i>	J5.18	EXT-9

3.11.1.4 Running mode

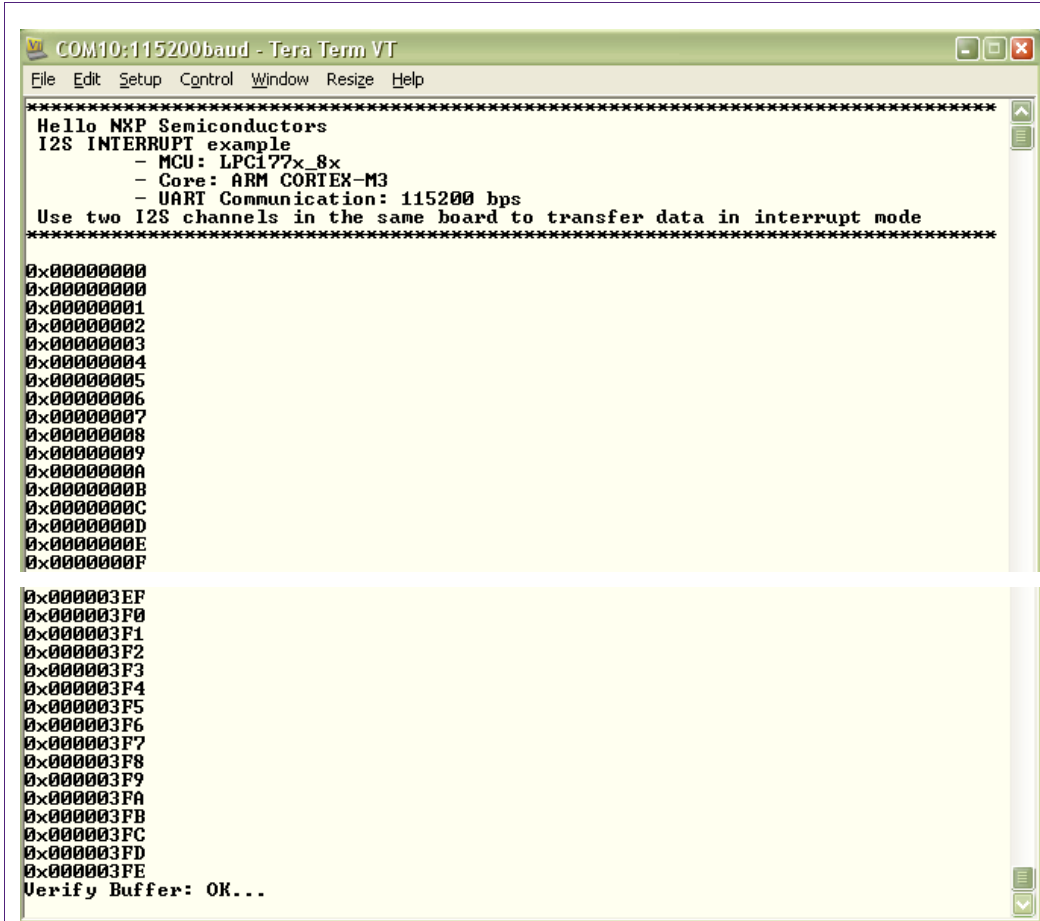
This example can run only on RAM /ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.11.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:

- After reset, then see the serial terminal, it should be like this:



```

COM10:115200baud - Tera Term VT
File Edit Setup Control Window Resize Help
*****
Hello NXP Semiconductors
I2S INTERRUPT example
- MCU: LPC177x_8x
- Core: ARM CORTEX-M3
- UART Communication: 115200 bps
Use two I2S channels in the same board to transfer data in interrupt mode
*****
0x00000000
0x00000001
0x00000002
0x00000003
0x00000004
0x00000005
0x00000006
0x00000007
0x00000008
0x00000009
0x0000000A
0x0000000B
0x0000000C
0x0000000D
0x0000000E
0x0000000F

0x000003EF
0x000003F0
0x000003F1
0x000003F2
0x000003F3
0x000003F4
0x000003F5
0x000003F6
0x000003F7
0x000003F8
0x000003F9
0x000003FA
0x000003FB
0x000003FC
0x000003FD
0x000003FE
Verify Buffer: OK...

```

Fig 42. Information on serial display

3.11.2 I2s_Dma

3.11.2.1 Example description

Purpose

This example describes how to use I2S in DMA mode

Knowledge and Process

I2S setup:

- wordwidth: 16 bits
- stereo mode
- master mode for TX and slave mode for RX
- `ws_halfword = 31`
- frequency = 44.1Khz (maximum is 96kHz)

GPDMA channel 0 and 1 are configured in this example.

DMA channel 0 used to transfer data from internal RAM source to I2S peripheral

DMA channel 1 used to transfer data from I2S peripheral to internal RAM destination.

- Transfer size = 0x0a bytes
- rx_depth_dma = 8
- tx_depth_dma = 1

So, when receive FIFO level = 8, it triggers a receive DMA request save data from I2S to destination memory.

And when transmit FIFO level = 1, it triggers a transmit DMA request to send data from source memory to I2S. After transmission finished, `Buffer_Verify()` function will be called to compare data from source and destination. If not similar, it will return FALSE.

Open serial terminal to observe I2S transfer process.

Please note that because I2S is the protocol for audio data transfer, so sometime it has dummy data while FIFO transmit is empty. These data are not importance and they can be ignored when verify.

3.11.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

I2s_Dma.c: Main program

3.11.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

I2S connection: I2S-RX connects to I2S-TX as following:

- ✓ **P0.4-I2SRX_CLK** connects to **P0.7-I2STX_CLK**
- ✓ **P0.5-I2SRX_WS** connects to **P0.8-I2STX_WS**
- ✓ **P0.6-I2SRX_SDA** connects to **P0.9-I2STX_SDA**

Pin-map:

	OEM board	IAR board
<i>I2SRX_CLK</i>	J5.15	EXT-4
<i>I2SRX_WS</i>	J5.16	EXT-5
<i>I2SRX_SDA</i>	J3.18	EXT-6
<i>I2STX_CLK</i>	J5.17	EXT-7
<i>I2STX_WS</i>	J3.19	EXT-8
<i>I2STX_SDA</i>	J5.18	EXT-9

3.11.2.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.11.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - After reset, it needs to follow the requests as press '1', '2', '3' to do necessary things of the test.
 - Check the serial terminal like this:

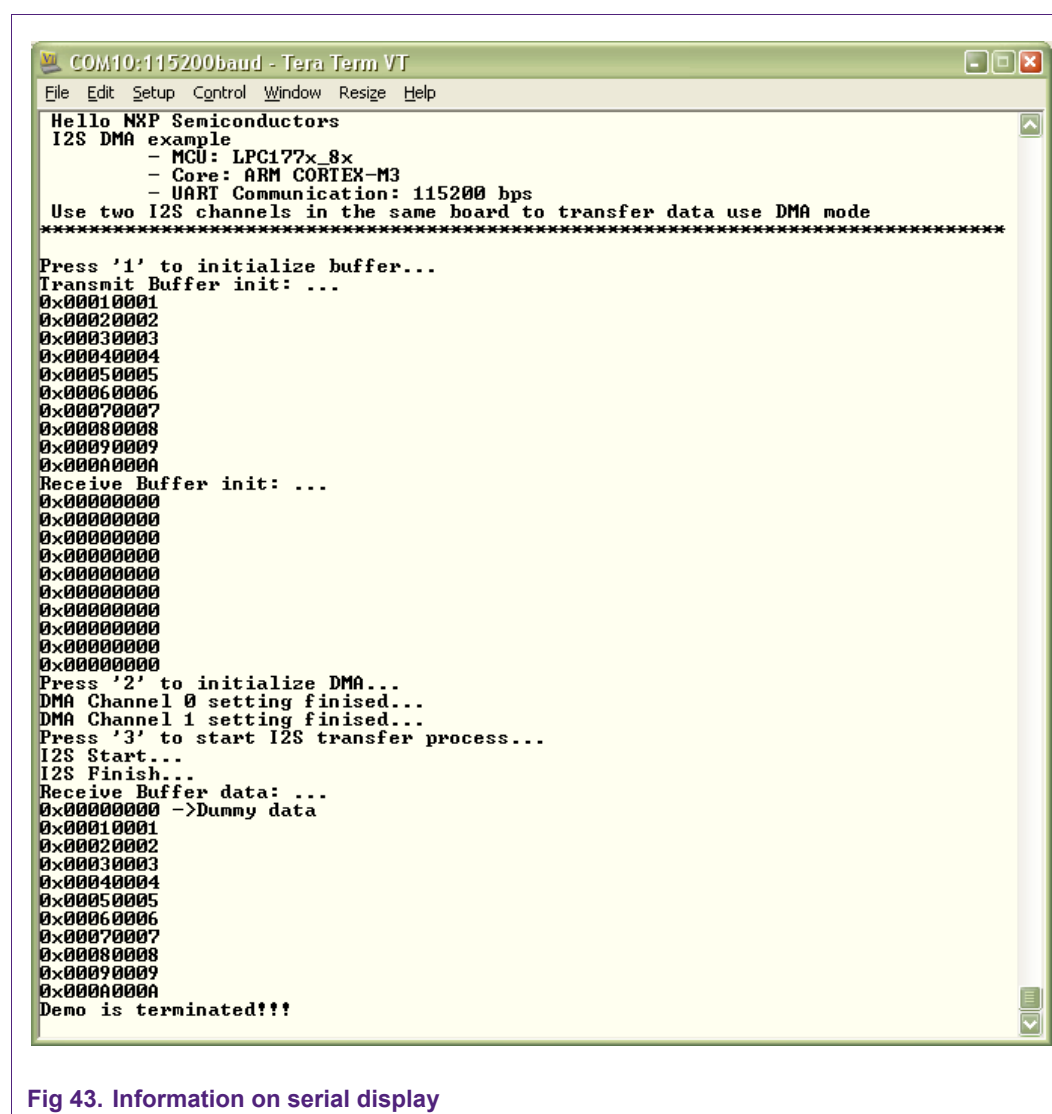


Fig 43. Information on serial display

3.11.3 I2s_Mclk

3.11.3.1 Example description

Purpose

This example describes how to use I2S master clock as I2S clock source

Knowledge and Process

I2S setup:

- wordwidth: 16 bits
- stereo mode
- master mode for TX and slave mode for RX
- `ws_halfword = 31`
- frequency = 44.1Khz (maximum is 96kHz)

I2S Clock setting:

- Transmitter:
 - Select the RX fractional rate divider clock output as the source
 - Disable 4-wire mode
 - Enable TX_MCLK output (this setting is optional, just used to observe this clock on oscilloscope)
- Receiver:
 - Select the TX_MCLK signal as RX_MCLK clock source
 - Disable 4-wire mode
 - Disable RX_MCLK output

Clock calculation formula:

- $MCLK = I2S_freq * (bitrate + 1)$
- $bitrate = channel * wordwidth - 1$

Ex: a 48 kHz sample rate for 16-bit stereo data will have master clock:

$$MCLK = 48K * 16 * 2 = 1.536Mhz$$

This example sets I2S receiver use MCLK from I2S transmitter. So we don't have to connect wire between *I2SRX_CLK* and *I2STX_CLK*. I2S transmits data in interrupt mode. After finished, received and transmitted buffer will be verified, the result will be print out serial display.

Please note that because I2S is the protocol for audio data transfer, so sometime it has dummy data while FIFO transmit is empty. These data are not importance and they can be ignored when verify.

3.11.3.2 Directory contents

\EWARM:	includes EWARM (IAR) project and configuration files
\Keil:	includes RVMDK (Keil) project and configuration files
I2s_Mclk.c:	Main program

3.11.3.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

I2S connection: I2S-RX connects to I2S-TX as following:

- ✓ **P0.5-I2SRX_WS** connects to **P0.8-I2STX_WS**
- ✓ **P0.6-I2SRX_SDA** connects to **P0.9-I2STX_SDA**

Pin-map:

	OEM board	IAR board
<i>I2SRX_CLK</i>	J5.15	EXT-4
<i>I2SRX_WS</i>	J5.16	EXT-5
<i>I2SRX_SDA</i>	J3.18	EXT-6
<i>I2STX_CLK</i>	J5.17	EXT-7
<i>I2STX_WS</i>	J3.19	EXT-8
<i>I2STX_SDA</i>	J5.18	EXT-9

3.11.3.4 Running mode

This example can run only on RAM /ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.11.3.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - After reset, then see the serial terminal, it should be like this:

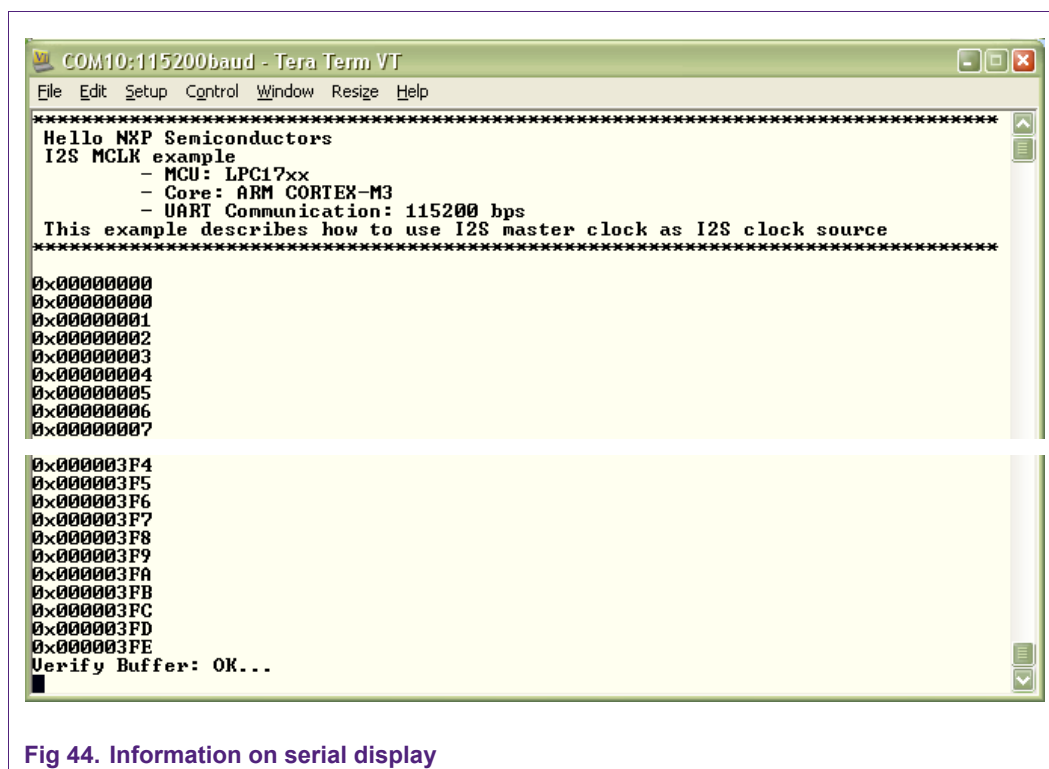


Fig 44. Information on serial display

3.11.4 I2s_4Wire

3.11.4.1 Example description

Purpose

This example describes how to configure I2S peripheral to run in 4 wire mode

Knowledge and Process

I2S setup:

- wordwidth: 16 bits
- stereo mode
- master mode for TX and slave mode for RX
- `ws_haftword = 31`
- frequency = 44.1Khz (maximum is 96kHz)

This setup can be changed to test with other configurations.

Source data are initialized at `I2S_BUFFER_SRC` and will be copy to `I2S_BUFFER_DST` by I2S peripheral.

Transfer size is 0x400 bytes

I2S Receiver is set in 4-wire mode, sharing the Transmitter bit clock and WS.

So no need connect I2S clock and WS pin.

This example is not use interrupt mode but use polling mode instead to handle I2S operation.

After transmission finished, `Buffer_Verify()` function will be called to compare data from source and destination. If not similar, it will return `FALSE`.

Open serial terminal to observe I2S transfer process.

Please note that because I2S is the protocol for audio data transfer, so sometime it has dummy data while FIFO transmit is empty. These data are not importance and they can be ignored when verify.

3.11.4.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

I2s_4Wire.c: Main program

3.11.4.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD in .\.\.\BoardSupport\bsp.h` file for current working board.

I2S connection: I2S-RX connects to I2S-TX as following:

✓ **P0.6-I2SRX_SDA** connects to **P0.9-I2STX_SDA**

Pin-map:

	OEM board	IAR board
<i>I2SRX_CLK</i>	J5.15	EXT-4
<i>I2SRX_WS</i>	J5.16	EXT-5
<i>I2SRX_SDA</i>	J3.18	EXT-6
<i>I2STX_CLK</i>	J5.17	EXT-7
<i>I2STX_WS</i>	J3.19	EXT-8
<i>I2STX_SDA</i>	J5.18	EXT-9

3.11.4.4 Running mode

This example can run on both RAM/ ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.11.4.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - After reset, then see the serial terminal, it should be like this:

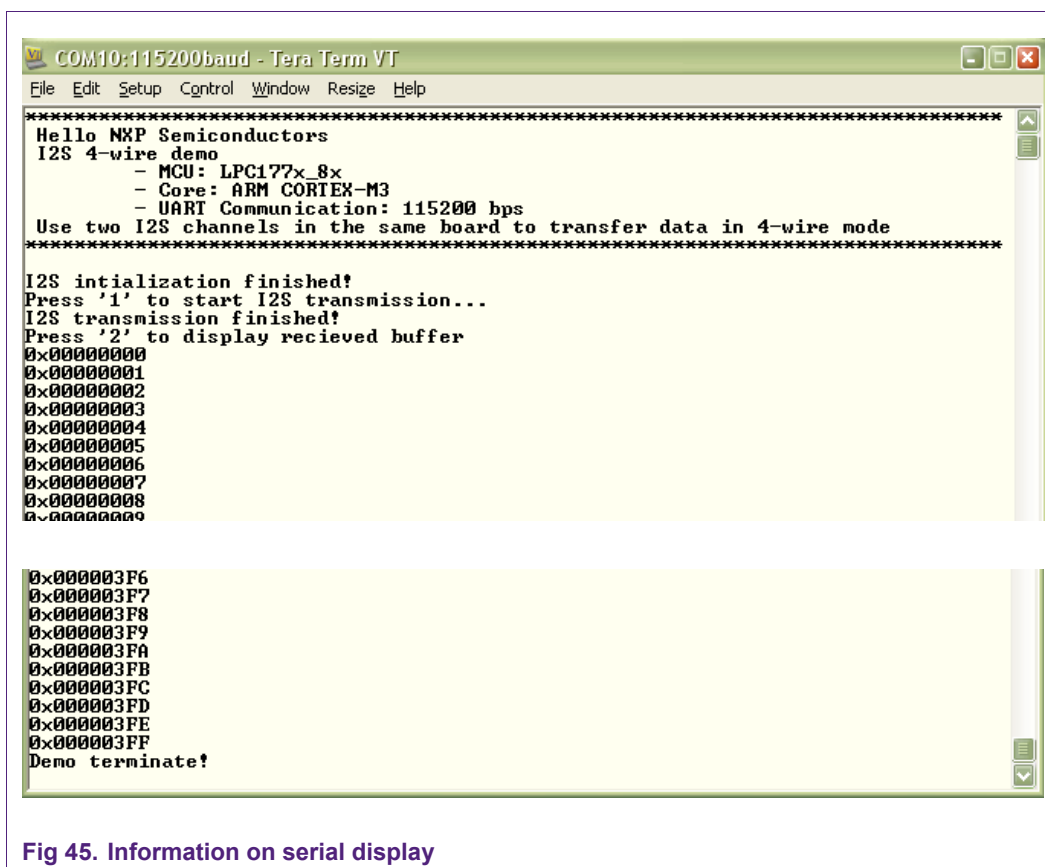


Fig 45. Information on serial display

3.11.5 I2s_Audio

3.11.5.1 Example description

Purpose

This example describes how to use I2S to transfer audio data through I2S port to play a short music sound and it can be heard from headphone line.

Knowledge and Process

I2S setup:

- wordwidth: 16 bits
- stereo mode
- master mode for TX
- frequency = 44.1Khz (maximum is 96kHz)
- enable transmit interrupt mode
- IRQ TXFIFO depth = 4

A sample of audio data is stored in `tx_buffer[]` buffer for using later in this example.

This application is using external I2S DAC IC: *UDA1134ATS* to output audio sound via headphone. Audio data will be copied from `tx_buffer[]` to TXFIFO register of the IC, after half of buffer have been sent already, call `I2S_Callback()` function to re-fill data.

3.11.5.2 Directory contents

EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

I2s_Audio.c: Main program

3.11.5.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD in .\.\.\BoardSupport\bsp.h` file for current working board.

I2S connection: I2S-TX as following:

- ✓ **I2S_TX_CLK (P0.7)** connects with **UDA1134-pin1-BCK**
- ✓ **I2S_TX_WS (P0.8)** connects with **UDA1134-pin2-WS**
- ✓ **I2S_TX_SDA (P0.9)** connects with **UDA1134-pin3-DATA**

Please see *UDA1134ATS.png* file to know how to connect I2S ports with external I2S DAC IC.

Pin-map:

	OEM board	IAR board
<i>I2SRX_CLK</i>	J5.15	EXT-4
<i>I2SRX_WS</i>	J5.16	EXT-5
<i>I2SRX_SDA</i>	J3.18	EXT-6
<i>I2STX_CLK</i>	J5.17	EXT-7
<i>I2STX_WS</i>	J3.19	EXT-8
<i>I2STX_SDA</i>	J5.18	EXT-9

3.11.5.4 Running mode

This example can run only on ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.11.5.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - Connect a headphone and hear the sound from there

3.12 LCD (LIQUID CRYSTAL DISPLAY)

3.12.1 Lcd_GFT035A320240Y

3.12.1.1 Example description

Purpose

This example describes how to interact with LCD GFT035A320240Y.

Knowledge and Process

The LCD GFT035A320240Y is mounted on *IAR (LPC1788 IAR Start Kit Rev.B)* board until this time.

This code is referenced from IAR source code.

The IAR, NXP and Olimex logos appear on the LCD and the LCD hardware cursor moves if the board moves (the acceleration sensor is used).

It shows basic use of I/O, timer, interrupt and LCD controllers.

3.12.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

\app: Include main file, logo & cursor image

\board: board configurations file

\modules: include supported drivers for this example

3.12.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

➤ LPC1788 OEM board rev A connects with OEM base board rev A

Because on this OEM board, there's no LCD mounted, this example is unable to run with this kind of board

➤ LPC1788 IAR Start Kit Rev.B

This colorful LCD is running well with a good power source.

Notes: Should take care whether it's enough power (current) for the LCD working. In some cases, the LCD is dim because of the power supplier.

3.12.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.12.1.5 Step to run

(1) Build example

(2) Burn built image file (in hex) into the board (in case of ROM mode running)

(3) Do hardware configuration exactly following the above instructions.

- (4) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (5) Testing actions and results: See the colorful content on the LCD
 - See the colorful content on the LCD

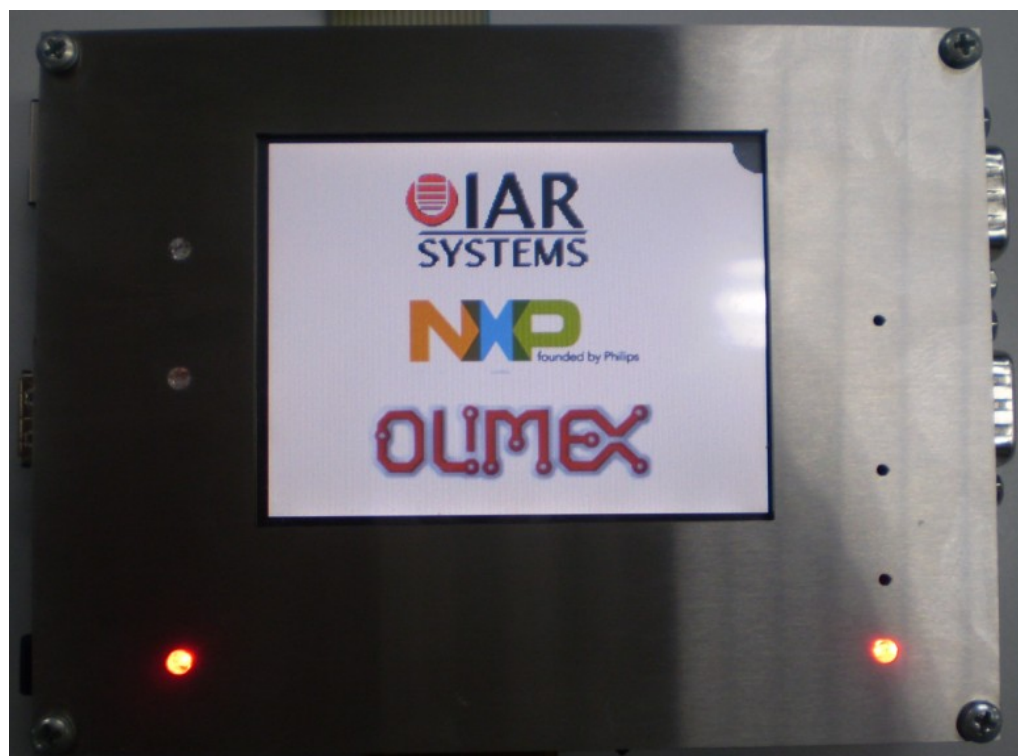


Fig 46. Whole the area of the IAR board when running LCD example

- Check the cursor on the screen

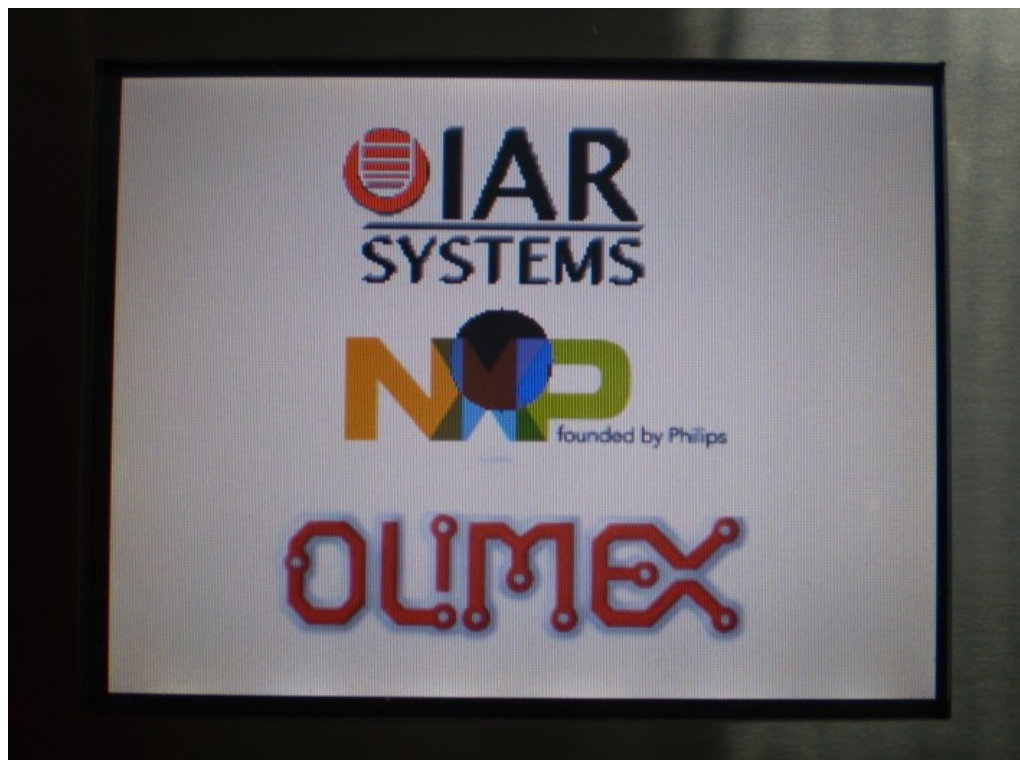


Fig 47. Another position of the cursor on the screen

3.13 MCI (MULTIMEDIA CARD INTERFACE)

3.13.1 Mci_CidCard

3.13.1.1 Example description

Purpose

This example describes how to use Multimedia Card Interface (MCI) on LPC177x_8x IC

Process

Plug the card to the slot, LPC1788 will read the ID of the card and show it to a serial terminal on a PC.

It can be used with SD card, micro SD card (with adapter) or MMC card.

3.13.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Mci_CidCard.c: Main program

3.13.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

This example does not need any added configuration for hardware.

3.13.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.13.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - This example is working well with SD Card (or micro-SD card with adapter) which is plugged in the card slot.

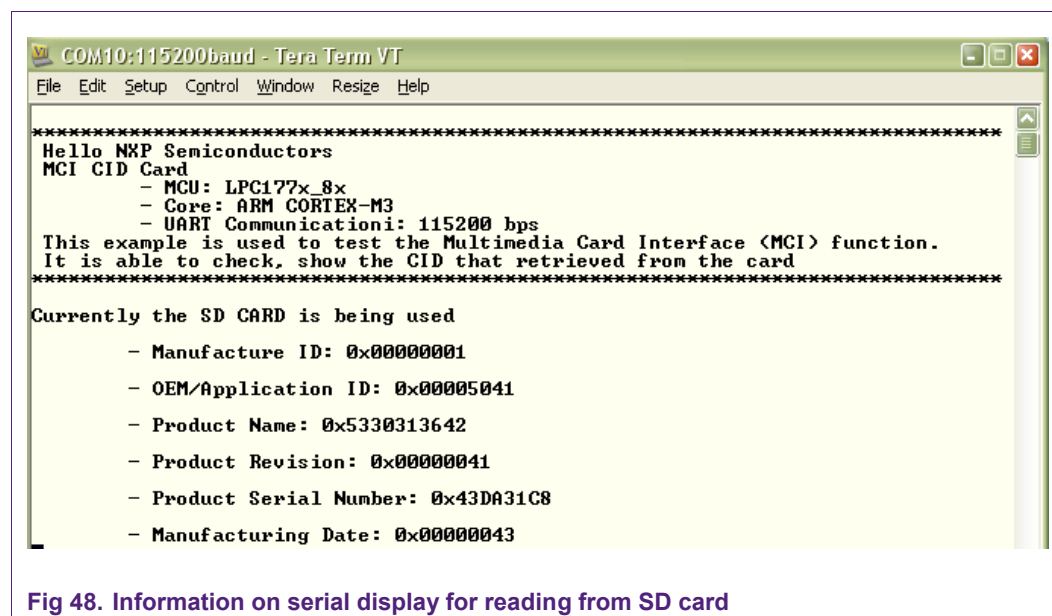
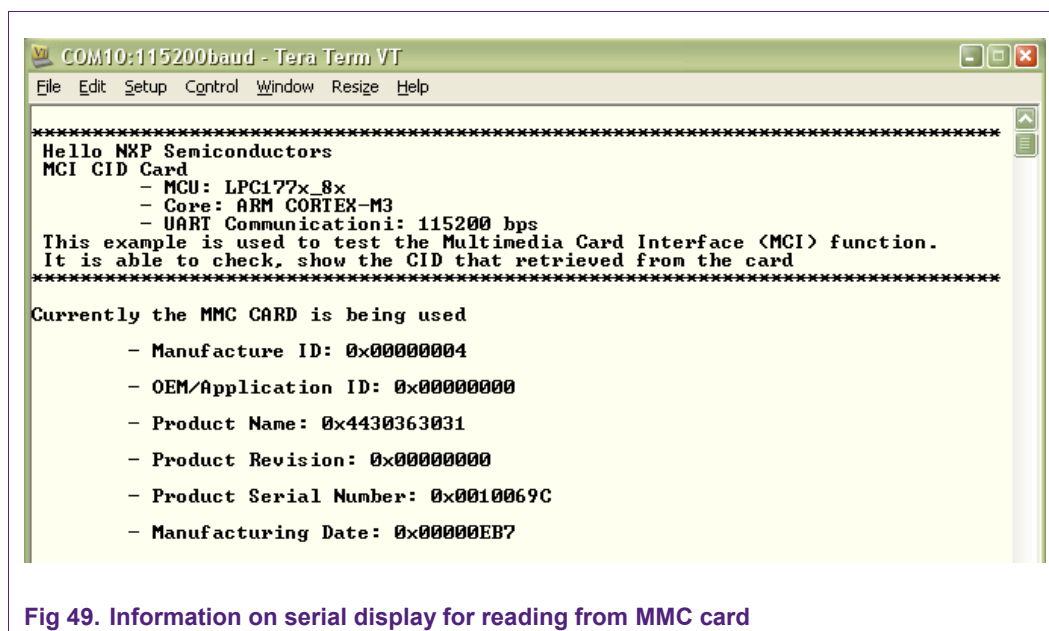


Fig 48. Information on serial display for reading from SD card

- This example also is working well with MMC Card which is plugged in the card slot



3.14 MCPWM (MOTOR CONTROL PWM)

3.14.1 MCPWM_Simple

3.14.1.1 Example description

Purpose

This example describes how to test Motor Control PWM module in LPC177x_8x.

Process

Testing function on MCPWM could be:

- 3 – phase AC mode: inverted output is enable, A0 and A1 output pin is internally routed to A0 signal
- 3 – phase DC mode: inverted output is enable
- Capture on Motor Control: in this case is used to detect the falling edge on *MCO0B* output pin. The *MCFB0* input pin therefore must be connected to MCO0B.
 - Capture Channel 0.
 - Capture falling edge on MCFB0 input pin.
 - Interrupt enabled on capture event.

Channel 0, 1, 2 will be configured as follows:

- edge aligned operation
- polarity pin: Passive state is LOW, active state is HIGH
- disable dead time
- enable update value
- period time = 300
- pulse width value:
 - channel 0 = 0

- channel 1 = 100
- channel 2 = 200

The program will update the value for pulse width for 3 channels continuously from 0 to 300, increase 20 each update time. After each update, the serial will write "Update!" into screen. After that, `CapFlag` will be checked if detect falling edge on MCO0B or not. If yes, the program will print the current capture value into screen.

3.14.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

mcpwm_simple.c: Main program

3.14.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

For the capturing function, pin MCFB/MCI0 (P1.20) must be connected with pin MC0B (P1.22) to capture the pulse-width of signal output of MC0B from the beginning to the falling edge.

The output signal can be observed by oscilloscope on these pins below:

- P1.19 - MC0A
- P1.22 - MC0B
- P1.25 - MC1A
- P1.26 - MC1B
- P1.28 - MC2A
- P1.29 - MC2B

This example does not need any jumper configuration.

3.14.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.14.1.5 Step to run

(1) Setting the example for expected purpose:

Testing purpose(s): it needs some pre-setting here in file `Mcpwm_Simple.c`:

- If test 3 – phase DC mode: Do definement below:

```
#define MCPWM_WORKING_MODE DC_MODE_TEST
```

- If test 3 – phase AC mode: Do definement below:

```
#define MCPWM_WORKING_MODE AC_MODE_TEST
```

- If test Capture MCPWM mode, do definement below:

```
#define CAPTURE_MODE_TEST 1
```

- (2) Build example
- (3) Burn built image file (in hex) into the board (in case of ROM mode running)
- (4) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (5) Do hardware configuration exactly following the above instructions.
- (6) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (7) Testing actions and results:
 - Monitor the wave form at MC0A (P1.19) and MC0B (P1.22) pins on the oscilloscope in the case of test: it should be one of 2 captured below
 - ✓ If test 3-phase DC mode:

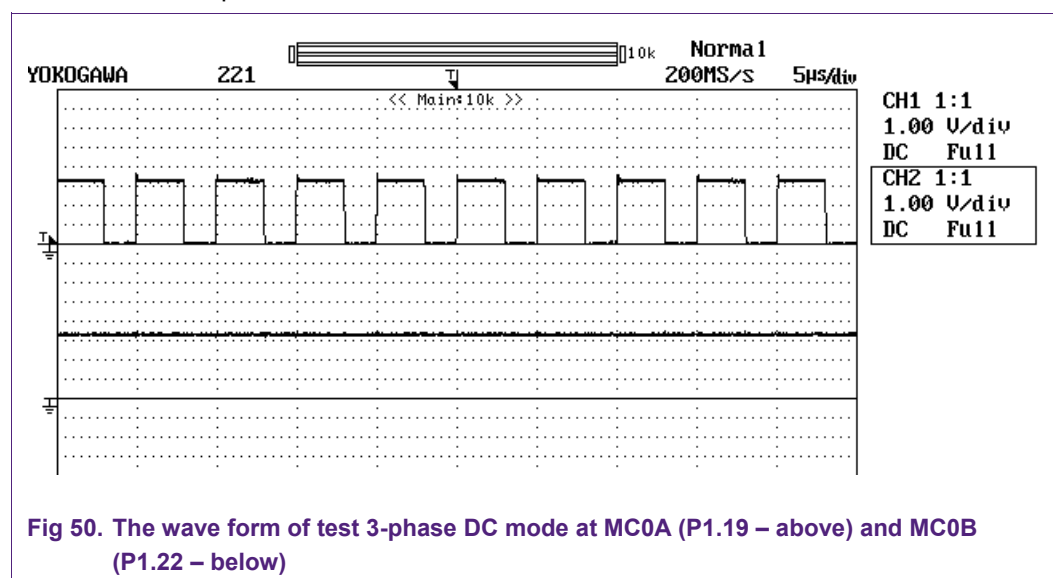


Fig 50. The wave form of test 3-phase DC mode at MC0A (P1.19 – above) and MC0B (P1.22 – below)

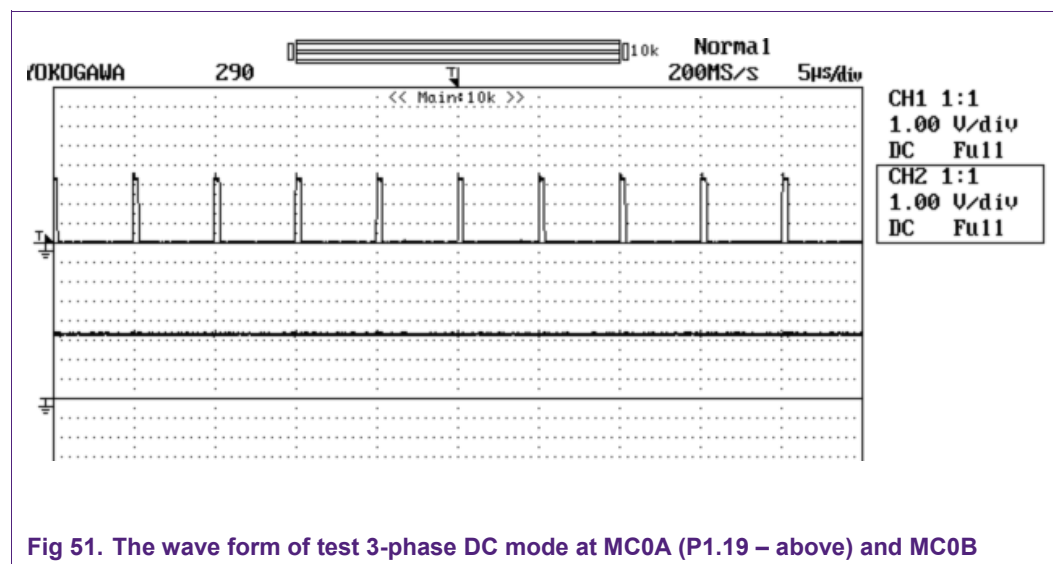


Fig 51. The wave form of test 3-phase DC mode at MC0A (P1.19 – above) and MC0B (P1.22 – below)

(P1.22 – below) at another time (the signals changed)

✓ If test 3-phase AC mode:

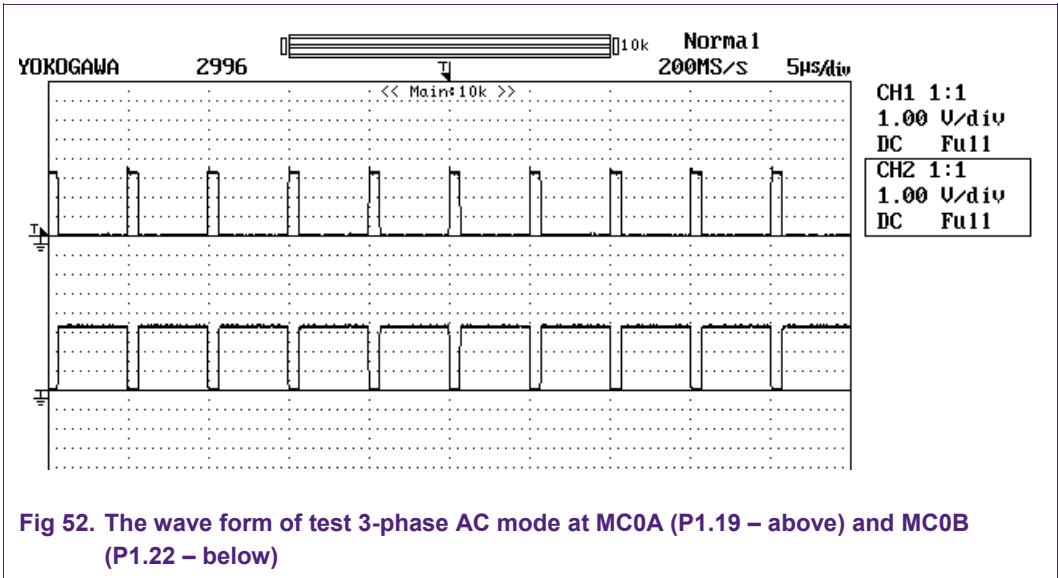


Fig 52. The wave form of test 3-phase AC mode at MC0A (P1.19 – above) and MC0B (P1.22 – below)

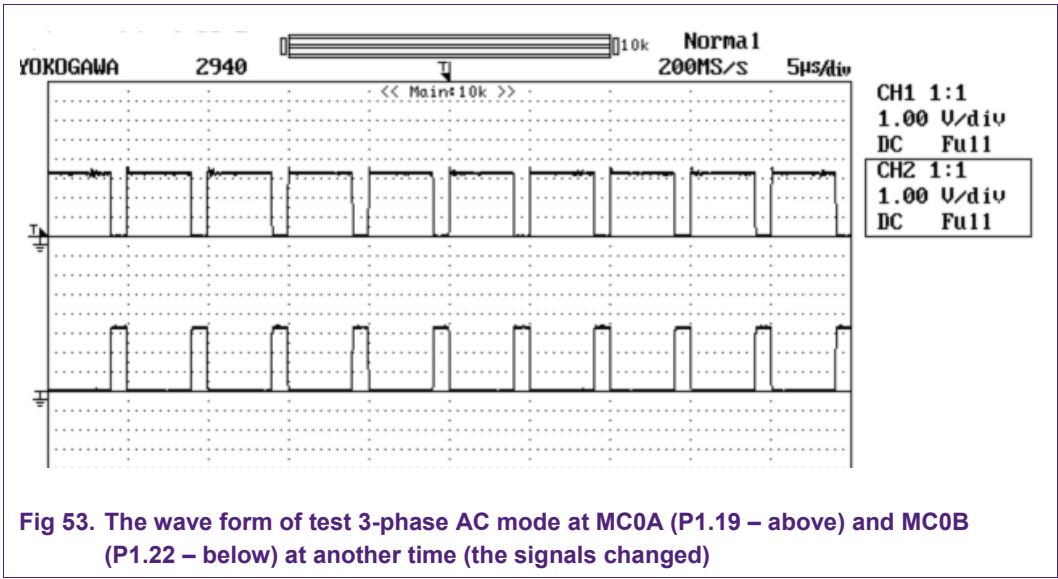


Fig 53. The wave form of test 3-phase AC mode at MC0A (P1.19 – above) and MC0B (P1.22 – below) at another time (the signals changed)

- Notes:** The duty cycle of the signals (P1.19 and P1.22) is changeable by time.
- The content on the serial terminal should be like this if using capture mode

The screenshot shows a Tera Term window titled "COM10:115200baud - Tera Term VT". The menu bar includes File, Edit, Setup, Control, Window, Resize, and Help. The main text area displays the following content:

```

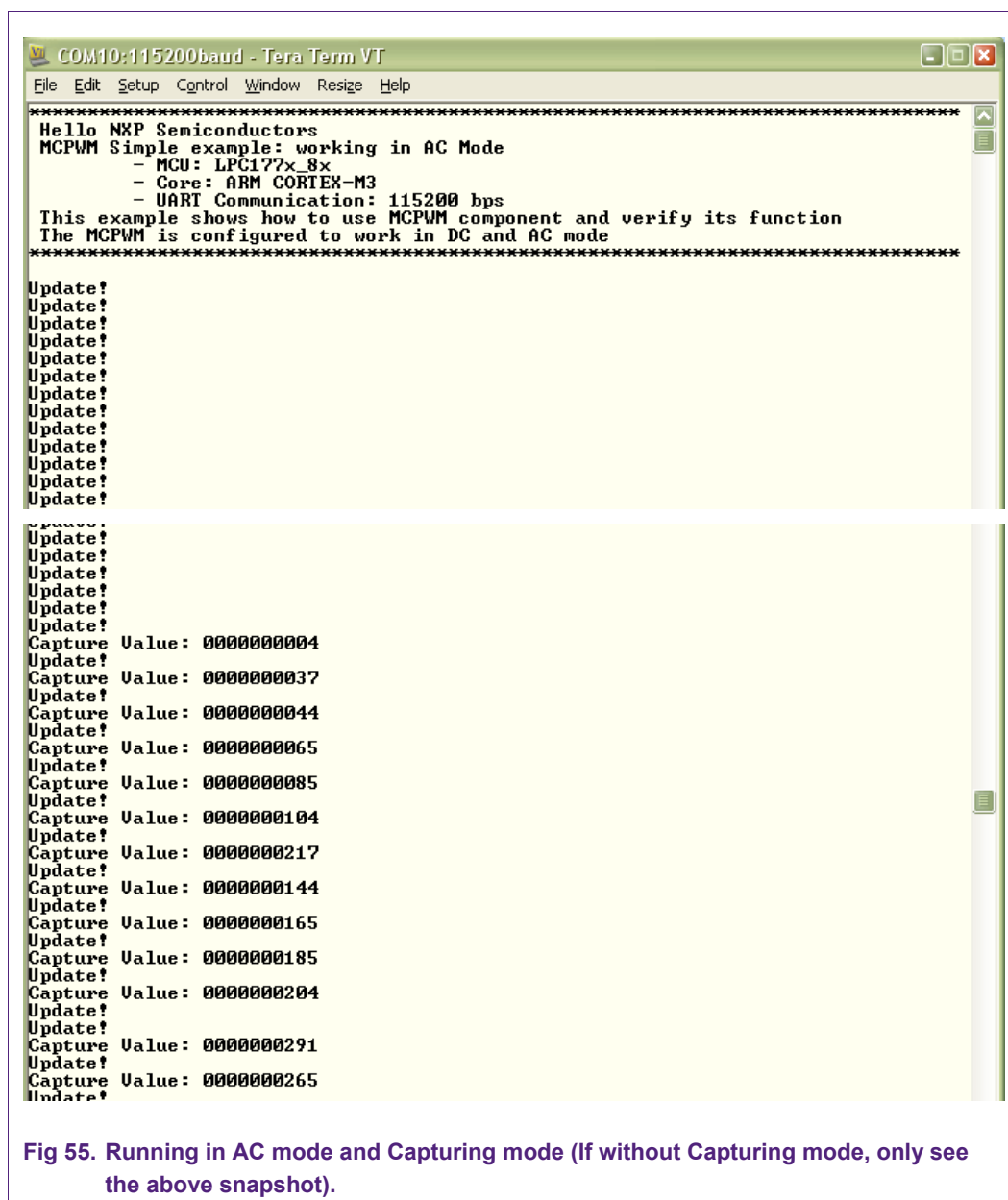
*****
Hello NXP Semiconductors
MCPWM Simple example: working in DC Mode
  - MCU: LPC177x_8x
  - Core: ARM CORTEX-M3
  - UART Communication: 115200 bps
This example shows how to use MCPWM component and verify its function
The MCPWM is configured to work in DC and AC mode
*****

Update!
Update!
Update!
Update!
Update!
Update!
Update!
Update!
Update!
Update!
Update!

Update!
Update!
Update!
Capture Value: 0000000007
Update!
Capture Value: 0000000006
Update!
Capture Value: 0000000007
Update!
Capture Value: 0000000003
Update!
Update!
Capture Value: 0000000003
Update!
Update!
Capture Value: 0000000002
Update!
Capture Value: 0000000002
Update!
Capture Value: 0000000002
Update!
Capture Value: 0000000007
Update!
Capture Value: 0000000002

```

The output shows a sequence of "Update!" messages, followed by a series of "Capture Value:" messages with hexadecimal values. The values are: 0000000007, 0000000006, 0000000007, 0000000003, 0000000003, 0000000002, 0000000002, 0000000002, and 0000000007.



3.15 NVIC (NESTED VECTORED INTERRUPT CONTROLLER)

3.15.1 Nvic_Priorities

3.15.1.1 Example description

Purpose

This example describes how to configure NVIC priority grouping for testing tail-chaining or Late-arriving of interrupt mode.

Knowledge and Process

This example uses 2 interrupts for IRQ channels. They are ADC interrupt and External Interrupt 0 (EINT0).

Setting `INT_MODE` macro to chose interrupt mode for test.

- In case that `INT_MODE = 0`: Tail-chaining interrupt testing
 EINT0 interrupt is assigned group-priority = 0, sub-priority = 2
 ADC interrupt is assigned group-priority = 0, sub-priority = 1
 Two IRQ channels has same group. So, new ISR coming can not pre-empt current interrupt while its execution even if new ISR has higher priority than current one.
- In case that `INT_MODE = 1`: Late-arriving interrupt testing.
 EINT0 interrupt is assigned group-priority = 0, sub-priority = 0
 ADC interrupt is assigned group-priority = 1, sub-priority = 0
 EINT0 interrupt has higher group-priority than ADC interrupt, so EINT0 interrupt can pre-empt ADC interrupt when it's executing.

In this example, EINT0 interrupt occurs when pressing button INT0 (on OEM board) or pull down the P2.10 to GND (on IAR board).

EINT0 interrupt will blink LED P1.18 10 (LED USB OTG) times.

ADC interrupt occurs periodically to take the ADC samples from the ADC potentiometer
 ADC interrupt will blink LED P0.13 (LED USB Host) 10 times

3.15.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Nvic_Priorities.c: Main program

3.15.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

This example does not need any jumper configuration.

Note:

- **LED 1** is P1.13 on IAR board or P1.18 on OEM board
- **LED 2** is P1.18 on IAR board or P0.13 on OEM board

3.15.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.15.1.5 Step to run

(1) Setting the example for expected purpose:

Testing purpose(s): it needs some pre-setting here in file `Nvic_Priorities.c`:

- If going to test TAIL CHAIN interrupt, define as below

```
#define INT_MODE 0
```

- If going to test LATE – ARRIVING interrupt, define as below

```
#define INT_MODE 1
```

(2) Build example

(3) Burn built image file (in hex) into the board (in case of ROM mode running)

(4) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)

(5) Do hardware configuration exactly following the above instructions.

(6) Reset the board (in case of ROM mode running) and Run the example on the test board.

(7) Testing actions and results:

- At the beginning, the ADC interrupt will normally happen and blink LED2 in 10 times per an interrupt occurrence.
- If EXT0 interrupt is occurred (by hitting INT0 button), LED1 will blink 10 times.
- Depending on the test, the phenomenon will be different. It is:
 - **Test tail-chaining interrupt mode:** When LED2 is blinking (because of ADC interrupt), if external interrupt 0 occurs, its interrupt will blink LED1 10 times right after an ADC interrupt is finished. It means EINT0 interrupt must wait for completion of ADC interrupt in order to start itself.
 - **Test late-arriving interrupt mode:** When LED2 is blinking (because of ADC interrupt), if external interrupt 0 occurs, it immediately forces ADC interrupt stop to start blinking LED1 10 times. It's without caring about the completion of the ADC interrupt. In this case, the INT0 interrupt does pre-empt the ADC interrupt
- After this INT0 is completed, the ADC interrupt will re-start to blink the LED2 continuously because its interrupt occurrence.

3.15.2 Nvic_VectorTableRelocation

3.15.2.1 Example description

Purpose

This example describes how to relocation vector table.

Knowledge and Process

Vector Table will be remapped at new address `VTOR_OFFSET`.

- In ROM mode: Vector Table will be initialized at 0x00000000
- In RAM mode: Vector Table will be initialized at 0x10000000

So, we need copy vector table from initialized address to new address.

Check vector remapping successful or not, use System Tick Timer interrupt to blink LEDs on board.

3.15.2.2 Directory contents

\EWARM:	includes EWARM (IAR) project and configuration files
\Keil:	includes RVMDK (Keil) project and configuration files

Nvic_VectorTableRelocation.c: Main program

3.15.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

This example does not need any jumper configuration.

Note:

- LED 1 is P1.13 on IAR board or P1.18 on OEM board
- LED 2 is P1.18 on IAR board or P0.13 on OEM board

3.15.2.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.15.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - See LED1 blinking continuously if vector remapping is done successfully
 - Check the below content on a PC serial terminal

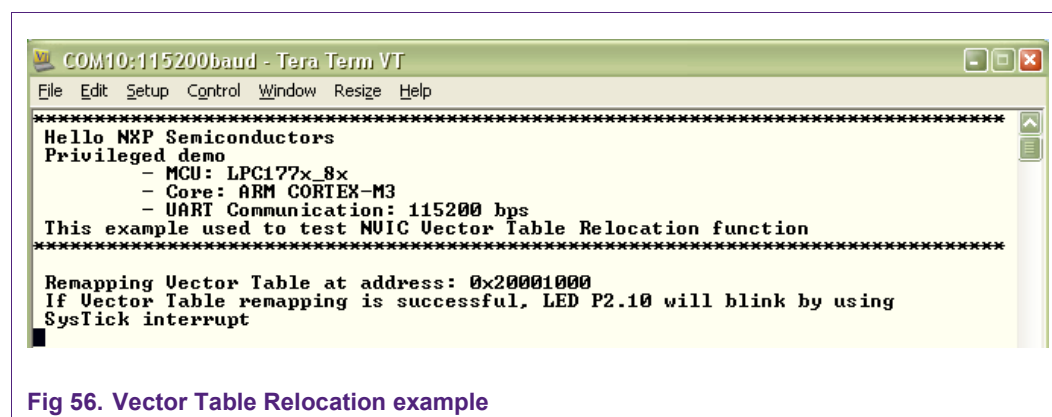


Fig 56. Vector Table Relocation example

3.16 PWM (PULSE WIDTH MODULATOR)

3.16.1 Single_Edge

3.16.1.1 Example description

Purpose

This example describes how to use PWM signal on 6 Channels in single edge mode

Process

This program illustrates the PWM signal on 6 Channels in single edge mode

Peripheral clock for PWM: PWM_PCLK = CCLK / 4 = 72MHz/4 = 18MHz

There is no pre-scale for PWM.

The PWM timer/counter clock is at 18MHz.

The base rate is set to 256

The base PWM frequency is at 18MHz/256 = 70.312 KHz (Period = ~14.22 microsecond)

Each PWM channel (1 to 6) will be configured as following:

- PWM1.1 = (10/256) (period = 0.21 microsecond) (P2.0)
- PWM1.2 = (40/256) (period = 0.83 microsecond) (P2.1)
- PWM1.3 = (70/256) (period = 1.45 microsecond) (P2.2)
- PWM1.4 = (100/256) (period = 2.07 microsecond) (P2.3)
- PWM1.5 = (130/256) (period = 2.69 microsecond) (P2.4)
- PWM1.6 = (160/256) (period = 3.31 microsecond) (P2.5)

Using Oscilloscope to observe the PWM signals

3.16.1.2 Directory contents

- \EWARM:** includes EWARM (IAR) project and configuration files
- \Keil:** includes RVMDK (Keil) project and configuration files

Pwm_SingleEdge: Main program

3.16.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

PWM Pin Map

	OEM Board
PWM0:	
PWM0.1 (channel 1): P1.2	J5.27
PWM0.2 (channel 2): P1.3	J3.29
PWM0.3 (channel 3): P1.5	J5.28
PWM0.4 (channel 4): P1.6	J3.30
PWM0.5 (channel 5): P1.7	J5.29
PWM0.6 (channel 6): P1.11	J3.31
PWM1:	
PWM1.1 (channel 1): P2.0	J4.5

PWM1.2 (channel 2): P2.1	J4.7
PWM1.3 (channel 3): P2.2	J4.6
PWM1.4 (channel 4): P2.3	J4.8
PWM1.5 (channel 5): P2.4	J3.16
PWM1.6 (channel 6): P2.5	J3.14

PWM Pins to be observed: are all the pin of and PWM number

3.16.1.4 Running mode

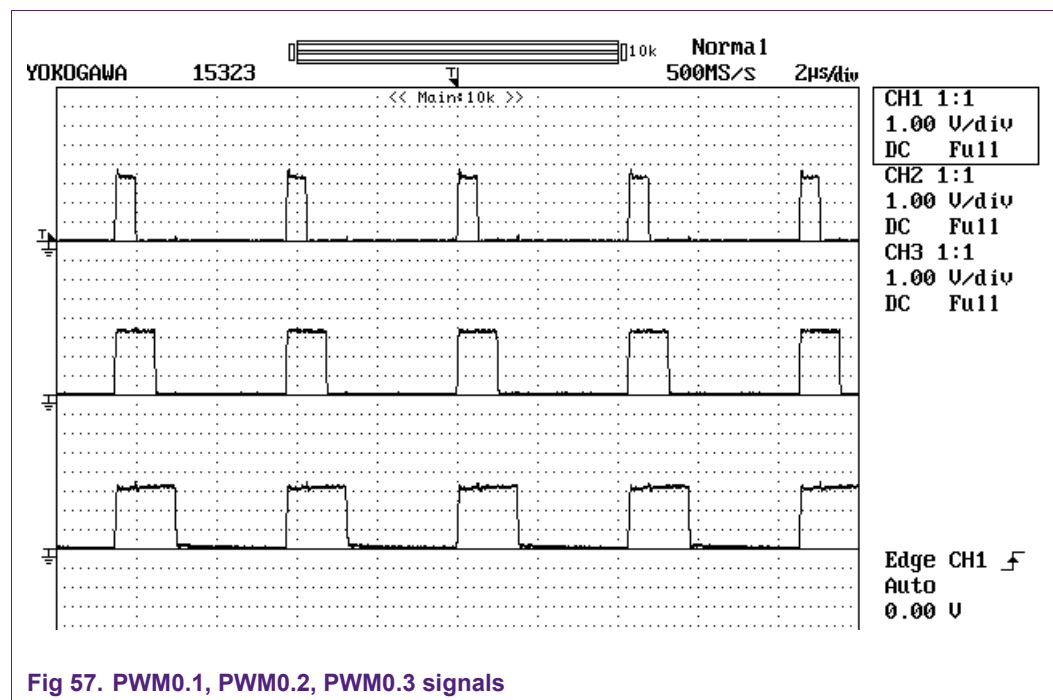
This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.16.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:

Observe the wave form from the PWM pins (P2.0, P2.1, P2.2, P2.3, P2.4, and P2.5) by oscilloscope.



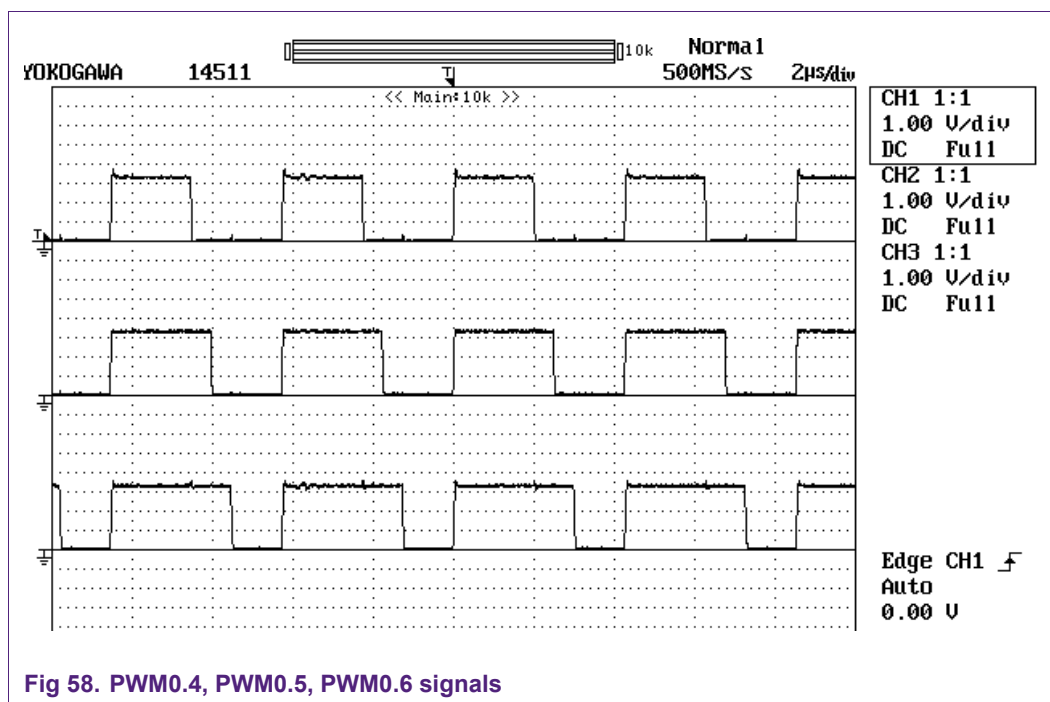


Fig 58. PWM0.4, PWM0.5, PWM0.6 signals

3.16.2 Pwm_DualEdge

3.16.2.1 Example description

Purpose

This example describes how to generate PWM signal on 3 channels in both edge mode and single mode.

Knowledge and Process

This program illustrates the PWM signal on 3 Channels in both edge mode and single mode.

Peripheral clock for PWM: $\text{PWM_PCLK} = \text{CCLK} / 2 = 120\text{MHz} / 2 = 60\text{MHz}$

There is no pre-scale for PWM. The PWM timer/counter clock is at 48MHz.

The base rate is set to 100

The base PWM frequency is at $60\text{MHz} / 100 = 600\text{ KHz}$.

Each PWM channel will be configured as following:

- Channel 2: Double Edge (P2.1)
- Channel 4: Double Edge (P2.5)
- Channel 5: Single Edge (P2.6)

The Match register values are as follows:

- MR0 = 100 (PWM rate)
- MR1 = 41, MR2 = 78 (PWM2 output)
- MR3 = 53, MR4 = 27 (PWM4 output)
- MR5 = 65 (PWM5 output)

PWM Duty on each PWM channel:

- Channel 2: Set by match 1, Reset by match 2.
- Channel 4: Set by match 3, Reset by match 4.
- Channel 5: Set by match 0, Reset by match 5.

Using Oscilloscope to observe the PWM signals

3.16.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Pwm_DualEdge.c: Main program

3.16.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

PWM Pin Map

OEM Board

PWM0:

PWM0.1 (channel 1): P1.2	J5.27
PWM0.2 (channel 2): P1.3	J3.29
PWM0.3 (channel 3): P1.5	J5.28
PWM0.4 (channel 4): P1.6	J3.30
PWM0.5 (channel 5): P1.7	J5.29
PWM0.6 (channel 6): P1.11	J3.31

PWM1:

PWM1.1 (channel 1): P2.0	J4.5
PWM1.2 (channel 2): P2.1	J4.7
PWM1.3 (channel 3): P2.2	J4.6
PWM1.4 (channel 4): P2.3	J4.8
PWM1.5 (channel 5): P2.4	J3.16
PWM1.6 (channel 6): P2.5	J3.14

PWM Pins observed

Monitor PWM wave signal on these pin

- PWMx.2 (channel 2): P2.1_PWM1.2 or P1.3_PWM0.2
- PWMx.4 (channel 4): P2.3_PWM1.4 or P1.6_PWM0.4
- PWMx.5 (channel 5): P2.4_PWM1.5 or P1.7_PWM0.5

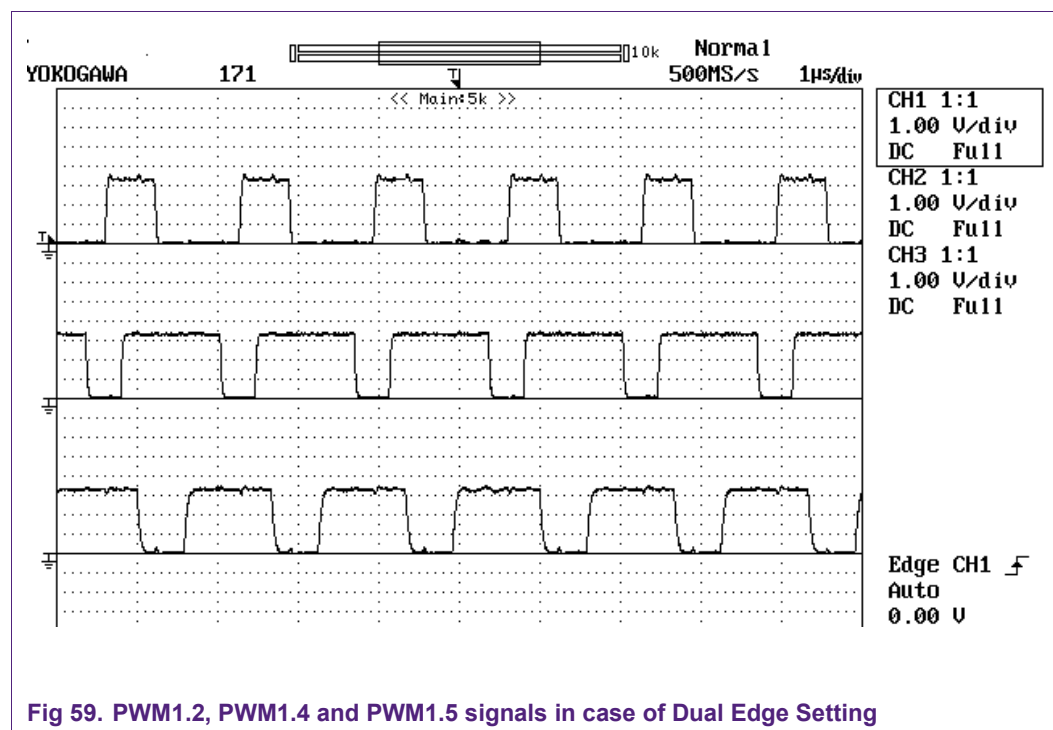
3.16.2.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.16.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - Observe the wave form from the PWM pins by oscilloscope.



3.16.3 Pwm_MatchInterrupt

3.16.3.1 Example description

Purpose

This example describes how to use PWM Match function in interrupt mode.

Knowledge and Process

This program illustrates the PWM signal on 6 Channels in single edge mode

Peripheral clock for PWM: $PWM_PCLK = CCLK / 2 = 96MHz / 2 = 48MHz$

There is no pre-scale for PWM. The PWM timer/counter clock is at 48MHz.

The base rate is set to 256

The base PWM frequency is at $48MHz / 256 = 187.5 KHz$ (Period = ~5.3 microsecond)

Each PWM channel (1 to 6) will be configured as following:

- PWM1.1 = (10/256) (period = 0.21 microsecond) (P2.0)
- PWM1.2 = (40/256) (period = 0.83 microsecond) (P2.1)
- PWM1.3 = (70/256) (period = 1.45 microsecond) (P2.2)
- PWM1.4 = (100/256) (period = 2.07 microsecond) (P2.3)
- PWM1.5 = (130/256) (period = 2.69 microsecond) (P2.4)
- PWM1.6 = (160/256) (period = 3.31 microsecond) (P2.5)

Using Oscilloscope to observe the PWM signals

Here, PWM1.1 value keeps changing; it will be increased by the time from 0 to 256 period and restart. Match interrupt for channel 0 is set when timer of PWM reach to 256 (value of channel 0 match). Then an interrupt for matching will be invoked and used to update the value of *PWM1.1*.

The time duration for every updating of this value is 4096 match interrupts or:

$$\text{Period} * 4096 = 5.3 * 4096 = 21708.8 \text{ (microsecond)}$$

And this value will be reset to 0 after:

$$\text{Period} * 4096 * 256 = 5557452.8 \text{ (microsecond)} = \sim 5.5 \text{ (second)}$$

3.16.3.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Pwm_MatchInterrupt.c: Main program

3.16.3.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

PWM Pin Map

OEM Board

PWM0:

PWM0.1 (channel 1): P1.2	J5.27
PWM0.2 (channel 2): P1.3	J3.29
PWM0.3 (channel 3): P1.5	J5.28
PWM0.4 (channel 4): P1.6	J3.30
PWM0.5 (channel 5): P1.7	J5.29
PWM0.6 (channel 6): P1.11	J3.31

PWM1:

PWM1.1 (channel 1): P2.0	J4.5
PWM1.2 (channel 2): P2.1	J4.7

PWM1.3 (channel 3): P2.2 J4.6

PWM1.4 (channel 4): P2.3 J4.8

PWM1.5 (channel 5): P2.4 J3.16

PWM1.6 (channel 6): P2.5 J3.14

PWM Pins to be observed: are all the pin of and PWM number

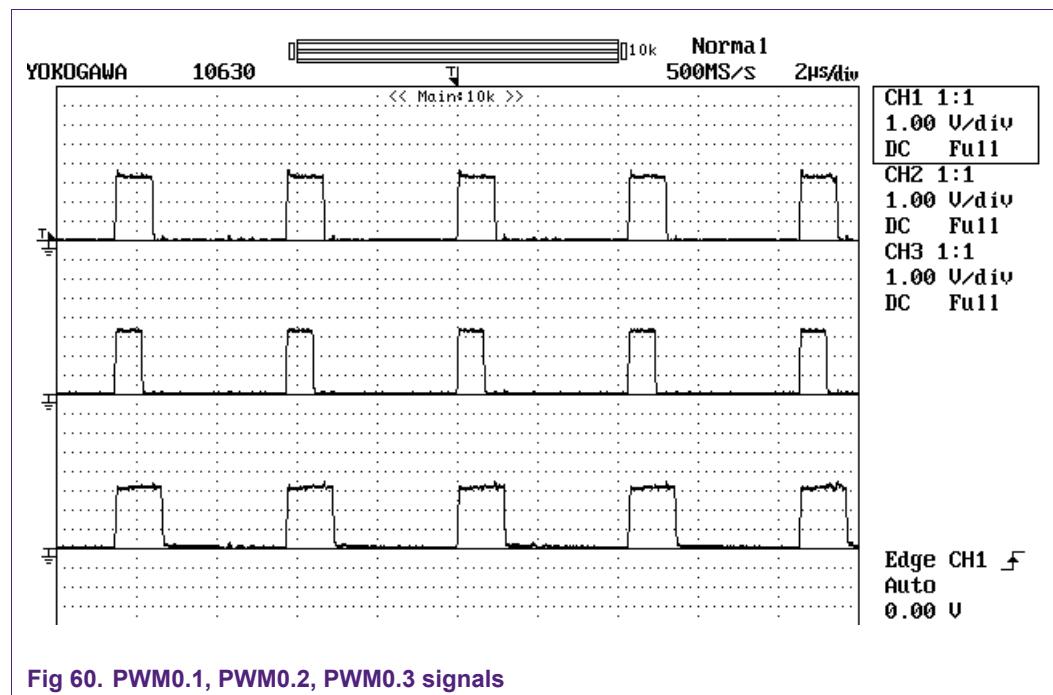
3.16.3.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.16.3.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - Observe the wave form from the PWM pins by oscilloscope.



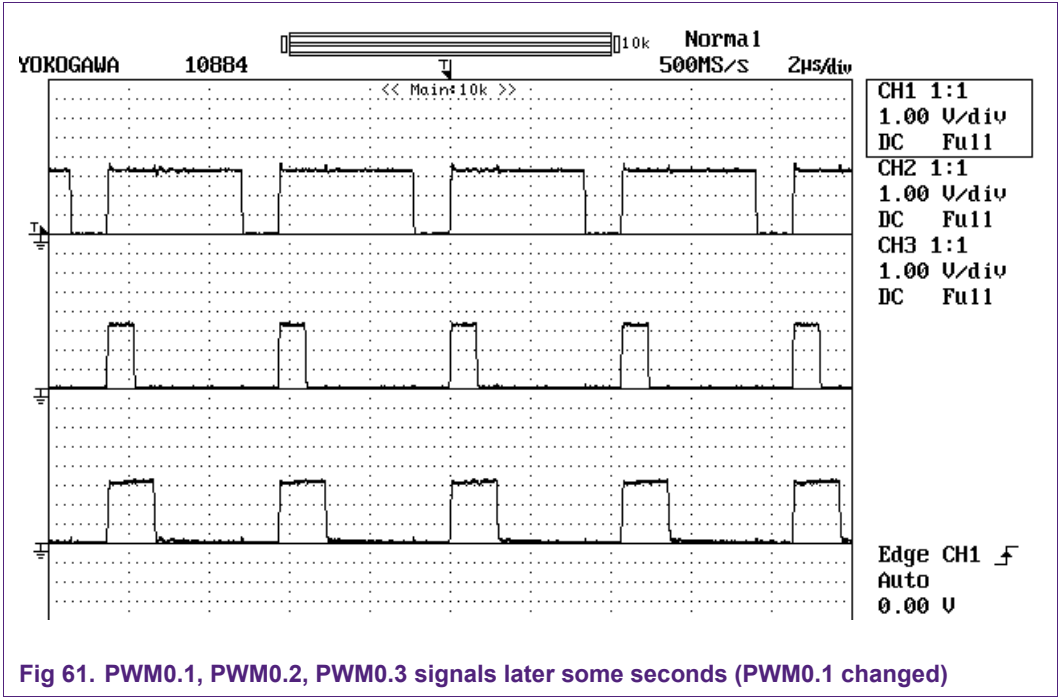


Fig 61. PWM0.1, PWM0.2, PWM0.3 signals later some seconds (PWM0.1 changed)

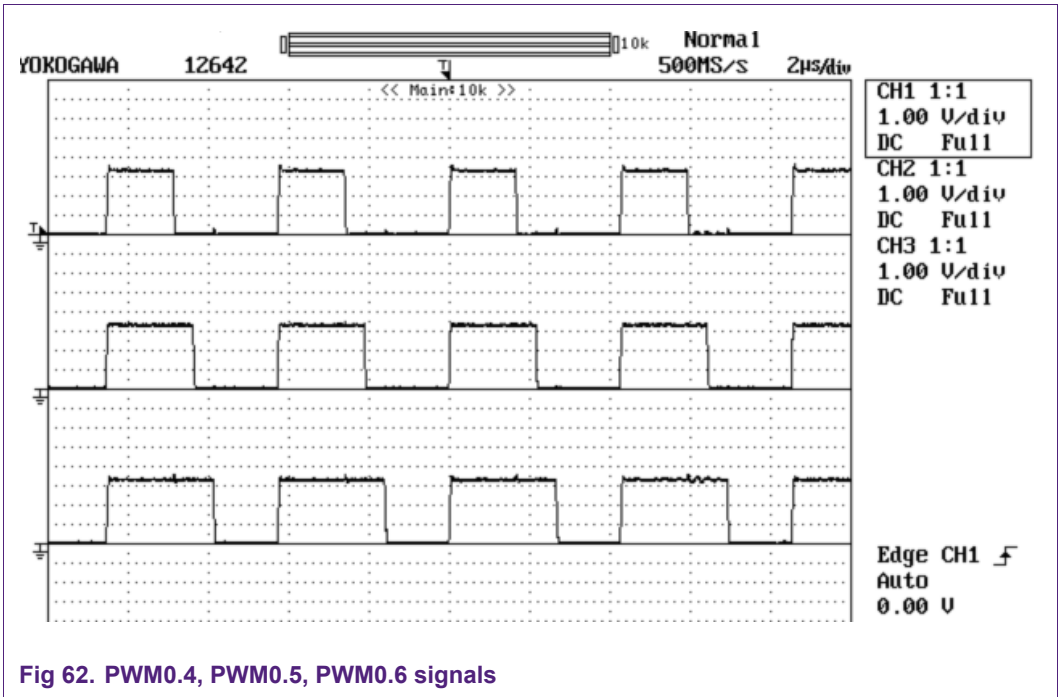


Fig 62. PWM0.4, PWM0.5, PWM0.6 signals

3.17 PWR (POWER MANAGEMENT)

3.17.1 Pwr_Sleep

3.17.1.1 Example description

Purpose

This example describes how to enter system in sleep mode and wake-up by using external interrupt

Process

First the program is in normal mode.

Once receiving '1' character from serial display, the system call `CLKPWR_Sleep()` and enter sleep mode.

To wake up the system, we should cause the External Interrupt 0. Then UART displaying will be continued.

3.17.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Pwr_Sleep.c: Main program

3.17.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

3.17.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.17.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - Press '1' on PC serial terminal for the system to enter Sleep mode.
 - Press External Interrupt 0 to wakeup the system.
 - See the content on serial terminal, it should be like this:

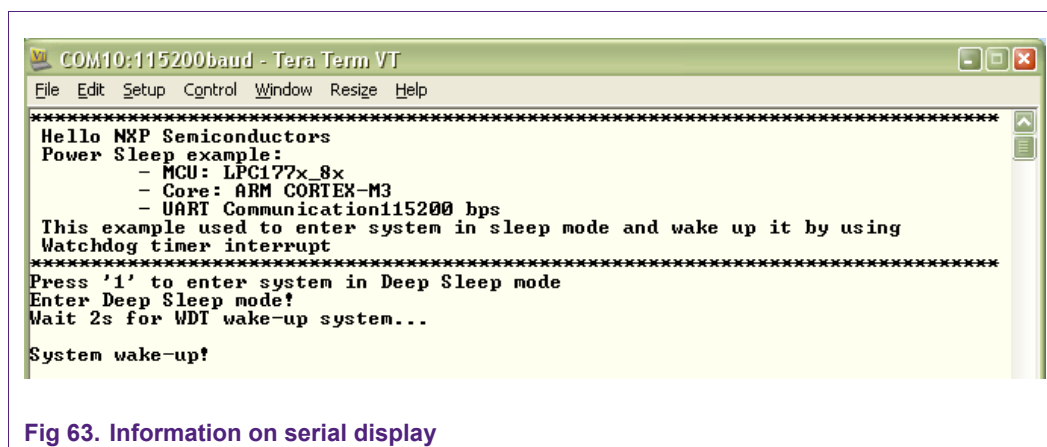


Fig 63. Information on serial display

3.17.2 Pwr_DeepSleep

3.17.2.1 Example description

Purpose

This example describes how to enter system in deep sleep mode and wake-up by using external interrupt

Process

First the program is in normal mode.

When receive '1' character from serial display, the system call `CLKPWR_DeepSleep()` function to enter deep sleep mode.

Once external interrupt 0 is generated, the system will be waken up and display a notice on serial display screen.

In this example, EINT0 interrupt occurs when pressing button INT0 (on OEM board) or pull down the P2.10 to GND (on IAR board).

3.17.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Pwr_DeepSleep.c: Main program

3.17.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

➤ LPC1788 OEM board rev A connects with OEM base board rev A

After Deep Sleep, to wake up, it's needed to press **SW6** button

➤ LPC1788 IAR Start Kit Rev.B

After Deep Sleep, to wake up, it's needed to connect **ISP_E[2]** to **GND**

3.17.2.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.17.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - Press '1' on PC serial terminal for the system to enter Deep PowerDown mode.
 - To wake up the IC: we should:
 - o To generate EINT0 interrupt
 - o Or press "RESET" button to wake it up immediately.
 - See the content on serial terminal, it should be like this:

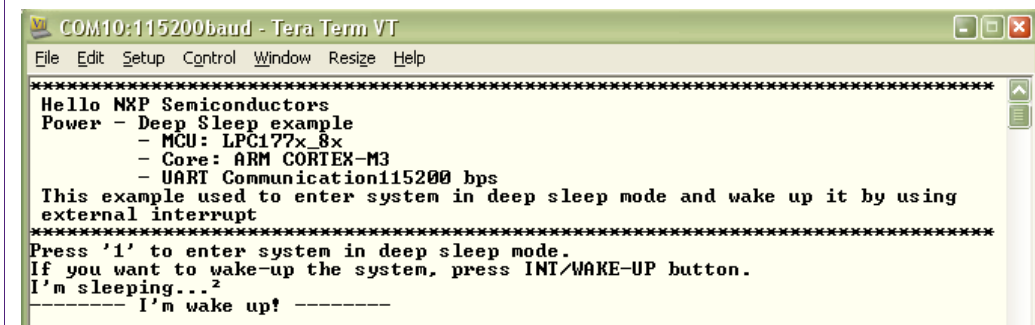


Fig 64. Information on serial display

3.17.3 Pwr_PowerDown

3.17.3.1 Example description

Purpose

This example describes how to enter system in PowerDown mode and wake-up it by using NMI (Non-Maskable Interrupt)

Process

NMI (Non-Maskable Interrupt) is the highest priority interrupt; it takes effect as soon as it registers. Connection logic 1 to the pin will cause the NMI to be processed.

P2.10 pin is configured as NMI pin.

Before resetting the board, it's needed to ensure that P2.10 pin pulls HIGH. This is to avoid that the IC executes in ISP program mode.

After resetting the board, pin 2.10 should pull LOW to disable its interrupt of NMI. This is for ability to force the system into PowerDown mode.

The call of `CLKPWR_PowerDown()` will make the system enter PowerDown state.

Note: In this time, NMI pin must be connected with ground to disable NMI interrupt).

While the system is powering down, if pulling up NMI to HIGH, NMI occurs and wakes-up system.

3.17.3.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Pwr_PowerDown.c: Main program

3.17.3.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not require any jumper configuration or link.

Note: OEM board use SW6 as NMI pin (P2.10) while IAR board use ISP_E[2]

3.17.3.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.17.3.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - First, sure that NMI pin (P2.10) has logic 1.
 - Press "RESET" button to run example.

```

*****
Hello NXP Semiconductors
Power - Power Down example:
  - MCU: LPC177x_8x
  - Core: ARM CORTEX-M3
  - UART Communication115200 bps
This example used to enter system in PowerDown mode and wake up it by using
NMI <Non-Maskable Interrupt>
*****

```

Fig 65. Pwr-PowerDown welcome screen

- Connect P2.10 with ground to disable NMI interrupt

```

*****
Hello NXP Semiconductors
Power - Power Down example:
  - MCU: LPC177x_8x
  - Core: ARM CORTEX-M3
  - UART Communication115200 bps
This example used to enter system in PowerDown mode and wake up it by using
NMI <Non-Maskable Interrupt>
*****
Press '1' to enter system in PowerDown mode

```

Fig 66. NMI-PowerDown after connect NMI pin with ground

- Press '1' on PC serial terminal for the system to enter PowerDown mode.

```

Enter PowerDown mode...
Press INT0 button to wake-up system

```

Fig 67. Enter system to deep-sleep mode

- After sleeping..., pull up P2.10 to generate NMI interrupt to wake-up system.

```

*****
Hello NXP Semiconductors
Power - Power Down example:
  - MCU: LPC177x_8x
  - Core: ARM CORTEX-M3
  - UART Communication115200 bps
This example used to enter system in PowerDown mode and wake up it by using
NMI <Non-Maskable Interrupt>
*****
Press '1' to enter system in PowerDown mode
Enter PowerDown mode...
Press INT0 button to wake-up system
System waked-up!

```

Fig 68. PowerDown example serial whole content

Note: If it has not display above notice, P2.10 should pull-down to disable NMI interrupts for UART displaying.

3.17.4 Pwr_DeepPowerDown

3.17.4.1 Example description

Purpose

This example describes how to enter system in Deep PowerDown mode and wake-up by using RTC (Real-time clock) interrupt

Knowledge and Process

When system enters in Deep PowerDown mode, it just can be waked up when an external reset signal is applied, or the RTC interrupt is enabled and an RTC interrupt is generated.

In this case, we can use RTC interrupt or hit RESET button to wake-up system

RTC configure:

- Alarm time: 5s

So, after each 5s, RTC will generate Alarm interrupt to wake up system.

3.17.4.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Pwr_DeepPowerDown.c: Main program

3.17.4.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

3.17.4.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.17.4.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - Press '1' on PC serial terminal for the system to enter Deep PowerDown mode.
 - To wake up the IC: we should
 - o Wait 5s for RTC waking up our system automatically
 - o Or press "RESET" button to wake it up immediately.
 - See the content on serial terminal, it should be like this:

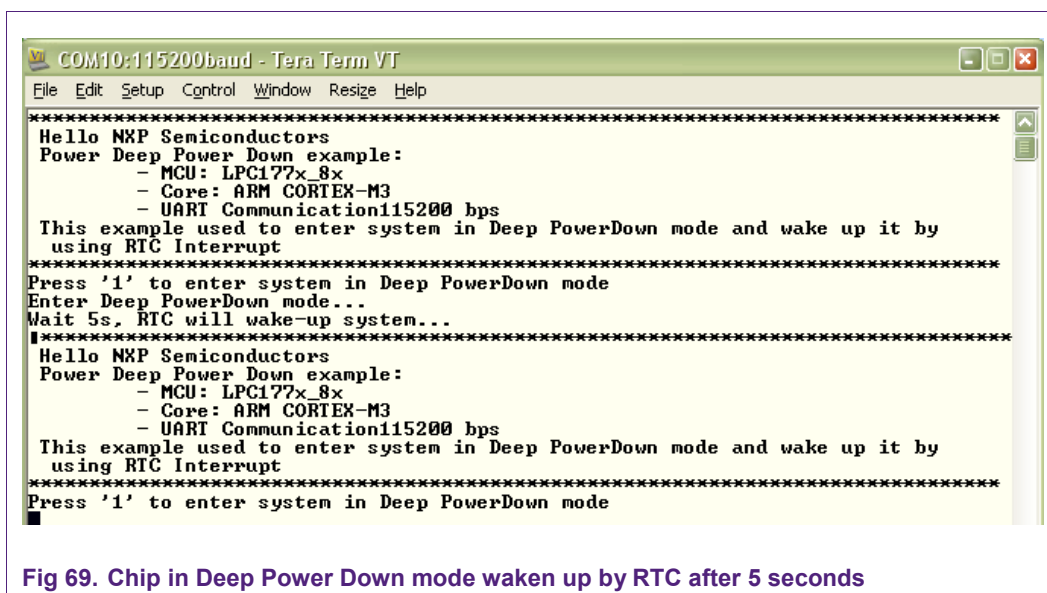


Fig 69. Chip in Deep Power Down mode waken up by RTC after 5 seconds

3.18 QEI (QUADRATURE ENCODER INTERFACE)

3.18.1 QEI_Velo

3.18.1.1 Example description

Purpose

This example describes how to use QEI (Quadrature Encoder Interface) component for velocity calculation

Knowledge and Process

This is just a simple QEI demo for demonstrate QEI operation. It needs 2 signals that input to QEI_PhA and QEI_PhB channel of QEI component. About this, there're 2 possibilities:

- Apply directly 2 signals outside (the board, the IC, the code,...) to the pins of these 2 channel of QEI
- Enable `VIRTUAL_QEI_SIGNAL` flag to 1 to generate virtual signals by the code inside this example code. The virtual signals will come to QEI_PhA and QEI_PhB pin for QEI operations.

This example uses a timer with matching function for interrupt to make such signals. These signals are outputted by timer will input to above QEI pin for its capturing function.

The source of the virtual signals is from:

- LPC1788 OEM board rev A connects with OEM base board rev A
 - From *P1.18* to *QEI_PhA*
 - From *P1.19* to *QEI_PhB*
- LPC1788 IAR Start Kit Rev.B
 - From *P0.5* to *QEI_PhB*
 - From *P0.7* to *QEI_PhA*

In this case, a “virtual encoder” that has these following parameters:

- *Encoder type*: Quadrature encoder
- *Max velocity*: MAX_VEL (Round Per Minute)
- *Encoder Resolution*: ENC_RES (Pulse Per Round)

The calculated frequency is:

$$Freq = (MAX_VEL * ENC_RES * COUNT_MODE) / 60 \text{ (Hz)}$$

The timer therefore should be set to tick every cycle $T = 1/Freq$ (second)

3.18.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Qei_Velo.c: Main program

3.18.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

QEI Pins Map:

	OEM Brd
P1.18	J5.31
P1.19	J3.33
P1.20	J5.32
P1.23	J3.35

These jumpers, links must be configured as follows:

➤ LPC1788 OEM board rev A connects with OEM base board rev A

- ✓ P1.18 (**J3.31**) with P1.20 (**J3.32**) – for QEI_PhA
- ✓ P1.19 (**J1.33**) with P1.23 (**J1.35**) – for QEI_PhB

➤ LPC1788 IAR Start Kit Rev.B

- ✓ P0.5 (**EXT-5**) with P1.20 (QE14 at QEI connector) - QEI_PhA
- ✓ P0.7 (**EXT-7**) with P1.23 (QE15 at QEI connector) - QEI_PhB

3.18.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.18.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)

- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - See the content on serial terminal, it should be like this:

```

COM10:115200baud - Tera Term VT
File Edit Setup Control Window Resize Help
*****
Hello NXP Semiconductors
QEI Velocity example:
- MCU: LPC177x_8x
- Core: ARM CORTEX-M3
- UART Communication: 115200 bps
This example use Quadrature Encoder Interface module to calculate velocity
*****
Speed will be sampled every each 0000250000 us
This value will be accumulated to display as RPM after every each 00003000000 us
Init the Virtual Signal
Initializing Virtual QEI signal...
Direction has changed: 1
Start the loop
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Direction has changed: 0
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Direction has changed: 1
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Direction has changed: 0
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Direction has changed: 1
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM
Sampling Speed: 0000000599 RPM

```

Fig 70. Information on serial display

Notes: When removing jumper between P1.18 - P1.20/P1.19 - P1.23, direction changing will be occurred.

3.19 RTC (REAL TIME CLOCK)

3.19.1 Rtc_Alarm

3.19.1.1 Example description

Purpose

This example describes how to generate interrupt in Second Counter Increment Interrupt (1 second) and generate Alarm interrupt after every 10 seconds.

Process

After initializing RTC, set current time for RTC with this value:

6:45:00PM, 2011-03-25

And set Alarm time at 10s

RTC is set generate interrupt each second, since `ISR RTC_IRQHandler()` will be called after every 1 second to get time from RTC registers and display on serial screen.

It all checks if alarm match interrupt occurs or not. (This interrupt just only occurs at 10s).

So after 10 seconds, alarm interrupt occurs and a notice sentence will be written on serial display screen.

3.19.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Rtc_Alarm.c: Main program

3.19.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

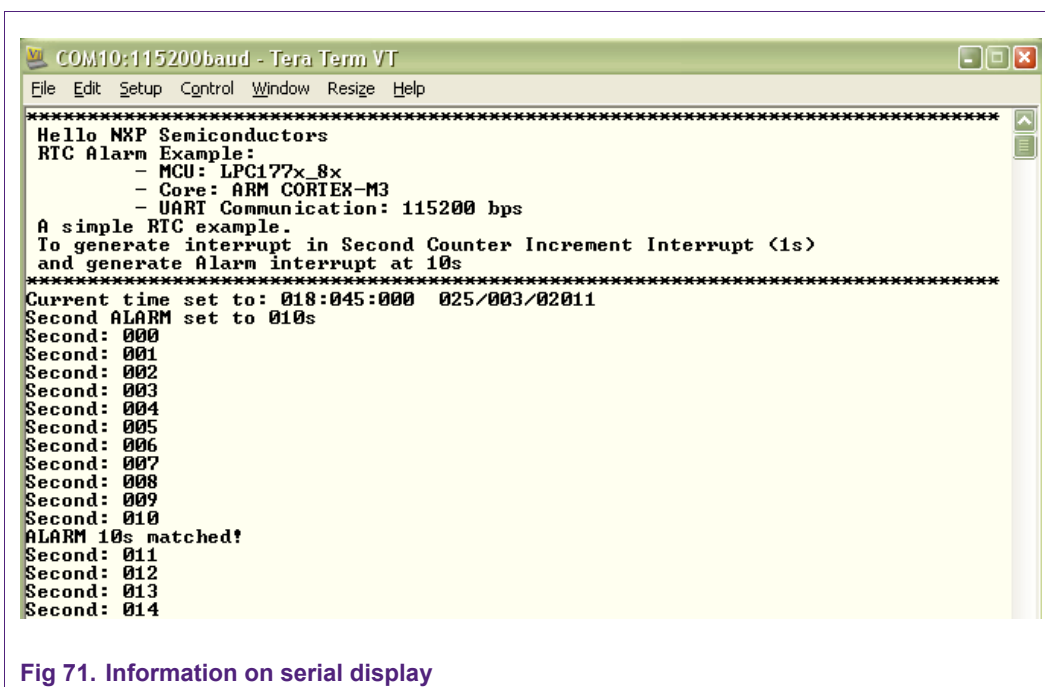
3.19.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.19.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - See the content on serial terminal, it should be like this:



```
COM10:115200baud - Tera Term VT
File Edit Setup Control Window Resize Help
*****
Hello NXP Semiconductors
RTC Alarm Example:
  - MCU: LPC177x_8x
  - Core: ARM CORTEX-M3
  - UART Communication: 115200 bps
A simple RTC example.
To generate interrupt in Second Counter Increment Interrupt <1s>
and generate Alarm interrupt at 10s
*****
Current time set to: 018:045:000 025/003/02011
Second ALARM set to 010s
Second: 000
Second: 001
Second: 002
Second: 003
Second: 004
Second: 005
Second: 006
Second: 007
Second: 008
Second: 009
Second: 010
ALARM 10s matched!
Second: 011
Second: 012
Second: 013
Second: 014
```

Fig 71. Information on serial display

3.19.2 Rtc_Calibration

3.19.2.1 Example description

Purpose

This example describes how to calibrate real-time clock

Process

The calibration logic can periodically adjust the time counter either by not incrementing the counter, or by incrementing the counter by 2 instead 1. This allow calibration the RTC oscillator under some typical voltage and temperature conditions without the need to externally trim the RTC oscillator

In this example:

- Calibration setting:
 - Calibration value = 5s;
 - Direction: Forward calibration
- Real-time clock setting:
 - enable incrementing second counter interrupt

The application is configured that after every 5 seconds, real-time clock will adjust automatically by incrementing the counter by 2 instead of 1. This can be observed the calibration process via serial display.

3.19.2.2 Directory contents

- | | |
|---------|---|
| \EWARM: | includes EWARM (IAR) project and configuration files |
| \Keil: | includes RVMDK (Keil) project and configuration files |

Rtc_Calibration.c: Main program

3.19.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

3.19.2.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.19.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - See the content on serial terminal to understand about the calibration for RTC, it should be like this:

```

COM10:115200baud - Tera Term VT
File Edit Setup Control Window Resize Help
*****
Hello NXP Semiconductors
RTC Calibration demo
- MCU: LPC177x_8x
- Core: ARM CORTEX-M3
- UART Communication: 115200 bps
This example describes how to calibrate RTC
*****
Second: 000
Second: 001
Second: 002
Second: 003
Second: 004
Second: 006
Second: 007
Second: 008
Second: 009
Second: 010
Second: 012
Second: 013
Second: 014
Second: 015
Second: 016
Second: 018
Second: 019
Second: 020
Second: 021
Second: 022
Second: 024
Second: 025

```

Fig 72. Information on serial display

3.20 SSP (SYNCHRONOUS SERIAL PORT)

3.20.1 Ssp_Dma

3.20.1.1 Example description

Purpose

This example describes how to use SSP (Synchronous Serial Port) peripheral with DMA support.

Knowledge and Process

SSP configuration:

- *CPHA* = 0: data is sampled on the first clock edge of SCK.
- *CPOL* = 0: SCK is active high
- Clock rate = 1MHz
- *DSS* = 8: 8 bits per transfer
- *MSTR* = 1: SSP operates in Master mode
- *FRF* = 0: SPI Frame format

This example configures SSP function in MASTER role with Loop-back mode (MOSI <-> MISO). The SSP will transfer a number of data byte (in DMA mode for both Tx and Rx channel).

GPDMA channel 0 and 1 are used in this example.

- GPDMA channel 0 is used to transfer data from source buffer to SSP peripheral.
- GPDMA Channel 1 is used to transfer data from SSP peripheral to destination buffer.

After transmission completed, two buffers will be compared, if they are not similar, the program will enter infinite loop and print error notice to serial display.

3.20.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Ssp_Dma.c: Main program

3.20.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD in .\.\.\BoardSupport\bsp.h` file for current working board.

SSP pins map:

	<i>OEM Board</i>	<i>IAR board</i>
<i>SSP0_MISO</i>	J5.20	
<i>SSP0_MOSI</i>	J3.23	
<i>SSP1_MISO</i>	J3.19	EXT-8

SSP1_MOSI J5.18 EXT-9

For this example working properly, MOSI must be connected with MISO pin.

Since, these jumpers must be configured as follows:

- **LPC1788 OEM board rev A connects with OEM base board rev A**
 - ✓ **J5.20** connects to **J3.23** if using SSP0
 - ✓ **J3.19** connects to **J5.18** if using SSP1
- **LPC1788 IAR Start Kit Rev.B**
 - ✓ Only working well with SSP1, **EXT-8** connects to **EXT-9**. Because other SSP pins are not available for use

3.20.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.20.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - The content on serial terminal should be like this:

Fig 73. Information on serial display

3.21 SysTick (SYSTEM TICK TIMER)

3.21.1 Systick_10msBase

3.21.1.1 Example description

Purpose

This example describes how to configure System Tick timer to generate interrupt each 10ms

Knowledge and Process

In this example, System Tick timer is clocked internal by the CPU clock

In this case, CPU clock = cclk = 120MHz

System Tick timer configure:

- time interval = 10ms
- enable System Tick interrupt

After each 10ms, System Tick will generate interrupt, interrupt service routine `SysTick_Handler()` will be invoke and toggle *P0.26* pin.

Use oscilloscope to monitor the signal from *P0.26* and measure time between falling and rising edge, it would be: 10ms.

3.21.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Systick_10msBase.c: Main program

3.21.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

Pins map:

	<i>OEM Board</i>	<i>IAR Board</i>
P0.26	J3.28	EXT-11

This example does not need any jumper, link configuration.

3.21.1.4 Running mode

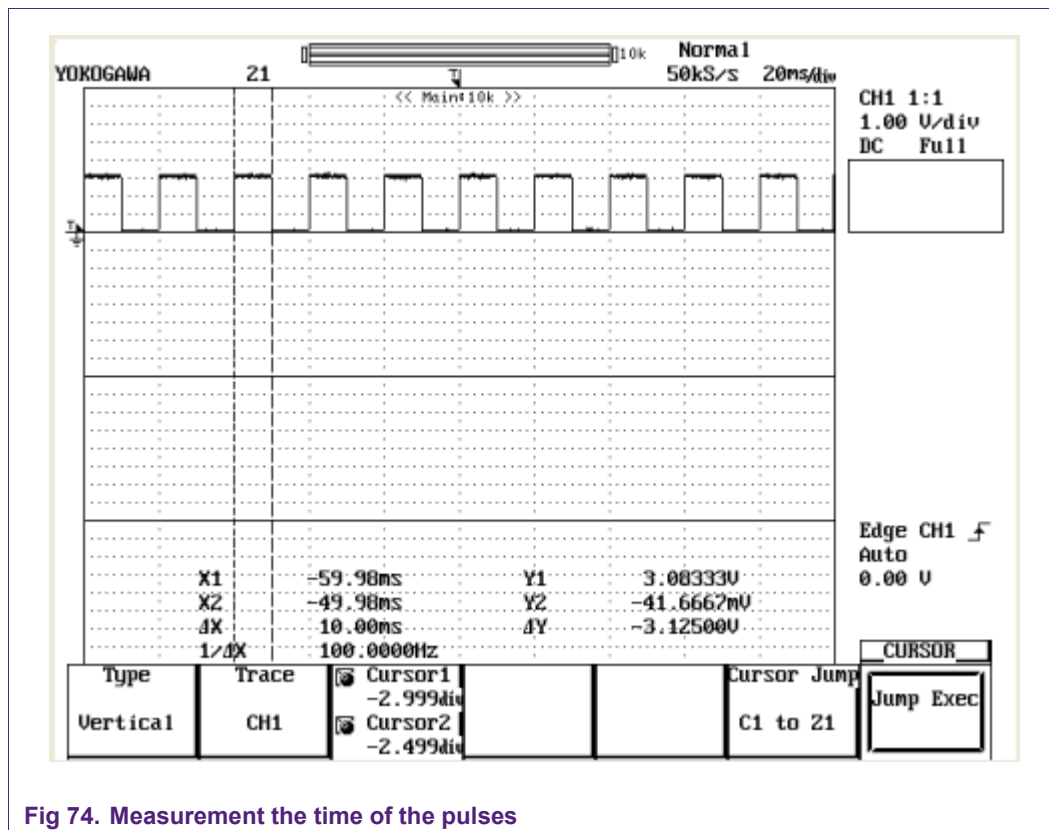
This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.21.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:

- Verify the signal from *P0.26* on the oscilloscope. It should be like below



3.21.2 SysTick_Stclk

3.21.2.1 Example description

Purpose

This example describes how to configure System Tick timer use external clock source STCLK

Knowledge and Process

In this example, System Tick timer is clocked by external clock STCLK

- STCLK supplied by MAT0.0 (P1.28) that generated by timer match channel 0
- STCLK frequency = 50kHz

Setting time interval = 10ms.

After each 10ms, System Tick will generate interrupt, interrupt service routine `SysTick_Handler()` will be invoke and toggle `P2.10` pin.

Use oscilloscope to observe signal on *P2.10* and measure time between falling and rising edge, it would be: 10ms

3.21.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Systick_Stclk.c: Main program

3.21.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

Required Connection

- **STCLK(P3.26)** connects with **MAT0.0 (P1.28)**
 - ➔ These jumpers must be configured as follows:
- **LPC1788 OEM board rev A connects with OEM base board rev A**
 - **J3.50 (P3.26)** connects to **J5.36 (P1.28)**
- **LPC1788 IAR Start Kit Rev.B**
 - There's no available pin for this connection. Since, it's unable to run well on this board.

3.21.2.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.21.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - Verify the signal from **P2.10** on the oscilloscope. It should be like below

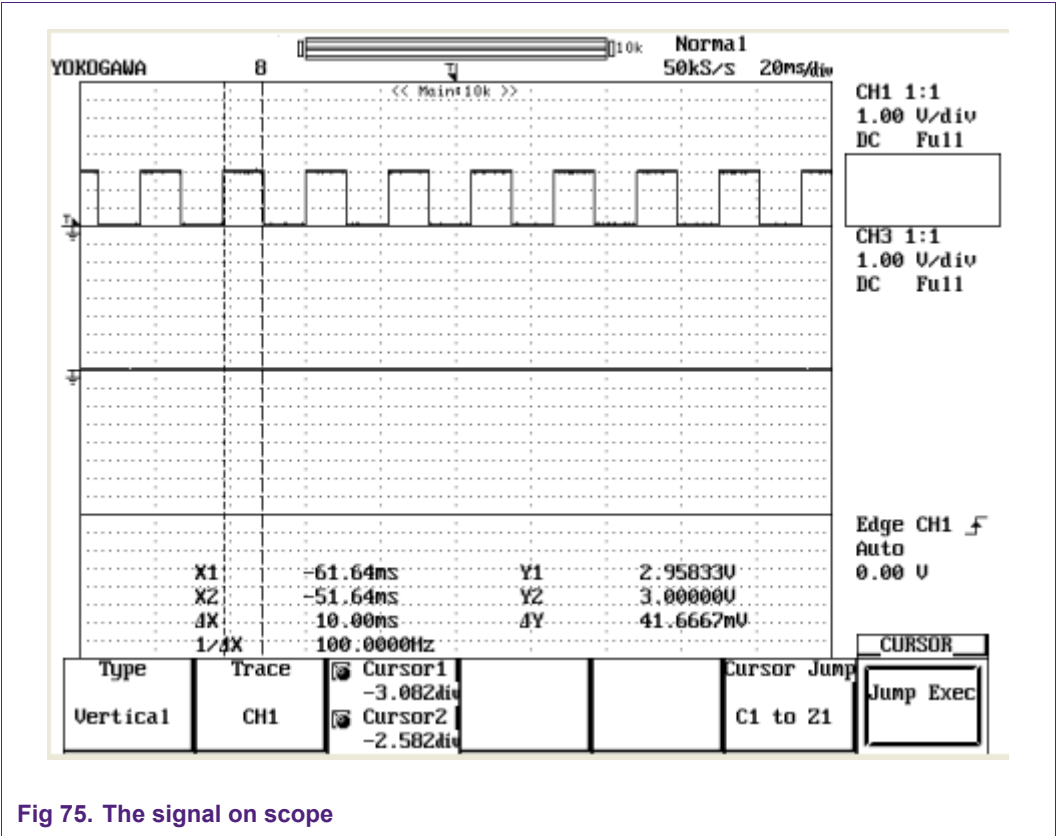


Fig 75. The signal on scope

3.22 TIMER

3.22.1 Timer_MatchInterrupt

3.22.1.1 Example description

Purpose

This example describes how to use Timer Match to generate specific time in interrupt mode.

Process

In this case, the specific time generated is 1s.

Timer configurations:

- Timer channel: 0
- Pre-scaler in microsecond value
- Pre-scaler value = 100us

Match configurations:

- Use channel 0, MR0
- Match value = 10000
 - Because timer tick = pre-scaler = 100us
 - So match time = 100 * 10000 = 1000000us = 1s
- Timer reset after match

- Not stop MR0 when match
- Toggle MR0.0 when match
 - Because match time = 1s
 - So MAT0 will be toggled at frequency = 1Hz
- Generate match interrupt

Whenever MR0 matches the value in TC register, match interrupt is occurs, the timer will invoke interrupt service routine to handle event. In this case, it will print a notice sentence into serial display, and the MAT channel of this timer will be toggling and timer will be reseted.

Note that: print data via UART can cause delay when match time set is too small.

3.22.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Timer_MatchInterrupt.c: Main program

3.22.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

These jumpers must be configured as follows:

➤ LPC1788 OEM board rev A connects with OEM base board rev A

All timers can be used on this board. But the example is planned to use Timer 0 and its Match 0 **P1.28 (J5.36)** pin for the above actions.

➤ LPC1788 IAR Start Kit Rev.B

Only Capture 0 Channel of Timer 2 (T2_CAP0) is ready for using on this board. Since, this example can only try with Timer 2 with its Match 0. It is **P0.6 (EXT-6)**

3.22.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.22.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.

(6) Testing actions and results:

- See the wave form on the oscilloscope from pin P1.28 (MAT0.0 on OEM board) or P0.6 (MAT2.0 on IAR board)
- Check the below content on a PC serial terminal

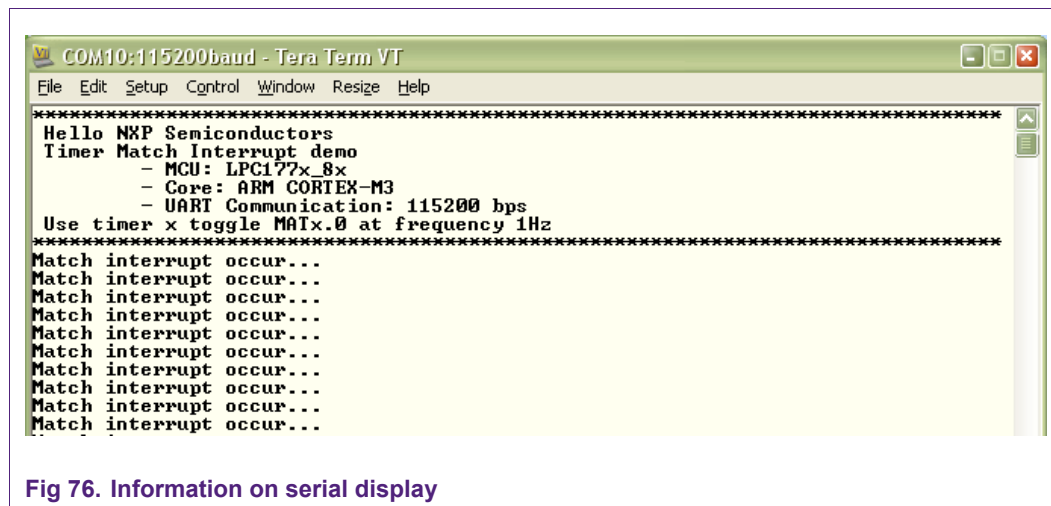


Fig 76. Information on serial display

3.22.2 Timer_Capture**3.22.2.1 Example description****Purpose**

This example describes how to use Capture Timer function.

Process

This example sets up a Timer take a snapshot of the timer value. This is done once there's a transition on input CAP signal of the current using timer.

Timer configuration:

- Pre-scaler in microsecond value
- Pres-caler value = 1000000us = 1s
- Use channel, CAPn.0
- Enable capture both on rising and falling edge
- Generate capture interrupt

Whenever capture interrupt occurs, TIMER interrupt service routine will be invoke to get captured time and display it into serial display

It's needed to change connection to the CAP channel between GND and VCC whenever we want to capture time.

3.22.2.2 Directory contents

- \EWARM:** includes EWARM (IAR) project and configuration files
- \Keil:** includes RVMDK (Keil) project and configuration files

Timer_Capture.c: Main program

3.22.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

These jumpers must be configured as follows:

➤ **LPC1788 OEM board rev A connects with OEM base board rev A**

All timers can be used on this board. But the example is planned to use Timer 0 and its Capture 0 **P1.26 (J5.35)** pin for the above actions.

➤ **LPC1788 IAR Start Kit Rev.B**

Only Capture 0 Channel of Timer 2 (T2_CAP0) is ready for using on this board. Since, this example can only try with Timer 2 with its capture 0. It is **P0.4 (EXT-4)**

3.22.2.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.22.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - Transition connection of the capture channel (P1.26 or P0.4) between GND and VCC whenever want to capture time.
 - Check the below content on a PC serial terminal:

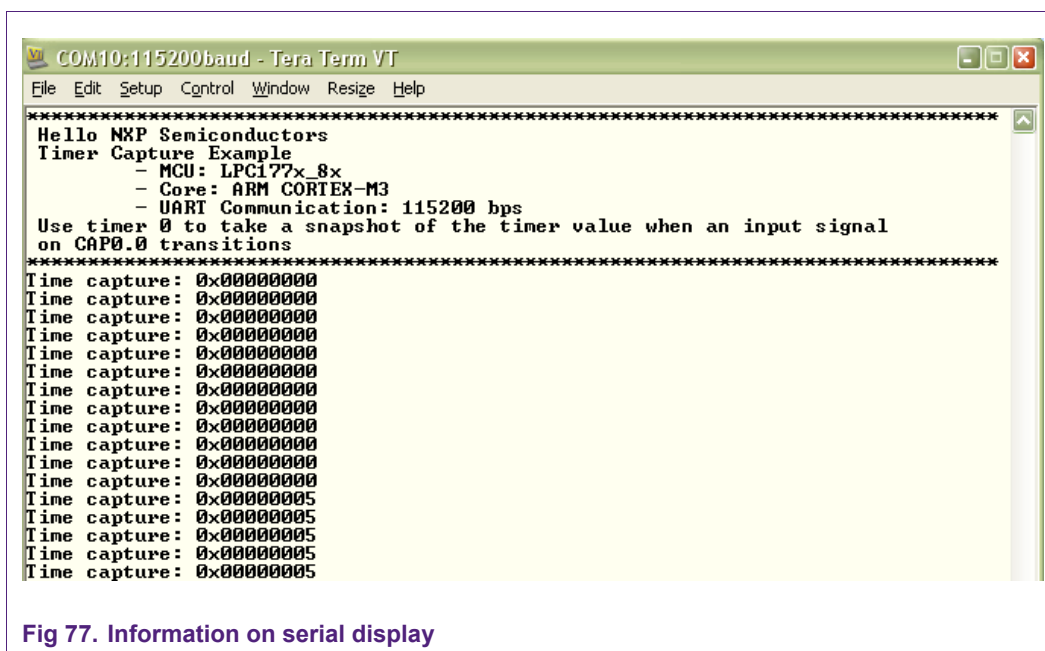


Fig 77. Information on serial display

3.22.3 Timer_FreqMeasure

3.22.3.1 Example description

Purpose

This example describes how to use Timer to measure a signal's frequency.

Knowledge and Process

To do frequency measurement, it's required a signal whose frequency needs to be determined. This sample signal will input to a capture (CAP) channel of a timer (called measuring timer) whose functionality is to measure the frequency.

In this example, in order to create the sample signal, it is used another to timer. This signal timer allows that users input a value from a serial terminal on PC and this value is considered as a pre-defined frequency of the signal that is being created. Then the created sample signal (outputs from a MAT channel of the timer) is come to measuring timer to exam the cycle.

Settings for sample timer:

- Pre-scale register = 1 us.
- Match register = $1 / ((\text{frequency}) * 1\text{us} * 2) = 500000 / \text{frequency}$
- No interrupt no stop but reset timer counter on match.
- Configure TIMER0 as follow:
- Pre-scale register = 1 us.
- Capture register: channel 0, capture on rising edge, and do interrupt on capturing.

Settings for measuring timer:

- Pre-scale register = 1 us.
- Prepare 5 (this value is defined by NO_MEASURING_SAMPLE) interrupt times, in which timer counter and pre-scale are reset, for stable.

- After 5 interrupt times, it will get the capture value, raise “Done” flag.
Wait for “Done” flag, then calculate and print out the measure frequency by:
one cycle = capture value * 1us (as measuring timer’s configuration).
➔ signal frequency = 1/one cycle = 1,000,000 / capture value.

3.22.3.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

freqmeasure.c: Main program

3.22.3.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It’s required a link from the MAT channel of the sample timer to CAP channel of the measuring timer.

Since, it must follow the configurations below:

➤ LPC1788 OEM board rev A connects with OEM base board rev A

- Sample timer using for this board is TIMER2
 - Measuring timer using for this board is TIMER0
- ➔ Link CAP0.0 with MAT2.0: Wire *P1.26 (@ J5.35)* to *P0.6 (@ J3.18)*

➤ LPC1788 IAR Start Kit Rev.B

- Sample timer using for this board is TIMER3
 - Measuring timer using for this board is TIMER2
- ➔ Link CAP2.0 with MAT3.0: Wire *P0.4 (@ EXT-4)* to *P0.10 (@ UXT-6)*

3.22.3.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.22.3.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:

- Input any value to serial terminal on computer as an expecting frequency of the sample signal: (this number should have 3 digits).
- This example will measure input signal and print into screen of the PC terminal.
- If going to test more, please type 'c' to on PC terminal.

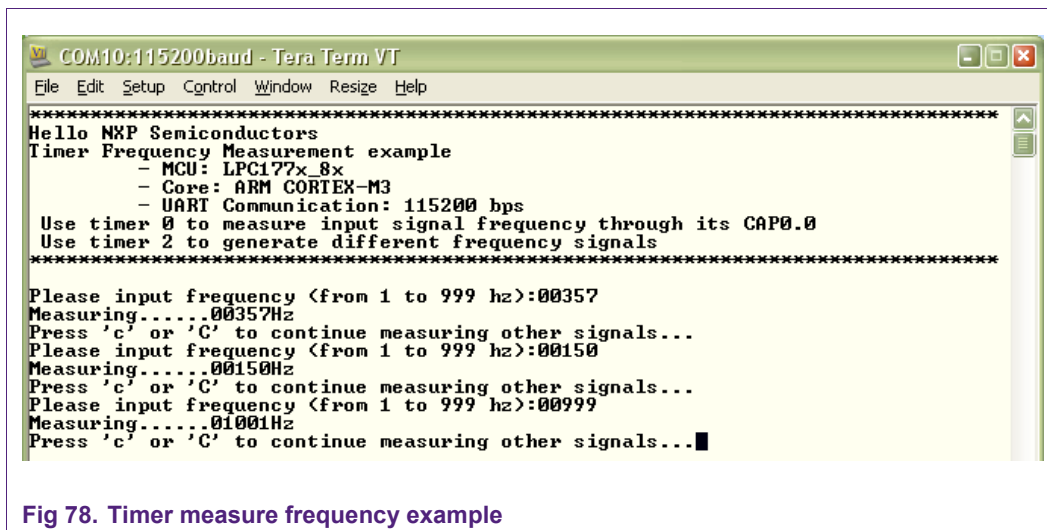


Fig 78. Timer measure frequency example

Notes: We can use oscilloscope to observe sample signal frequency.

3.23 UART (UNIVERSAL ASYNCHRONOUS RECEIVER/TRANSMITTER)

3.23.1 Uart_Polling

3.23.1.1 Example description

Purpose

This example describes how to use UART in polling mode

Process

UART is configured as the following:

- Baudrate = 9600bps
- 8 data bit
- 1 Stop bit
- None parity
- No flow control
- Receive and transmit enable

The data received here is obtained by keep on polling the character of receiving buffer register of UART component. This data should process immediately after retrieving.

UART will print welcome screen first, then:

- Press any key to have it read in from the terminal and returned back to the terminal.
- Press ESC to exit.
- Press 'r' to print welcome screen menu again.

3.23.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Uart_Polling.c: Main program

3.23.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

➤ LPC1788 OEM board rev A connects with OEM base board rev A

This example is able to test on UART0/1/2

UART configuration:

- ✓ **UART0:** USB serial port
 - + All jumpers: Default
- ✓ **UART1:**
 - + P0.15 (@ J5.19) - JP12.2
 - + P0.16 (@ J3.24) - JP13.2
- ✓ **UART2:**
 - + JP6: 1-2: OFF
 - 3-4: OFF
 - 5-6: ON
 - + JP12: 1-2
 - + JP13: 1-2
 - + Other jumpers: Default

➤ LPC1788 IAR Start Kit Rev.B

Because only UART0 has enough pin-out that is ready for using. It is able to run with UART0 only.

- ✓ Jumpers: Optional

3.23.1.4 Running mode

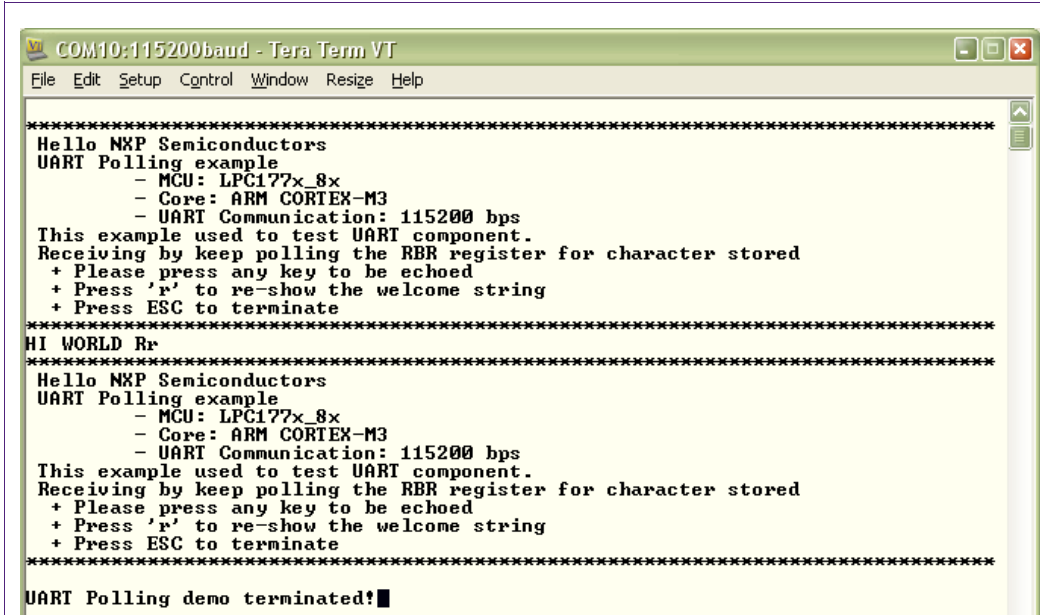
This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.23.1.5 Step to run

- (1) Choose the UART component to work with by `#define UART_TEST_NUM`
- (2) Build example
- (3) Burn built image file (in hex) into the board (in case of ROM mode running)

- (4) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (5) Do hardware configuration exactly following the above instructions.
- (6) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (7) Testing actions and results:
 - See the content on serial terminal, it should be like this:



The screenshot shows a Tera Term VT window titled 'COM10:115200baud - Tera Term VT'. The window contains the following text:

```
*****
Hello NXP Semiconductors
UART Polling example
- MCU: LPC177x_8x
- Core: ARM CORTEX-M3
- UART Communication: 115200 bps
This example used to test UART component.
Receiving by keep polling the RBR register for character stored
+ Please press any key to be echoed
+ Press 'r' to re-show the welcome string
+ Press ESC to terminate
*****
HI WORLD Rr
*****
Hello NXP Semiconductors
UART Polling example
- MCU: LPC177x_8x
- Core: ARM CORTEX-M3
- UART Communication: 115200 bps
This example used to test UART component.
Receiving by keep polling the RBR register for character stored
+ Please press any key to be echoed
+ Press 'r' to re-show the welcome string
+ Press ESC to terminate
*****
UART Polling demo terminated!■
```

Fig 79. Information on serial display

3.23.2 Uart_Interrupt

3.23.2.1 Example description

Purpose

This example describes how to use UART in interrupt mode

Process

UART configuration:

- 9600bps
- 8 data bit
- No parity
- 1 stop bit
- No flow control
- Receive and transmit enable

The data receiving is done by the interrupt handler. Everytime a new data is come, an interrupt will be generated. By this way, the example will automatically get the data on line and store it into a global buffer for later using.

UART will print welcome screen first, then:

- Press any key to have it read in from the terminal and returned back to the terminal.
- Press ESC to exit.
- Press 'r' to print welcome screen menu again.

3.23.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Uart_Interrupt.c: Main program

3.23.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

➤ LPC1788 OEM board rev A connects with OEM base board rev A

This example is able to test on UART0/1/2

UART configuration:

- ✓ **UART0:** USB serial port
 - + All jumpers: Default
- ✓ **UART1:**
 - + P0.15 (@ J5.19) - JP12.2
 - + P0.16 (@ J3.24) - JP13.2
- ✓ **UART2:**
 - + JP6: 1-2: OFF
 - 3-4: OFF
 - 5-6: ON
 - + JP12: 1-2
 - + JP13: 1-2
 - + Other jumpers: Default

➤ LPC1788 IAR Start Kit Rev.B

Because only UART0 has enough pin-out that is ready for using. It's able to run with UART0 only.

- ✓ Jumpers: optional

3.23.2.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.23.2.5 Step to run

- (1) Choose the UART component to work with by `#define UART_TEST_NUM`
- (2) Build example
- (3) Burn built image file (in hex) into the board (in case of ROM mode running)
- (4) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (5) Do hardware configuration exactly following the above instructions.
- (6) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (7) Testing actions and results:
 - See the content on serial terminal, it should be like this:

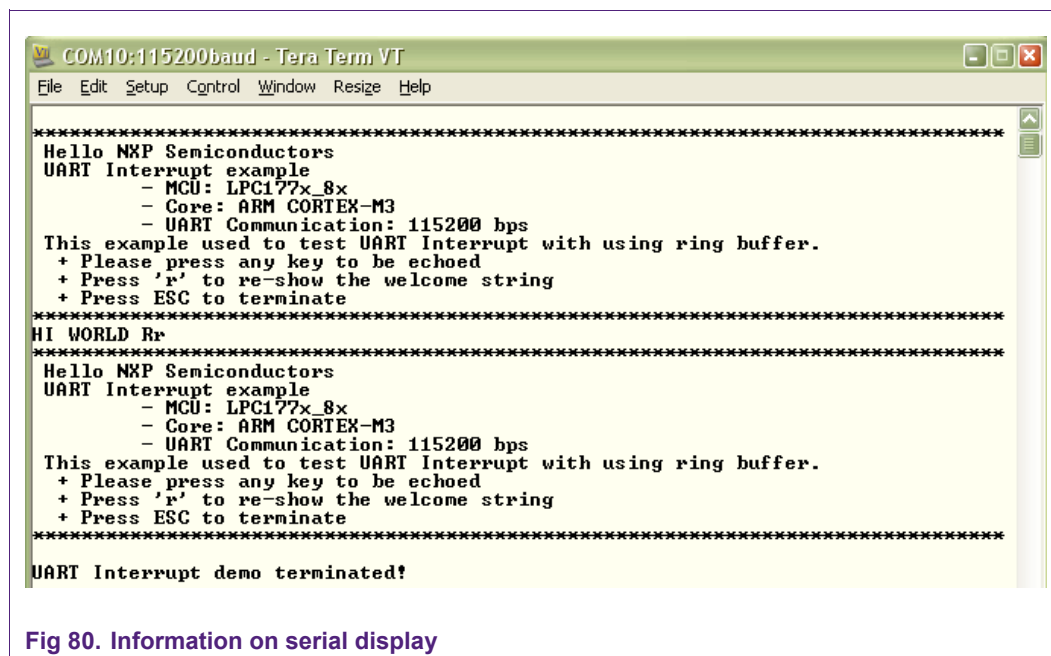


Fig 80. Information on serial display

3.23.3 Uart_Dma

3.23.3.1 Example description

Purpose

This example describes how to use UART in DMA mode

Knowledge and Process

UART is configured as the following:

- Baudrate = 9600bps
- 8 data bit
- 1 Stop bit
- None parity

GPDMA channel 0 using to transmit the welcome message (the destination source is UART0 transmit pin)

GPDMA channel 1 using to receive the character (the destination source is the UART0 receive pin)

3.23.3.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files
Uart_Dma.c: Main program

3.23.3.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

➤ LPC1788 OEM board rev A connects with OEM base board rev A

This example is able to test on UART0/1/2

UART configuration:

- ✓ **UART0:** USB serial port
 - + All jumpers: Default
- ✓ **UART1:**
 - + P0.15 (@ J5.19) - JP12.2
 - + P0.16 (@ J3.24) - JP13.2
- ✓ **UART2:**
 - + JP6: 1-2: OFF
3-4: OFF
5-6: ON
 - + JP12: 1-2
 - + JP13: 1-2
 - + Other jumpers: Default

➤ LPC1788 IAR Start Kit Rev.B

Because only UART0 has enough pin-out that is ready for using. It's able to run with UART0 only.

- ✓ Jumpers: Optional

3.23.3.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.23.3.5 Step to run

- (1) Choose the UART component to work with by `#define UART_TEST_NUM`
- (2) Build example

- (3) Burn built image file (in hex) into the board (in case of ROM mode running)
- (4) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (5) Do hardware configuration exactly following the above instructions.
- (6) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (7) Testing actions and results:
 - See the content on serial terminal, it should be like this:

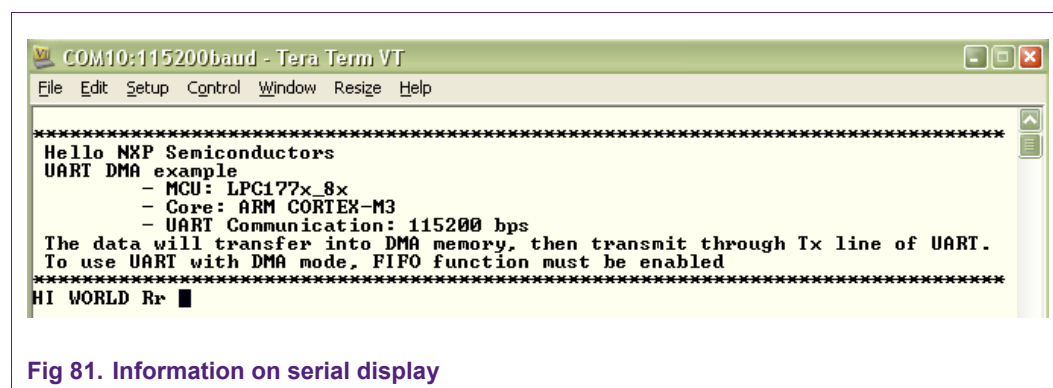


Fig 81. Information on serial display

3.23.4 Uart_Autobaud

3.23.4.1 Example description

Purpose

This is a simple UART example using auto baudrate mode

Process

UART0 is configured as the default settings:

- Baudrate set to auto mode
- 8 data bit
- 1 Stop bit
- None parity

After reset, firstly, type 'A' or 'a' character to start Auto baud rate mode.

Once Auto baud rate mode completed, print welcome screen, then press any key to have it read in from the terminal and returned back to the terminal.

3.23.4.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Uart_Autobaud.c: Main program

3.23.4.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

➤ **LPC1788 OEM board rev A connects with OEM base board rev A**

This example is able to test on UART0/1/2

UART configuration:

- ✓ **UART0:** USB serial port
 - + All jumpers: Default
- ✓ **UART1:**
 - + P0.15 (@ J5.19) - JP12.2
 - + P0.16 (@ J3.24) - JP13.2
- ✓ **UART2:**
 - + JP6: 1-2: OFF
 - 3-4: OFF
 - 5-6: ON
 - + JP12: 1-2
 - + JP13: 1-2
 - + Other jumpers: Default

➤ **LPC1788 IAR Start Kit Rev.B**

Because only UART0 has enough pin-out that is ready for using. It is able to run with UART0 only.

- ✓ Jumpers: Optional

3.23.4.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.23.4.5 Step to run

- (1) Choose the UART component to work with by `#define UART_TEST_NUM`
- (2) Build example
- (3) Burn built image file (in hex) into the board (in case of ROM mode running)
- (4) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (5) Do hardware configuration exactly following the above instructions.
- (6) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (7) Testing actions and results:
 - Select any baudrate for serial terminal on PC (as HyperTerminal, TeraTerm, Flash Magic terminal,...)

- Press 'A' or 'a' character on the PC terminal to send back to the board.
- After receiving the above character, the IC will print the welcome screen on PC terminal.
- If typing any character on PC terminal, this character will be echo
- The content on serial terminal should be like this:

```

COM10:115200baud - Tera Term VT
File Edit Setup Control Window Resize Help

?
AutoBaudrate Status: Synchronous!

*****
Hello NXP Semiconductors
UART Auto-Baudrate example
- MCU: LPC177x_8x
- Core: ARM CORTEX-M3
- UART Communication: 115200 bps
This example used to test UART component with autobaudrate function.
It will adjust its rate to synchronize with the sending data
+ Please press any key to be echoed
+ Press 'r' to re-show the welcome string
+ Press ESC to terminate
*****
aHI WORLD Rr
*****
Hello NXP Semiconductors
UART Auto-Baudrate example
- MCU: LPC177x_8x
- Core: ARM CORTEX-M3
- UART Communication: 115200 bps
This example used to test UART component with autobaudrate function.
It will adjust its rate to synchronize with the sending data
+ Please press any key to be echoed
+ Press 'r' to re-show the welcome string
+ Press ESC to terminate
*****
UART Polling demo terminated!

```

Fig 82. Information on serial display

3.23.5 Uart_FullModem

3.23.5.1 Example description

Purpose

This is a simple UART example using UART Full modem mode.

Process

Configure UART using the following settings:

- Baudrate = 9600bps
- 8 data bit
- 1 Stop bit
- None parity

After reset UART will send the welcome message then start to receive the character from PC and send back that character to PC.

3.23.5.2 Directory contents

- \EWARM: includes EWARM (IAR) project and configuration files
- \Keil: includes RVMDK (Keil) project and configuration files

Uart_FullModem.c: Main program

3.23.5.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

These jumpers must be configured as follows:

➤ LPC1788 OEM board rev A connects with OEM base board rev A

Using UART1 with Full Modem

- ✓ P0.15 (@ **J5.19**) connects to **JP12.2**
- ✓ P0.16 (@ **J3.24**) connects to **JP13.2**
- ✓ P0.17 (@ **J5.20**) connects to **GND**

Using UART2 with Full Modem

- ✓ P0.10 (@ **J3.20**) - pin 2 (at **JP12**)
- ✓ P0.11 (@ **J3.11**) - pin 2 (at **JP13**)
- ✓ P0.17 (@ **J5.20**) - **GND**

➤ LPC1788 IAR Start Kit Rev.B

Unable to run this example on this board because there's no pin-out

No flow control UART Full Modem connection

Because evaluation board does not wire all pins of UART1 to the COM1 port, the signal on CTS, RTS might be in incorrect state for UART1 running. In this case, CTS pin *P3.18* must be pulled-low

3.23.5.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.23.5.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the working board.
- (6) Testing actions and results:
 - See the content on serial terminal, it should be like this:

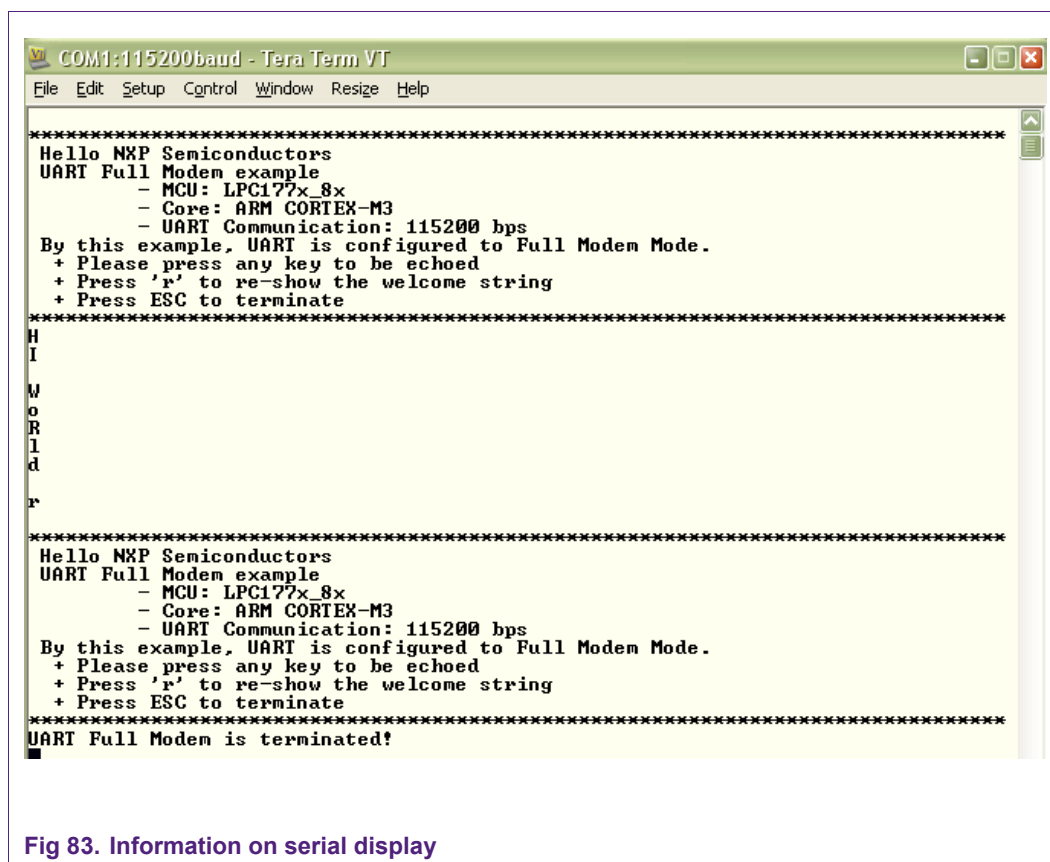


Fig 83. Information on serial display

3.23.6 Uart_IrdaTransmit

3.23.6.1 Example description

Purpose

This example, together with the transmit example, describes how to use UART in IrDA mode

Process

Only UART4 in the LPC177x_8x IC has the function for IrDA transfers. This application will illustrate this matter.

To test IrDA function of UART, we need 2 boards (2 devices), in which one for transmitting and one for receiving data from Infra Link.

Responsibility of this example is transmitting data (1 byte per time). These transmitting data is got from PC serial terminal where users input a byte (in Heximal base) to. The serial communication here to PC is done by another UART port number. This data from PC will be shown by LEDs at Infra Receiver site.

UART4 configuration to use with Infra mode:

- 9600bps
- 8 data bit
- No parity
- 1 stop bit
- No flow control

- Enable IrDA mode
- Transmit enable

UART4 will wait for 1-byte data from PC (obtained from another UART) then send it to Receiver via Infra Interface and keep on waiting data and send...

Notes: the data that will go through IrDA communication is formed by a number with 2 heximal digits (as 1 byte)

3.23.6.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files .

Uart_IrdaTransmit.c: Main program

3.23.6.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

UART IrDA connection:

Add-in hardware: 1 Infrared Transmit LED, 1 transistor C1815, 2 resistors 33R and 150R. They are wired with each other following below schema

- Base of the transistor C1815 - Resistor - P0.22 (who will transmit the UART data)
- Collector of the C1815 - Resistor - Cathode of Infra LED
- Emitter of C1815 - GND
- Anode of LED - 3.3V Pole

3.23.6.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.23.6.5 Step to run

- (1) Build this example and also build the IrDA Receiving example.
- (2) Burn both 2 built image files (in hex) into 2 boards (in case of ROM mode running): one for transmitting and another for receiving.
- (3) Connect expected UART from this testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset both 2 boards with loading 2 images about IrDA mentioned above (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - After reset, this board will ask 1-byte data from PC serial terminal.

- Data entered from PC terminal is received by this example, then parsing, process and lastly sending to receiver through IrDA interface.
- After transmitting IrDA data, we should check the indicators on the IrDA receiver.

The content on PC terminal after an IrDA transmission likes below:

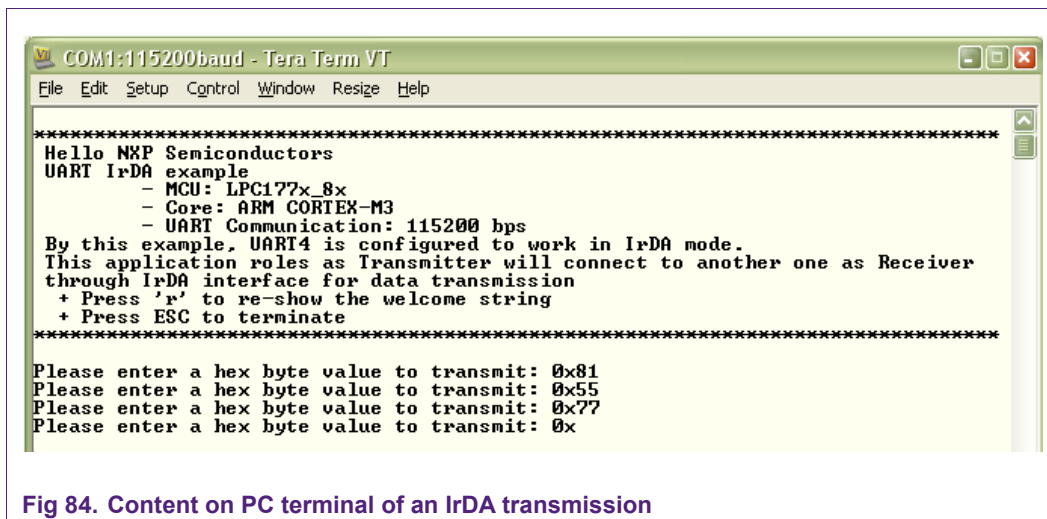


Fig 84. Content on PC terminal of an IrDA transmission

3.23.7 Uart_IrdaReceive

3.23.7.1 Example description

Purpose

This example, together with receive example, describes how to use UART in IrDA mode

Knowledge and Process

Only UART4 in the LPC177x_8x IC has the function for IrDA (Infrared Data Association) transfers. This application will illustrate this matter.

To test IrDA function of UART, we need 2 boards (2 devices), in which one for transmitting and one for receiving data from Infra

Responsibility of this example is receiving and processing what obtain through Infra Association.

UART4 configuration to use with Infra mode:

- 9600bps
- 8 data bit
- No parity
- 1 stop bit
- No flow control
- Enable IrDA mode
- Transmit enable

The data received through IrDA protocol will be displayed by the LEDs that controlled by PCA9532 (interacting with LPC177x_8x via I2C interface)

UART will print welcome screen first, then UART4 keep reading the income IrDA signal and output to 8 LEDs bank the received value.

3.23.7.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Uart_IrdaReceive.c: Main program

LPC178x_Uart_IrDA_Schema.JPG: schematic for UART IrDA

3.23.7.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

UART IrDA connection:

Add-in hardware: 1 Infrared receive LED, 1 resistor 150R

- Anode of Infra Receiver LED – 3.3V Pole
- Cathode of the Infra LED – Resistor 150R - GND
- Cathode of the Infra LED – P2.9

3.23.7.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode).

3.23.7.5 Step to run

- (1) Build this example and also build the IrDA Transmitting example.
- (2) Burn both 2 built image files (in hex) into 2 boards (in case of ROM mode running): one for receiving and another for transmitting.
- (3) Connect expected UART from this testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset both 2 boards with loading 2 images about IrDA mentioned above (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - Once 1-byte data on IrDA is received, this data will be parsed into executable form send to PCA9532 by I2C interface to control the LEDs displaying.
 - See and compare the LEDs displaying by the value that have been sent from IrDA Transmitter Example.
 - Bit 0 of this 1-byte data will turn OFF the LED at corresponding position in a set of 8 LEDs
 - Bit 1 of this 1-byte data will turn ON the LED at corresponding position in a set of 8 LEDs

3.23.8 Uart_Rs485Master

3.23.8.1 Example description

Purpose

This example describes how to use RS485 function via UART interface of LPC1788 in master mode.

Process

RS485 function on UART1 acts as Master mode on RS485 bus.

Master device will send a specified slave device address value first, then master device will send data frames. After sending completed, master device wait for response from slave device (example RS485_slave).

3.23.8.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files
\Keil: includes RVMDK (Keil) project and configuration files

Uart_Rs485Master.c: Main program

3.23.8.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

RS485 connection

Please see the 'Transceiver_Master.png' in this directory for wiring information.

3.23.8.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.23.8.5 Step to run

- (1) Build this example and also build the RS485 Slave example.
- (2) Burn both 2 built image files (in hex) into 2 boards (in case of ROM mode running): one for RS485 Master and another for RS485 Slave.
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration (especially RS485 connection) exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - See the content on serial terminal, it should be like this:

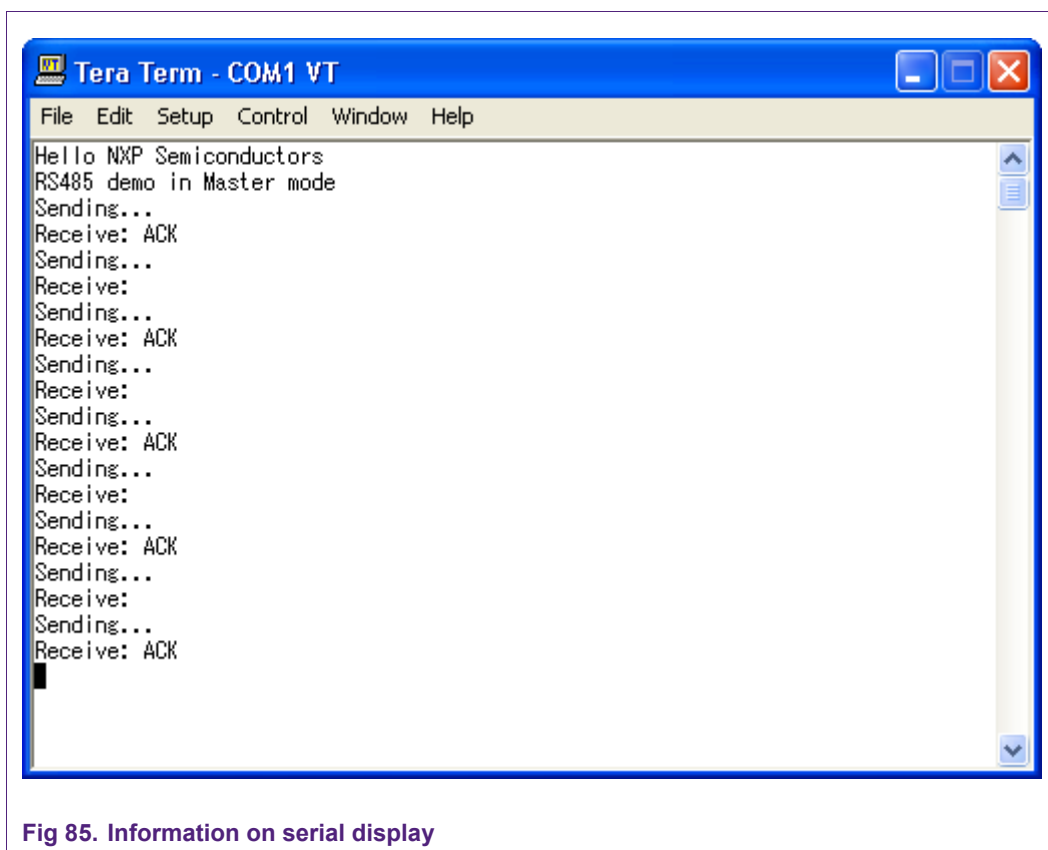


Fig 85. Information on serial display

Notes: This example should start running later than the RS485 Slave example loading

3.23.9 Uart_Rs485Slave

3.23.9.1 Example description

Purpose

This example describes how to use RS485 functionality on UART of LPC1788 in slave mode.

Process

RS485 function in this application acts in Slave mode on RS485 bus.

RS485 Slave device in this example can operate in separate mode as following:

- Slave device always receives all frames on RS485 bus, regardless data frame (9-bit mode with parity stick '0') or slave address frame (9-bit mode with parity stick '1').
- Slave device does not always receive all frames on RS485 bus. In this case, only Slave address frame can trigger an interrupt event, then Slave device can accept the following data frame by determine that Slave address frame is its own address or not (implemented by software).
- Slave device is in auto Slave address detection mode. In this mode, only Slave address frame with Slave address value that matched with pre-configured slave address will be accepted automatically (by hardware) and trigger an interrupt callback event to handle following data frames.

3.23.9.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Uart_Rs485Slave.c: Main program

3.23.9.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

RS485 connection

Please see the 'Transceiver_Master.png' in this directory for wiring information.

3.23.9.4 Running mode

This example can run on both RAM/ROM (FLASH) mode.

3.23.9.5 Step to run

- (1) Build this example and also build the RS485 Master example.
- (2) Burn both 2 built image files (in hex) into 2 boards (in case of ROM mode running): one for RS485 Slave and another for RS485 Master.
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration (especially RS485 connection) exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - See the content on serial terminal, it should be like this:

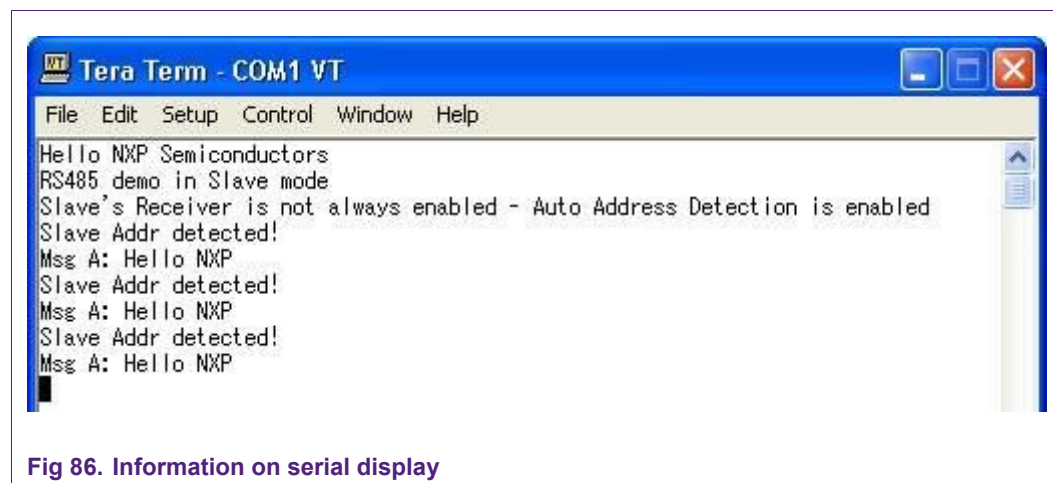


Fig 86. Information on serial display

Notes: This example should start running before the RS485 Master example.

3.24 USB DEVICES

3.24.1 Usb_MassStorage

3.24.1.1 Example description

Purpose

This example describes how to write a simple USB Mass Storage application on LPC1788.

Process

The MassStorage project is a Mass Storage simple demo run on LPC177x_8x

It demonstrates an USB Memory based on USB Mass Storage Class.

The USB Memory is automatically recognized by the host PC running Windows which will load a generic Mass Storage driver.

3.24.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

DiskImg.c: Disk Image (LPC178x) data

memory.h/.c: USB Memory Storage

msc.h: USB Mass Storage Class Definition

mscuser.h/.c: Mass Storage Class Custom User

usb.h: USB Definitions

usbcfg.h: USB Configuration Definition

usbcore.h/.c: USB Core Module

usbdesc.h/.c: USB Descriptors

usbhw.h/.c: USB Hardware Layer

usbreg.h: USB Hardware Layer Definitions for NXP Semiconductors LPC

usbuser.h/.c: USB Custom User Module

memory.h/.c: USB Memory Storage Demo (main program)

3.24.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

➤ LPC1788 OEM Board rev A and OEM Base Board rev A

✓ JP15: ON (near J19)

✓ JP16: ON (near J19)

➤ LPC1788 IAR Start Kit Rev.B

✓ Jumpers: Optional

3.24.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.24.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - After reset, see UGL (USB Good Link), then open “*My Computer*” on PC to check “LPC177x_8x” disk drive.
 - Open the disk, “*Readme.txt*” file will shown with read-only attribute.

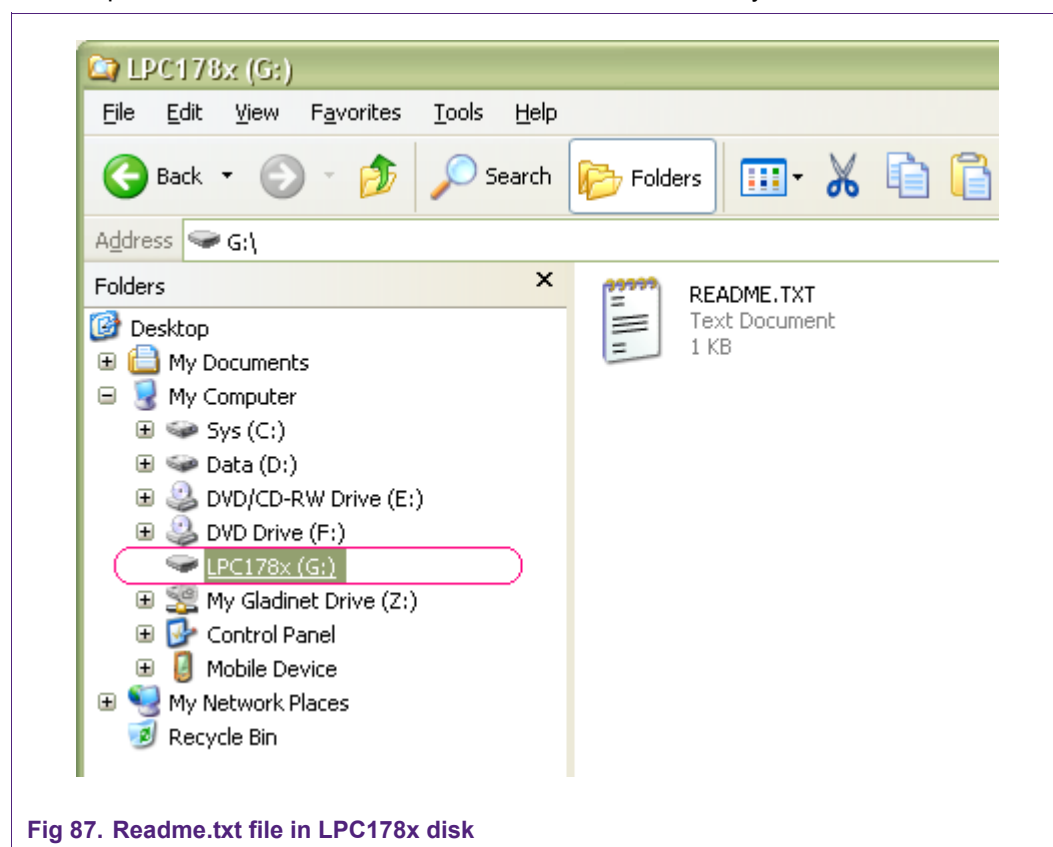


Fig 87. Readme.txt file in LPC178x disk

- Try handling data inside (as create, copy, delete...) → They are all ok.

3.24.2 Usb_VirtualCom

3.24.2.1 Example description

Purpose

This example describes how to configure USB Device of as a virtual COM port.

Process

The PC will install a virtual COM port on the PC (see Driver Installation).

After installation an additional port "*LPC178x USB VCom Port (COMx)*" can be found under *System/Hardware/Device Manager/Ports (COM & LPT)*.

Number "x" is not fixed as different PC configuration may have different "x" displayed on the device manager. The USB host driver assigns "x" dynamically based on the existing COM port configuration of the system.

To test the virtual COM Port that just linked, it should:

- Use one more standard UART connection between the board and the PC
- Checking if data transmitting from PC through standard COM to the board then back to PC through virtual COM is correct or not
- And oppositely checking if data transmitting from PC through virtual COM to the board then back to PC through standard COM is correct or not

The data can be monitored by 2 separate serial terminal programs.

Notes:

- To use standard UART Port on the board, it should consider `#define PORT_NUM` in `serial.h` for the UART number will be use directly with PC.
- RST jumper needs to be removed to start the Virtual COM port test.

3.24.2.2 Driver installation

"Welcome to the Found New Hardware Wizard" appears

- select *"No, not this time"*
- press 'Next'
- select *"Install from a list or specific location (Advanced)"*
- press 'Next'
- select *"Search for the best driver in these locations"*
- check *"include this location in the search"*
- set to <project folder>
- press 'Next'

"Hardware Installation" appears

:Has not passed Windows Logo testing..."

- press *"Continue Anyway"*

“Completing the Found New Hardware Wizard” appears

- press “Finish”

3.24.2.3 Directory contents

\EWARM:	includes EWARM (IAR) project and configuration files
\Keil:	includes RVMDK (Keil) project and configuration files
cdc.h:	USB CDC (Communication Device) Definitions
cdcuser.h/.c:	USB Communication Device Class User module
serial.h/.c:	serial port handling for LPC17xx
usb.h:	USB Definitions
usbcfg.h:	USB Custom Configuration
usbcore.h/.c:	USB Core Module
usbdesc.h/.c:	USB Descriptors
usbhw.h/.c:	SB Hardware Layer Module
usbreg.h:	USB Hardware Layer Definitions for NXP Semiconductors LPC
usbuser.h/.c:	USB Custom User Module
vcomdemo.h/.c:	main program
lpc17xx-vom.inf:	driver information of VCOM LPC17xx (required when Windows requires install driver)

3.24.2.4 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

- **LPC1788 OEM Board rev A and OEM Base Board rev A**
 - ✓ JP15: ON (near J19)
 - ✓ JP16: ON (near J19)
- **LPC1788 IAR Start Kit Rev.B**
 - ✓ Jumpers: Optional

3.24.2.5 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.24.2.6 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)

- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - If UGL (USB Good Link) LED on the board is turned ON, open “**Device Manager > Ports (COM & LPT)**” to check for the appearance of “**LPC178x USB VCOM Port (COMx)**” device.

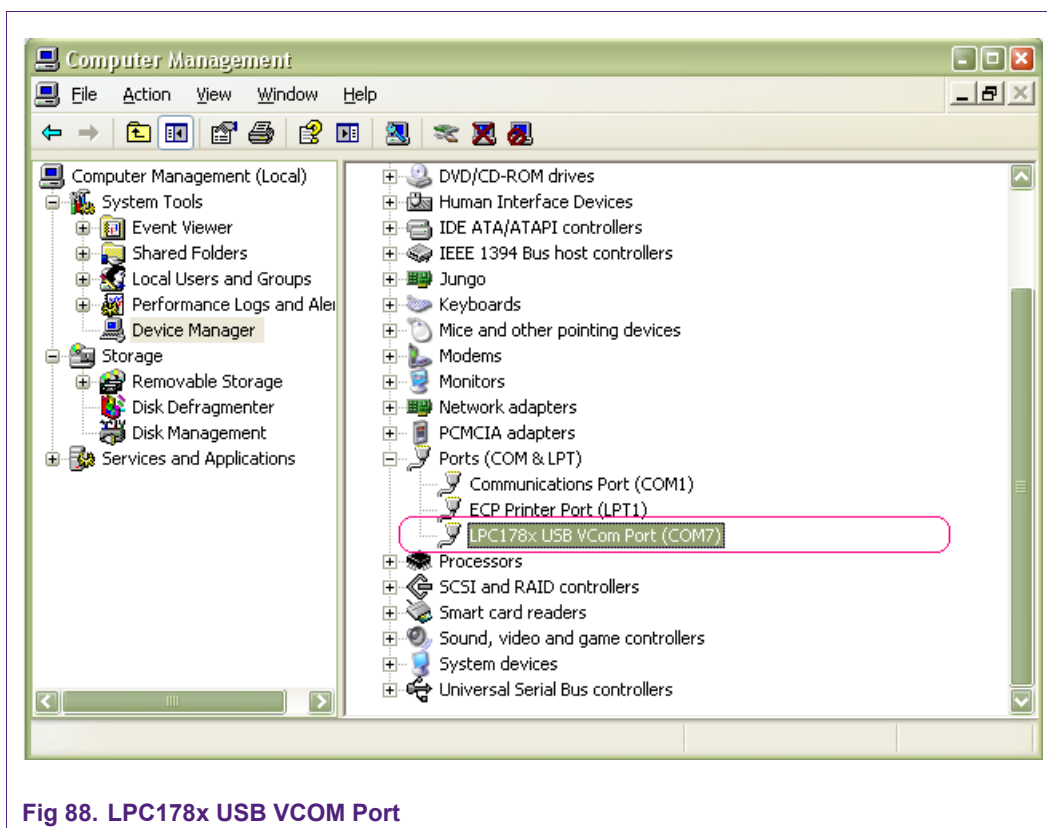


Fig 88. LPC178x USB VCOM Port

- Open 2 Serial Terminal on PC to test the virtual usb com port: one's settings for standard COM (COM1) and another's settings for virtual COM above (COMx) which is COM7 in this case. Assumption: 2 HyperTerminal programs of Windows is opened for this purpose. Their configurations as below:
 - 9600 bps
 - 8 data bits
 - none parity
 - 1 stop bit
 - None flow control

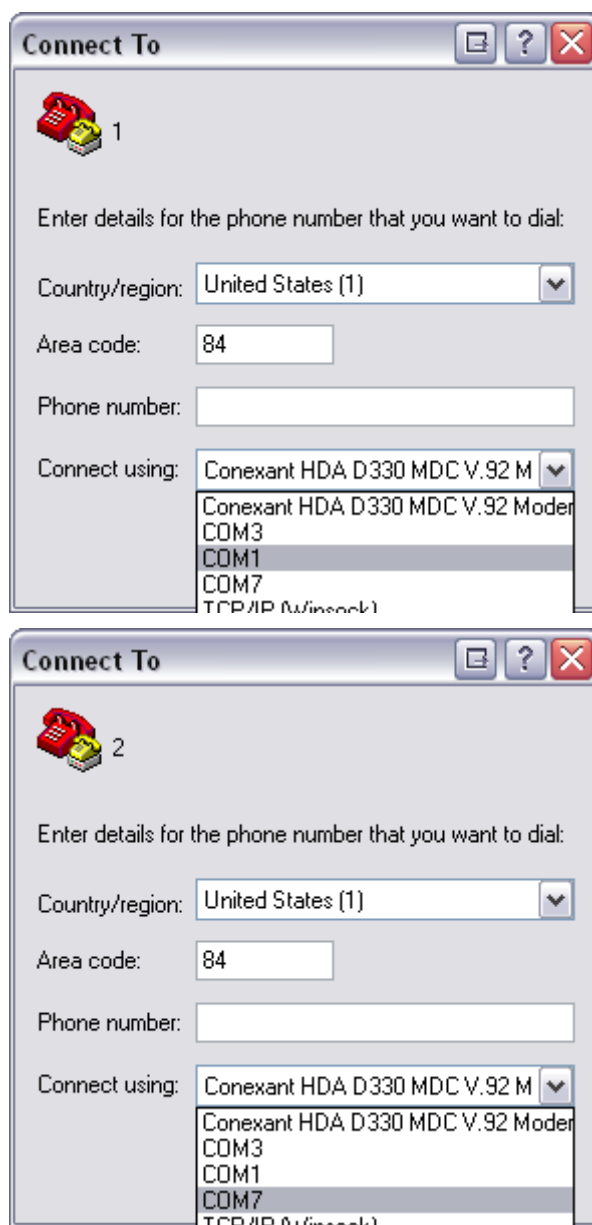


Fig 89. HyperTerminal for standard COM1 and Virtual COM7

- Type any character on one HyperTerminal screen and see this character will be echoed in other screen and vice versa.

Notes: Ensure that COM1 connection between the board and the PC is done already.

3.25 WDT (WATCHDOG TIMER)

3.25.1 Wdt_Interrupt

3.25.1.1 Example description

Purpose

This example describes how to use WDT to generate interrupt after a specific time.

Process

In this example, WDT disables reset function, WDT only generates interrupt when the Watchdog times out.

Before WDT interrupt, current value of Watchdog timer will be written continuously into serial display (these values will be decreased from 5000000 to 0).

Knowledge, the interrupt is occurred at the time when the WDT timer is timeout or when the WDT counter decreasing reaches the Warning Value stored in WDWARNINT register.

We can select the modes by input a serial terminal.

The WDT in this example has set the timeout value after 10s, and the value for warning is 0.5s

Note that: Data displaying via UART will cause delay.

3.25.1.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Wdt_Interrupt.c: Main program

3.25.1.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\BoardSupport\bsp.h` file for current working board.

It does not require any jumper configuration or link.

3.25.1.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.25.1.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.

- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
- The application will allow to choose which test to run:
 - o If input "1" to test the Timeout Interrupt

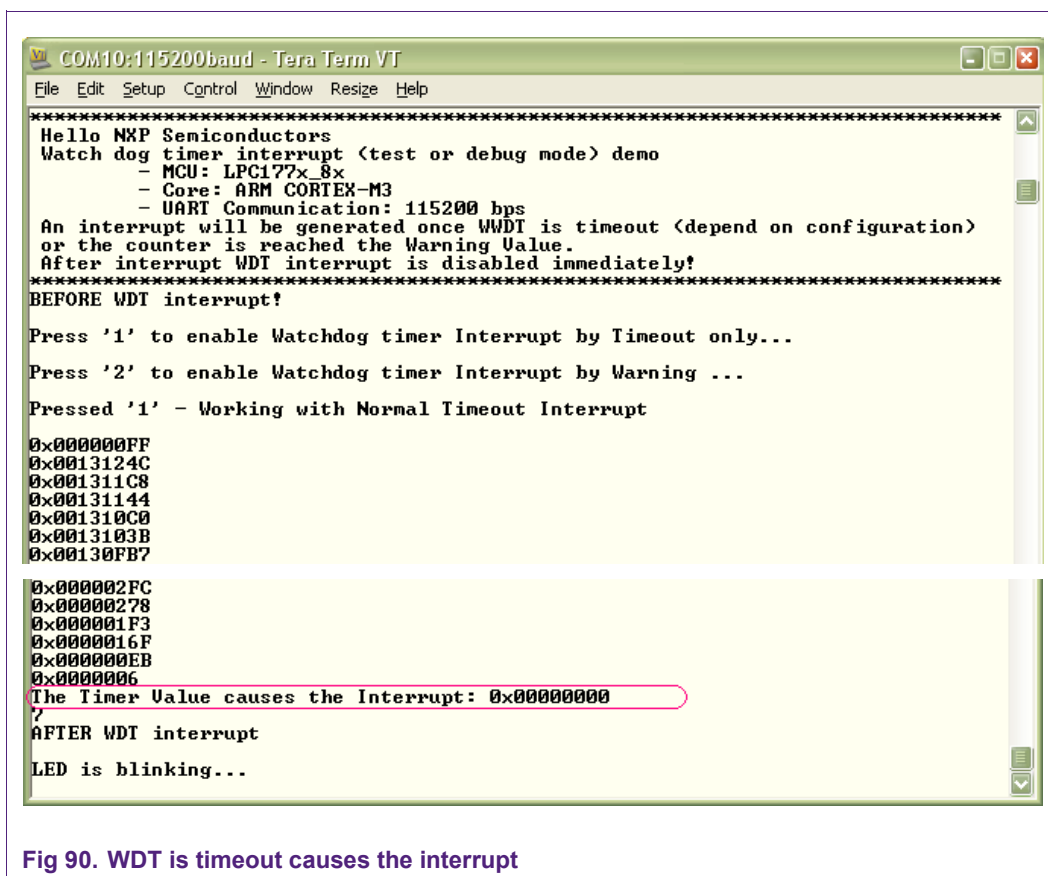


Fig 90. WDT is timeout causes the interrupt

- o If input "2" to test the Warning Interrupt

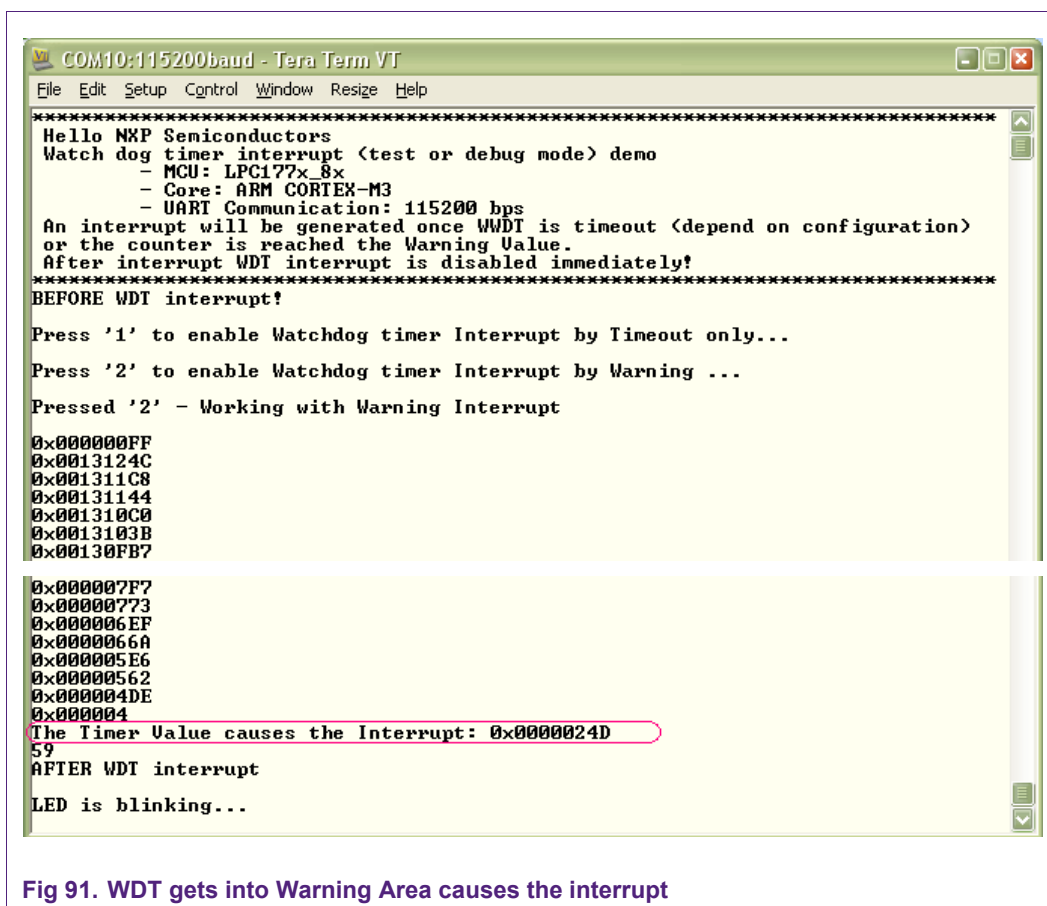


Fig 91. WDT gets into Warning Area causes the interrupt

3.25.2 Wdt_Reset

3.25.2.1 Example description

Purpose

This example describes how to use WDT to generate chip reset after a specific time.

Process

WDT setting:

- Generate reset chip when WDT times out.
- time-out = 5 seconds
- clock source: IRC (Internal RC oscillator)

After start, WDT counter decrease until underflow (5s) to generate a chip reset.

If between 5s, RESET button is pressed, chip force an external reset.

If not, WDT will reset chip after 5s automatically.

After reset, the program will determine what cause of last reset time (external reset or WDT Timeout reset)

3.25.2.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files.

Wdt_Reset.c: Main program

3.25.2.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not require any jumper configuration or link.

3.25.2.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.25.2.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - If in between 5s, hit RESET button, after reset, we will have the notice: "Last MCU reset caused by External!"
 - Unless, WDT cause chip reset and after reset, we will have the notice: "Last MCU reset caused by WDT TimeOut!"
 - See the content on serial terminal, it should be like this:

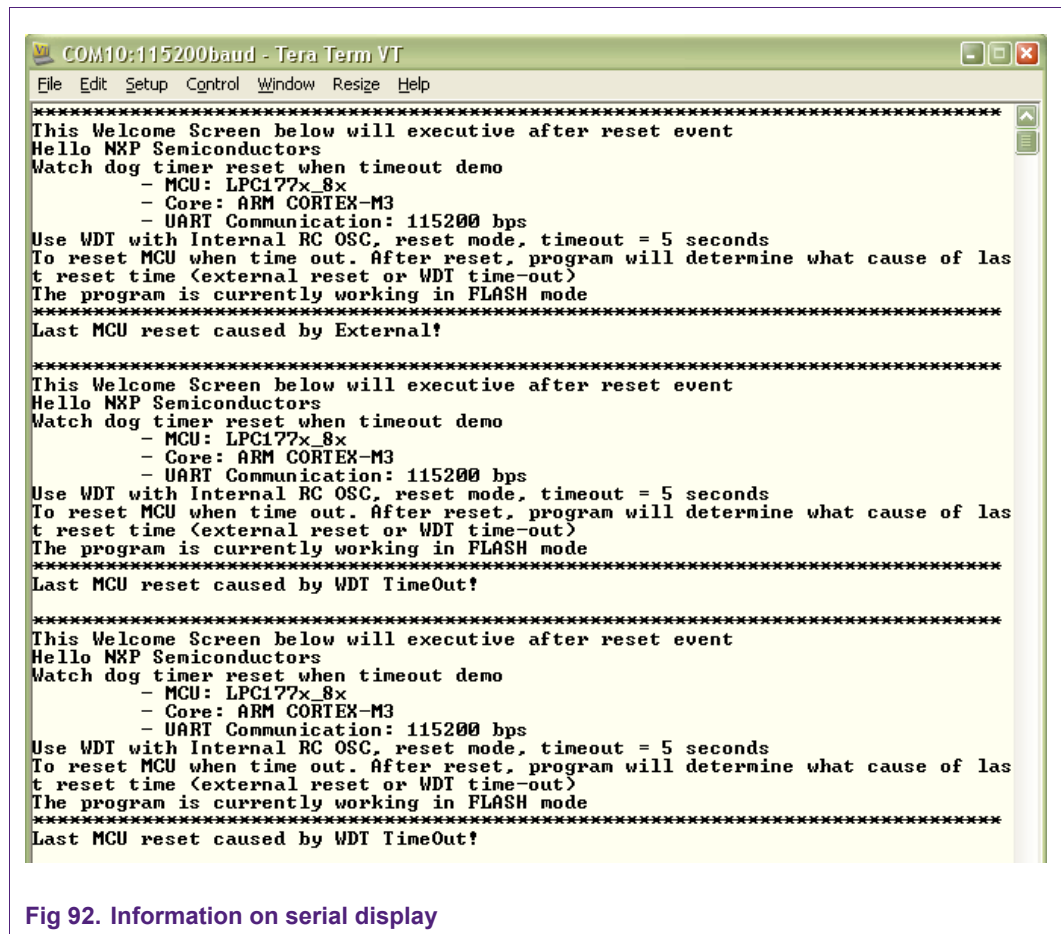


Fig 92. Information on serial display

3.25.3 Wdt_WindowMode

3.25.3.1 Example description

Purpose

To test the operation of Window Mode of Window Watchdog Timer

Process

In this example, WDT disables reset function, WDT just generates interrupt when the Watchdog times out.

WDT setting:

- clock source: IRC (Internal RC oscillator)

By the knowledge, if a feed valid sequence comes at the time when the WatchDog Timer Value (WDTV) is decreased but still high the value written in the WatchDog Timer Window WDWINDOW register, a WatchDog event will be occurred as interrupt, or Reset (by configuration in WatchDog Mode Register WDMOD). The example will configure to reset the IC if the above case happens

Note: data displaying via UART will cause delay.

3.25.3.2 Directory contents

\EWARM: includes EWARM (IAR) project and configuration files

\Keil: includes RVMDK (Keil) project and configuration files

Wdt_WindowMode.c: Main program

3.25.3.3 Hardware configuration

Make sure that the board is powered correctly. For more details, please reference at the [section 2.5.1](#)

Notes: Initialize macro `#define CURR_USING_BRD` in `.\.\.\BoardSupport\bsp.h` file for current working board.

It does not required any jumper configuration or link.

3.25.3.4 Running mode

This example can run only on RAM/ROM (FLASH) mode.

Note: In RAM mode, after the IC reset, the code will retrieve back the hex image in FLASH that has burned last time. It's unable to continue running the current image with debugging

Please reference [section 2.6](#) if going to burn the image hex file by Flash Magic (in case of running in Flash mode)

3.25.3.5 Step to run

- (1) Build example
- (2) Burn built image file (in hex) into the board (in case of ROM mode running)
- (3) Connect expected UART from the testing board to COM Port on PC for displaying messages on a serial terminal (look at [section 2.4](#) for more details)
- (4) Do hardware configuration exactly following the above instructions.
- (5) Reset the board (in case of ROM mode running) and Run the example on the test board.
- (6) Testing actions and results:
 - See the content on serial terminal, it should be like this:

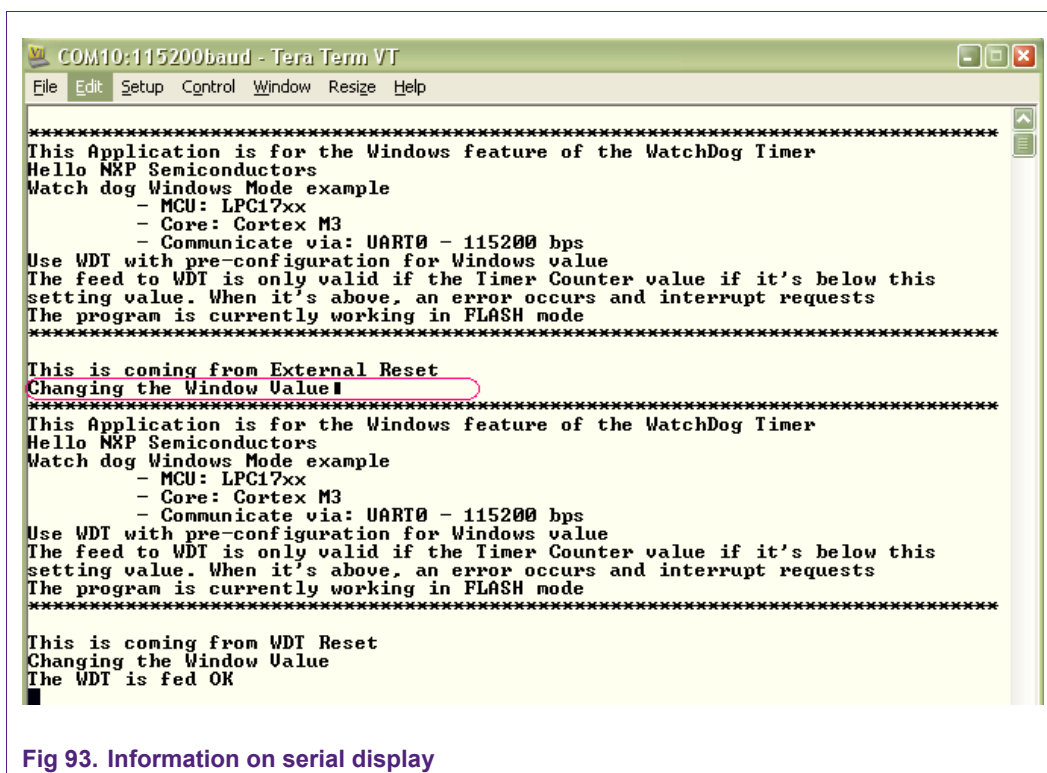


Fig 93. Information on serial display

4. Legal information

4.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

4.2 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

4.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

5. Contents

1.	Introduction	3	3.2.1	Can_Selftest	20
2.	General information	3	3.2.1.1	Example description	20
2.1	Microcontrollers	3	3.2.1.2	Directory contents	20
2.2	Evaluation boards	3	3.2.1.3	Hardware configuration	20
2.2.1	LPC1788 OEM board rev A connects with OEM base board rev A	3	3.2.1.4	Running mode	21
2.2.2	LPC1788 IAR Start Kit Rev.B	4	3.2.1.5	Step to run	21
2.2.3	Reference to the board that using to test by the code	5	3.2.2	Can_Bypass	21
2.3	Toolchains	5	3.2.2.1	Example description	21
2.4	Serial messages display	6	3.2.2.2	Directory contents	22
2.4.1	UART handling for debug messages in the software:	6	3.2.2.3	Hardware configuration	22
2.4.2	Serial Terminal Program on PC:	7	3.2.2.4	Running mode	22
2.5	Initial working condition	11	3.2.2.5	Step to run	22
2.5.1	Power the board:	11	3.2.3	Can_Aflut	23
2.6	Running mode	11	3.2.3.1	Example description	23
2.7	Flag definitions	12	3.2.3.2	Directory contents	24
3.	Examples	13	3.2.3.3	Hardware configuration	24
3.1	ADC (ANALOG – TO – DIGITAL CONVERTER)	13	3.2.3.4	Running mode	24
3.1.1	Adc_Polling	13	3.2.3.5	Step to run	24
3.1.1.1	Example description	13	3.3	CRC (CYCLIC REDUNDANCY CHECK)	27
3.1.1.2	Directory contents	13	3.3.1	Crc_Dma	27
3.1.1.3	Hardware configuration	13	3.3.1.1	Example description	27
3.1.1.4	Running mode	13	3.3.1.2	Directory contents	27
3.1.1.5	Steps to run	14	3.3.1.3	Hardware configuration	28
3.1.2	Adc_Interrupt	14	3.3.1.4	Running mode	28
3.1.2.1	Example description	14	3.3.1.5	Step to run	28
3.1.2.2	Directory contents	15	3.3.2	Crc_Demo	28
3.1.2.3	Hardware configuration	15	3.3.2.1	Example description	28
3.1.2.4	Running mode	15	3.3.2.2	Directory contents	29
3.1.2.5	Steps to run	15	3.3.2.3	Hardware configuration	29
3.1.3	Adc_Dma	16	3.3.2.4	Running mode	29
3.1.3.1	Example description	16	3.3.2.5	Step to run	29
3.1.3.2	Directory contents	16	3.4	DAC (DIGITAL – TO – ANALOG CONVERTER)	30
3.1.3.3	Hardware configuration	16	3.4.1	Dac_Dma	30
3.1.3.4	Running mode	17	3.4.1.1	Example description	30
3.1.3.5	Steps to run	17	3.4.1.2	Directory contents	31
3.1.4	Adc_Burst	17	3.4.1.3	Hardware configuration	31
3.1.4.1	Example description	17	3.4.1.4	Running mode	31
3.1.4.2	Directory contents	18	3.4.1.5	Step to run	31
3.1.4.3	Hardware configuration	18	3.4.2	Dac_SineWave	32
3.1.4.4	Running mode	19	3.4.2.1	Example description	32
3.1.4.5	Steps to run	19	3.4.2.2	Directory contents	33
3.2	CAN (CONTROLLER AREA NETWORK)	20	3.4.2.3	Hardware configuration	33
			3.4.2.4	Running mode	33
			3.4.2.5	Step to run	33
			3.5	DMA (DIRECT MEMORY ACCESS)	34

continued >>

3.5.1	Dma_Flash2Ram	34	3.8.3.3	Hardware configuration	50
3.5.1.1	Example description	34	3.8.3.4	Running mode	50
3.5.1.2	Directory contents	35	3.8.3.5	Step to run	50
3.5.1.3	Hardware configuration	35	3.9	GPIO (GENERAL PURPOSE INPUT/OUTPUT)	
3.5.1.4	Running mode	35		51
3.5.1.5	Step to run	35	3.9.1	Gpio_Interrupt	51
3.6	EEPROM (ELECTRICALLY ERASABLE		3.9.1.1	Example description	51
	PROGRAMMABLE READ – ONLY MEMORY)	35	3.9.1.2	Directory contents	51
3.6.1	Eeprom_Demo	35	3.9.1.3	Hardware configuration	51
3.6.1.1	Example description	35	3.9.1.4	Running mode	51
3.6.1.2	Directory contents	36	3.9.1.5	Step to run	51
3.6.1.3	Hardware configuration	36	3.9.2	Gpio_LedBlinky	52
3.6.1.4	Running mode	36	3.9.2.1	Example description	52
3.6.1.5	Step to run	36	3.9.2.2	Directory contents	52
3.7	EMAC (ETHERNET MEDIA ACCESS		3.9.2.3	Hardware configuration	52
	CONTROLLER)	37	3.9.2.4	Running mode	52
3.7.1	Emac_EasyWeb	37	3.9.2.5	Step to run	52
3.7.1.1	Example description	37	3.10	I2C (INTER – IC CONTROL BUS)	53
3.7.1.2	Directory contents	37	3.10.1	I2c_Pa9532Drv	53
3.7.1.3	Hardware configuration	38	3.10.1.1	Example description	53
3.7.1.4	Running mode	38	3.10.1.2	Directory contents	53
3.7.1.5	Step to run	38	3.10.1.3	Hardware configuration	53
3.7.2	Emac_Raw	41	3.10.1.4	Running mode	53
3.7.2.1	Example description	41	3.10.1.5	Step to run	54
3.7.2.2	Directory contents	41	3.11	I2S (INTER – IC SOUND BUS)	54
3.7.2.3	Hardware configuration	41	3.11.1	I2s_Interrupt	54
3.7.2.4	Running mode	41	3.11.1.1	Example description	54
3.7.2.5	Step to run	42	3.11.1.2	Directory contents	55
3.7.3	Emac_uIP	43	3.11.1.3	Hardware configuration	55
3.7.3.1	Example description	43	3.11.1.4	Running mode	55
3.7.3.2	Directory contents	44	3.11.1.5	Step to run	55
3.7.3.3	Hardware configuration	44	3.11.2	I2s_Dma	56
3.7.3.4	Running mode	45	3.11.2.1	Example description	56
3.7.3.5	Step to run	45	3.11.2.2	Directory contents	57
3.8	EMC (EXTERNAL MEMORY CONTROLLER)	46	3.11.2.3	Hardware configuration	57
3.8.1	Emc_NandFlashDemo	46	3.11.2.4	Running mode	57
3.8.1.1	Example description	46	3.11.2.5	Step to run	58
3.8.1.2	Directory contents	47	3.11.3	I2s_Mclk	59
3.8.1.3	Hardware configuration	47	3.11.3.1	Example description	59
3.8.1.4	Running mode	47	3.11.3.2	Directory contents	59
3.8.1.5	Step to run	47	3.11.3.3	Hardware configuration	60
3.8.2	Emc_NorFlashDemo	48	3.11.3.4	Running mode	60
3.8.2.1	Example description	48	3.11.3.5	Step to run	60
3.8.2.2	Directory contents	48	3.11.4	I2s_4Wire	61
3.8.2.3	Hardware configuration	48	3.11.4.1	Example description	61
3.8.2.4	Running mode	49	3.11.4.2	Directory contents	62
3.8.2.5	Step to run	49	3.11.4.3	Hardware configuration	62
3.8.3	Emc_SdramDemo	49	3.11.4.4	Running mode	62
3.8.3.1	Example description	49	3.11.4.5	Step to run	62
3.8.3.2	Directory contents	50	3.11.5	I2s_Audio	63

continued >>

3.11.5.1	Example description	63	3.16.2.3	Hardware configuration	81
3.11.5.2	Directory contents	63	3.16.2.4	Running mode	81
3.11.5.3	Hardware configuration	64	3.16.2.5	Step to run	82
3.11.5.4	Running mode	64	3.16.3	Pwm_MatchInterrupt	82
3.11.5.5	Step to run	64	3.16.3.1	Example description	82
3.12	LCD (LIQUID CRYSTAL DISPLAY)	65	3.16.3.2	Directory contents	83
3.12.1	Lcd_GFT035A320240Y	65	3.16.3.3	Hardware configuration	83
3.12.1.1	Example description	65	3.16.3.4	Running mode	84
3.12.1.2	Directory contents	65	3.16.3.5	Step to run	84
3.12.1.3	Hardware configuration	65	3.17	PWR (POWER MANAGEMENT)	85
3.12.1.4	Running mode	65	3.17.1	Pwr_Sleep	85
3.12.1.5	Step to run	65	3.17.1.1	Example description	85
3.13	MCI (MULTIMEDIA CARD INTERFACE)	67	3.17.1.2	Directory contents	86
3.13.1	Mci_CidCard	67	3.17.1.3	Hardware configuration	86
3.13.1.1	Example description	67	3.17.1.4	Running mode	86
3.13.1.2	Directory contents	67	3.17.1.5	Step to run	86
3.13.1.3	Hardware configuration	67	3.17.2	Pwr_DeepSleep	87
3.13.1.4	Running mode	68	3.17.2.1	Example description	87
3.13.1.5	Step to run	68	3.17.2.2	Directory contents	87
3.14	MCPWM (MOTOR CONTROL PWM)	69	3.17.2.3	Hardware configuration	87
3.14.1	MCPWM_Simple	69	3.17.2.4	Running mode	88
3.14.1.1	Example description	69	3.17.2.5	Step to run	88
3.14.1.2	Directory contents	70	3.17.3	Pwr_PowerDown	88
3.14.1.3	Hardware configuration	70	3.17.3.1	Example description	88
3.14.1.4	Running mode	70	3.17.3.2	Directory contents	89
3.14.1.5	Step to run	70	3.17.3.3	Hardware configuration	89
3.15	NVIC (NESTED VECTORED INTERRUPT CONTROLLER)	74	3.17.3.4	Running mode	89
3.15.1	Nvic_Priorities	74	3.17.3.5	Step to run	89
3.15.1.1	Example description	74	3.17.4	Pwr_DeepPowerDown	90
3.15.1.2	Directory contents	75	3.17.4.1	Example description	90
3.15.1.3	Hardware configuration	75	3.17.4.2	Directory contents	91
3.15.1.4	Running mode	75	3.17.4.3	Hardware configuration	91
3.15.1.5	Step to run	75	3.17.4.4	Running mode	91
3.15.2	Nvic_VectorTableRelocation	76	3.17.4.5	Step to run	91
3.15.2.1	Example description	76	3.18	QEI (QUADRATURE ENCODER INTERFACE)	92
3.15.2.2	Directory contents	76	3.18.1	QEI_Velo	92
3.15.2.3	Hardware configuration	77	3.18.1.1	Example description	92
3.15.2.4	Running mode	77	3.18.1.2	Directory contents	93
3.15.2.5	Step to run	77	3.18.1.3	Hardware configuration	93
3.16	PWM (PULSE WIDTH MODULATOR)	77	3.18.1.4	Running mode	93
3.16.1	Single_Edge	77	3.18.1.5	Step to run	93
3.16.1.1	Example description	77	3.19	RTC (REAL TIME CLOCK)	94
3.16.1.2	Directory contents	78	3.19.1	Rtc_Alarm	94
3.16.1.3	Hardware configuration	78	3.19.1.1	Example description	94
3.16.1.4	Running mode	79	3.19.1.2	Directory contents	95
3.16.1.5	Step to run	79	3.19.1.3	Hardware configuration	95
3.16.2	Pwm_DualEdge	80	3.19.1.4	Running mode	95
3.16.2.1	Example description	80	3.19.1.5	Step to run	95
3.16.2.2	Directory contents	81	3.19.2	Rtc_Calibration	96

continued >>

3.19.2.1	Example description	96	3.23.1.4	Running mode	110
3.19.2.2	Directory contents	96	3.23.1.5	Step to run	110
3.19.2.3	Hardware configuration	97	3.23.2	Uart_Interrupt	111
3.19.2.4	Running mode	97	3.23.2.1	Example description	111
3.19.2.5	Step to run	97	3.23.2.2	Directory contents	112
3.20	SSP (SYNCHRONOUS SERIAL PORT)	98	3.23.2.3	Hardware configuration	112
3.20.1	Ssp_Dma	98	3.23.2.4	Running mode	112
3.20.1.1	Example description	98	3.23.2.5	Step to run	113
3.20.1.2	Directory contents	98	3.23.3	Uart_Dma	113
3.20.1.3	Hardware configuration	98	3.23.3.1	Example description	113
3.20.1.4	Running mode	99	3.23.3.2	Directory contents	114
3.20.1.5	Step to run	99	3.23.3.3	Hardware configuration	114
3.21	SysTick (SYSTEM TICK TIMER)	99	3.23.3.4	Running mode	114
3.21.1	Systick_10msBase	99	3.23.3.5	Step to run	114
3.21.1.1	Example description	99	3.23.4	Uart_Autobaud	115
3.21.1.2	Directory contents	100	3.23.4.1	Example description	115
3.21.1.3	Hardware configuration	100	3.23.4.2	Directory contents	115
3.21.1.4	Running mode	100	3.23.4.3	Hardware configuration	115
3.21.1.5	Step to run	100	3.23.4.4	Running mode	116
3.21.2	Systick_Stclk	101	3.23.4.5	Step to run	116
3.21.2.1	Example description	101	3.23.5	Uart_FullModem	117
3.21.2.2	Directory contents	101	3.23.5.1	Example description	117
3.21.2.3	Hardware configuration	102	3.23.5.2	Directory contents	117
3.21.2.4	Running mode	102	3.23.5.3	Hardware configuration	118
3.21.2.5	Step to run	102	3.23.5.4	Running mode	118
3.22	TIMER	103	3.23.5.5	Step to run	118
3.22.1	Timer_MatchInterrupt	103	3.23.6	Uart_IrdaTransmit	119
3.22.1.1	Example description	103	3.23.6.1	Example description	119
3.22.1.2	Directory contents	104	3.23.6.2	Directory contents	120
3.22.1.3	Hardware configuration	104	3.23.6.3	Hardware configuration	120
3.22.1.4	Running mode	104	3.23.6.4	Running mode	120
3.22.1.5	Step to run	104	3.23.6.5	Step to run	120
3.22.2	Timer_Capture	105	3.23.7	Uart_IrdaReceive	121
3.22.2.1	Example description	105	3.23.7.1	Example description	121
3.22.2.2	Directory contents	105	3.23.7.2	Directory contents	122
3.22.2.3	Hardware configuration	106	3.23.7.3	Hardware configuration	122
3.22.2.4	Running mode	106	3.23.7.4	Running mode	122
3.22.2.5	Step to run	106	3.23.7.5	Step to run	122
3.22.3	Timer_FreqMeasure	107	3.23.8	Uart_Rs485Master	123
3.22.3.1	Example description	107	3.23.8.1	Example description	123
3.22.3.2	Directory contents	108	3.23.8.2	Directory contents	123
3.22.3.3	Hardware configuration	108	3.23.8.3	Hardware configuration	123
3.22.3.4	Running mode	108	3.23.8.4	Running mode	123
3.22.3.5	Step to run	108	3.23.8.5	Step to run	123
3.23	UART (UNIVERSAL ASYNCHRONOUS RECEIVER/TRANSMITTER)	109	3.23.9	Uart_Rs485Slave	124
3.23.1	Uart_Polling	109	3.23.9.1	Example description	124
3.23.1.1	Example description	109	3.23.9.2	Directory contents	125
3.23.1.2	Directory contents	110	3.23.9.3	Hardware configuration	125
3.23.1.3	Hardware configuration	110	3.23.9.4	Running mode	125
			3.23.9.5	Step to run	125

continued >>

3.24	USB DEVICES	126	3.25.1.5	Step to run	132
3.24.1	Usb_MassStorage	126	3.25.2	Wdt_Reset	134
3.24.1.1	Example description	126	3.25.2.1	Example description	134
3.24.1.2	Directory contents	126	3.25.2.2	Directory contents	134
3.24.1.3	Hardware configuration	126	3.25.2.3	Hardware configuration	135
3.24.1.4	Running mode	127	3.25.2.4	Running mode	135
3.24.1.5	Step to run	127	3.25.2.5	Step to run	135
3.24.2	Usb_VirtualCom	128	3.25.3	Wdt_WindowMode	136
3.24.2.1	Example description	128	3.25.3.1	Example description	136
3.24.2.2	Driver installation	128	3.25.3.2	Directory contents	136
3.24.2.3	Directory contents	129	3.25.3.3	Hardware configuration	137
3.24.2.4	Hardware configuration	129	3.25.3.4	Running mode	137
3.24.2.5	Running mode	129	3.25.3.5	Step to run	137
3.24.2.6	Step to run	129	4.	Legal information	139
3.25	WDT (WATCHDOG TIMER)	132	4.1	Definitions	139
3.25.1	Wdt_Interrupt	132	4.2	Disclaimers	139
3.25.1.1	Example description	132	4.3	Trademarks	139
3.25.1.2	Directory contents	132	5.	Contents	140
3.25.1.3	Hardware configuration	132			
3.25.1.4	Running mode	132			

continued >>