AN10863

LPC17xx example description

Rev. 05 - 16 July 2010 **Application note**

Document information

Info	Content
Keywords	LPC17xx, MCB1700, IAR LPC1768
Abstract	Describe the examples for testing each peripheral functions of LPC17xx in LPC1700CMSIS Library package.

Revision history

Rev	Date	Description
05	20100716	Fifth version – Add more examples description
04	20100618	Fourth version – Add more examples, pictures, remove debug_frmwrk.h/.c in directory contens
03	20100521	Third version –Change example description form, change bookmark
02	20091204	Secondary version
01	20090828	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 will describe the example code of LPC1700CMSIS library package that used to test each peripheral function of LPC17xx.

2. Additional condition

2.1 Running board

LPC1700CMSIS examples tested on:

- MCB1700 board with LPC1768 version 1.0
- IAR LPC1768 KickStart board version A

2.2 Serial display

All the example codes require UART for display, and must be configured with the following condition:

Common communication is UART0 port with parameter below:

- 115200 bps
- 8 databit
- None parity
- 1 stopbit
- No flow control

In order to use serial display function, include “debug_frmwrk” header file 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’

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

Software used on window to communicate via UART is HyperTerminal configured with parameters configured above:

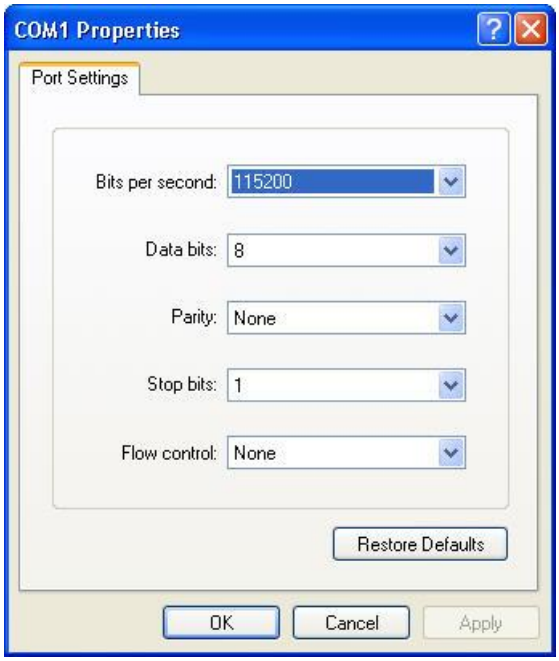

Fig 1. COM port setting on Window

Note: DO NOT USE HYPERTERMINAL if the 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). Error is as shown in Fig 2 when running the DMA example code.

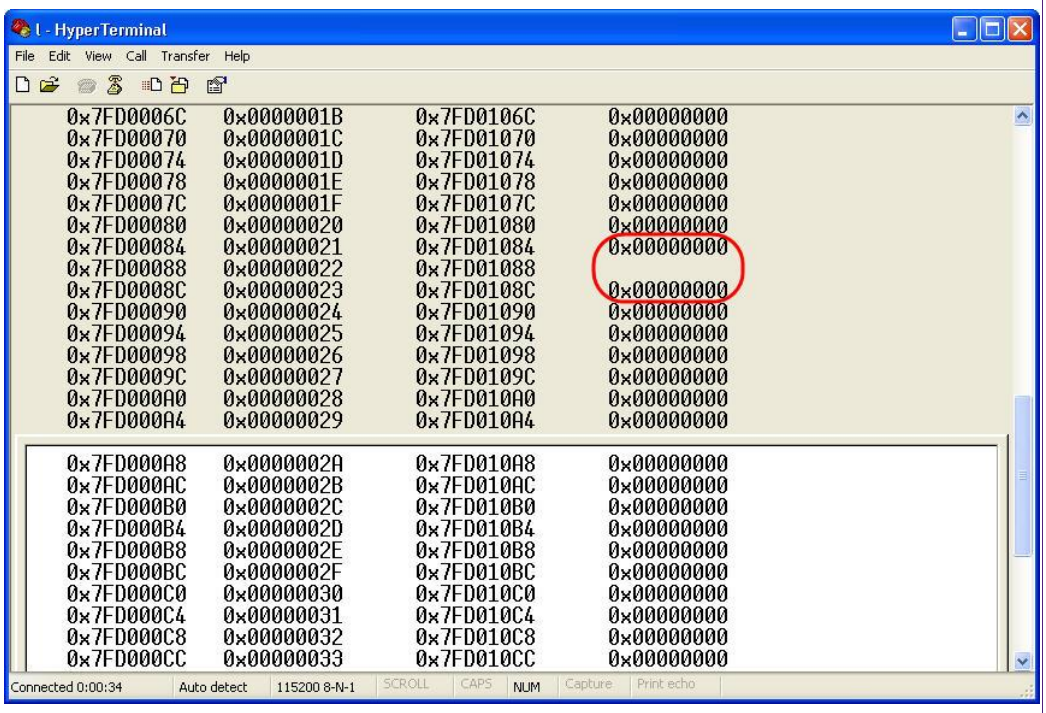

Fig 2. Some characters are lost

The solution to this is to use another communication tool, such as Flash magic, or TeraTerm.

Note: With Flash Magic, configure newlines: CR

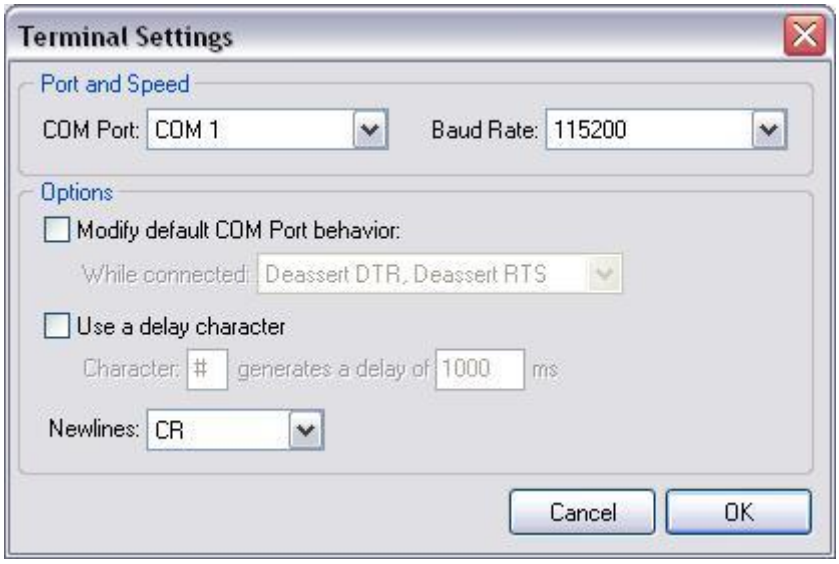

Fig 3. COM configuration on Flash Magic

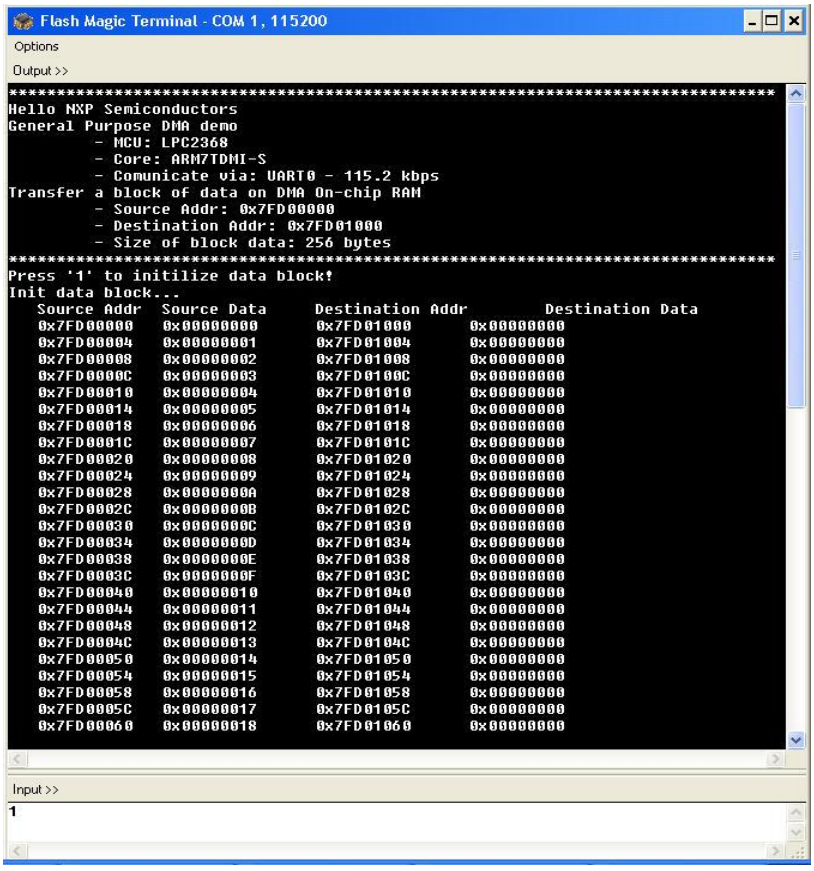

Fig 4. Terminal on Flash magic is smooth but response time of display is slow.

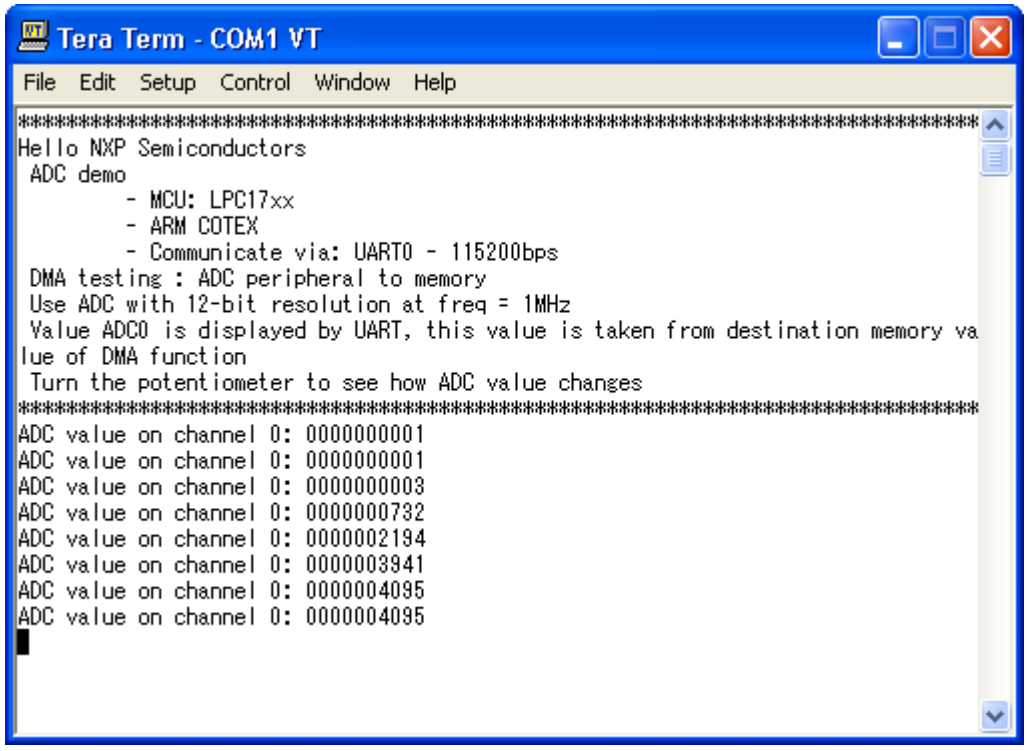

Fig 5. Teraterm IDE

2.3 Additional files requirement

A driver library configuration file must be also included in each example to enable/disable specified peripheral driver:

lpc17xx_libcfg.h

2.4 Running mode

Each example can be used to run in RAM mode, ROM (FLASH) mode or both. Please reference 'Running mode' section to know what mode that example can run.

In GNU toolchain, to reduce code size, you can use '**--gc-section**' flag to remove unused section. By uncommenting this flag at '..\makesection\LPC17xx\make.LPC17xx.gnu', the code size will be reduced, however it can affect debugging accuracy. Hence this is only recommended to be used in standalone mode

```
LK                  =  -static -mcpu=cortex-m3 -mthumb -mthumb-interwork
#LK                 = -static
#LK                 += -Wl,--start-group $(TARGET_FWLIB_LIB)
LK                  += -Wl,--start-group
LK                        += -L$(THUMB2GNULIB) -L$(THUMB2GNULIB2)
LK                  += -lc -lg -lstdc++ -lsupc++
LK                        += -lgcc -lm
LK                  += -Wl,--end-group
LK                  += -Wl,--gc-sections -Wl,--print-gc-sections
```

Fig 6. Uncomment '—gc-secions' flag to reduce code

Keil and IAR remove unused sections automatically. So don't care about it if run examples in Keil and IAR

2.4.1 RAM mode

All files in each example must be built to .elf file, this file will be loaded in to RAM through a debugger tool before running.

2.4.2 ROM (FLASH) mode

All files in each example must be built to .hex file; this file will be burned in to ROM (FLASH) memory through an external tool (i.e. Flash Magic, J-Flash…) before running.

Note: If want to burn hex file into board by using Flash Magic tool, these jumpers must be configured as follows:

- MCB1700 board:
  - RST: ON
  - ISP: ON
- IAR1700 board:
  - RST_E: ON
  - ISP_E: ON

But UART0 function could not operate normally when either RST jumper or ISP jumper are ON. So these jumper should be removed when running.

Please refer to the "LPC17xx_SoftwareDevelopmentToolchain" document for more details.

3. Examples

3.1 ADC

3.1.1 Burst

3.1.1.1 Example description

Purpose

This example describes how to use ADC conversion in burst mode with single or multiple input.

This example also describes how to inject an ADC conversion channel while the others are running.

Process

Because the potentiometer on different board connect to different ADC channel, so we have to configure correct

ADC channel on each board respectively. In this case,

- If using MCB1700 board: ADC is configured to convert signal on channel 2
- If using IAR-LPC1768-KS board: ADC is configured to convert signal on channel 5

The ADC conversion rate is 200KHz. A fully A fully accurate conversion requires 65 of these clocks.

So ADC clock = 200KHz * 65 = 13MHz.

Note that maximum ADC clock input is 13MHz.

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 UART0.

Turn potentiometer to change ADC signal input.

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

This example uses GPIO interrupt on P2.10 to trigger the injection, and uses P1.28 to toggle the LED indicating that an ADC channel has just been injected or removed.

3.1.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

adc_burst_test.c: Main program file

3.1.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON

- VDDREG: ON
- VBUS: ON
- AD0.2: ON
- SPK: OFF
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

ADC connection

For multiple inputs:

- The ADCx sample pin must not left floating, otherwise, the readout 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

3.1.1.4 Running mode

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

3.1.1.5 Step to run

Step 1: Choose correct working board by uncomment correct defined board in adc_burst_test.c file

- If using MCB1700 board, uncomment "#define MCB_LPC_1768"
  - If test multiple input, then uncomment "#define MCB_LPC_1768_ADC_BURST_MULTI"
  - If test injection, then uncomment:
    - "#define MCB_LPC_1768_ADC_BURST_MULTI"
    - "#define MCB_LPC17XX_ADC_INJECT_TEST"
- If using IAR-LPC1768-KS board, uncomment "#define MCB_LPC_1768"

  (Should not uncomment both symbols at the same time)

Step 2: Build example.

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect UART0 on this board to COM port on your computer

Step 5: Configure hardware and serial display as above instruction

Step 6: Run example, turn potetiometer and observe data on serial display

- For injection test:
  - Press the INT0 button (on MCB1700 board) to see the state of LED P1.28:
    - ON: ADC channel 3 has just been inserted.
    - OFF: ADC channel 3 has just been removed.

- Look at the PC's terminal application screen, associate with the state of LED P1.28:
  - ON: the ADC channel 3 value is continuosly updated.
  - OFF: the ADC channel 3 value is unchanged, keep its last updated value.

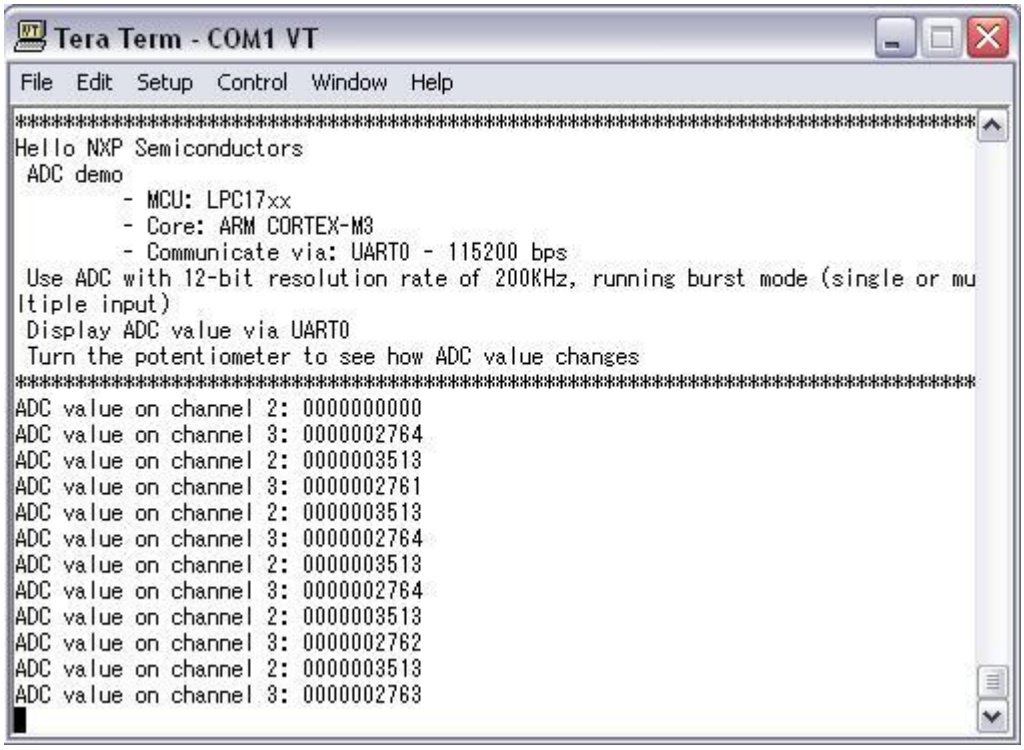

Fig 7. ADC burst mode demo

3.1.2 DMA

3.1.2.1 Example description

Purpose

This example describes how to use ADC conversions and transfer converted data by using DMA.

Process

Because the potentiometer on different board connect to different ADC channel, so we have to configure correct

ADC channel on each board respectively. In this case,

- MCB1700 board: ADC is configured to convert signal on channel 2
- IAR-LPC1768-KS board: ADC is configured to convert signal on channel 5

DMA channel 0 was configured in this example.

The ADC conversion rate is 200KHz. A fully accurate conversion requires 65 of these clocks.

So ADC clock = 200KHz * 65 = 13MHz.

Note: maximum ADC clock input is 13MHz.

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 display this data into serial display via UART0, DMA re-setup for next transfer.

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

adc_dma_test.c: Main program file

3.1.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- AD0.2: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.1.2.4 Running mode

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

3.1.2.5 Step to run

Step 1: Choose working board by uncomment correct defined board in adc_dma_test.c file:

- MCB1700 board, uncomment “#define MCB_LPC_1768”
- IAR-LPC1768-KS board, uncomment “#define IAR_LPC_1768”

```
//#define MCB_LPC_1768
#define IAR_LPC_1768
```

```
#define MCB_LPC_1768
//#define IAR_LPC_1768
```

Fig 8. Choose working board

Step 2: Build example

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect UART0 on this board to COM port on your computer

Step 5: Configure hardware and serial display as above instruction

Step 6: Run example, turn potetiometer and observe data on serial display

The screen will display like this:

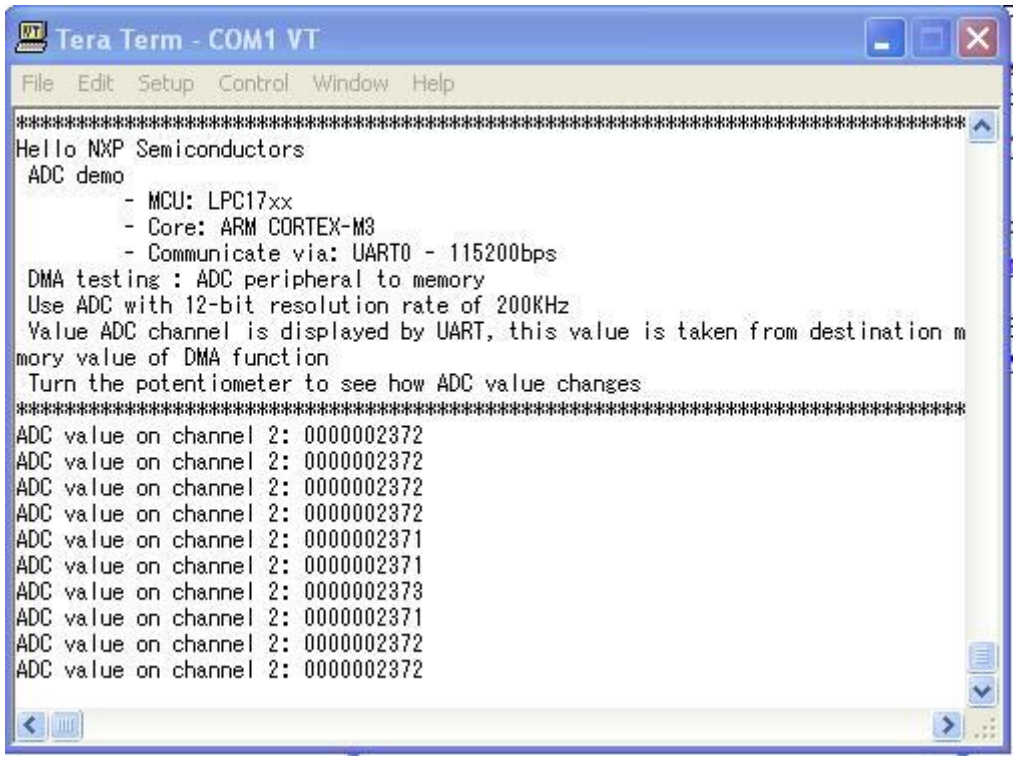

Fig 9. ADC DMA demo

3.1.3 HardwareTrigger

3.1.3.1 Example description

Purpose

This example describes how to use ADC conversion in hardware-triggered mode.

Process

The ADC conversion rate is 200KHz. A fully A fully accurate conversion requires 65 of these clocks.

So ADC clock = 200KHz * 65 = 13MHz.

Note that maximum ADC clock input is 13MHz.

When INT0 falling adge occurs, ADC will start converting.

ADC will generate interrupt at the end of each conversion. ADC service routine will be invoked to check ADC status, if "DONE bit" 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 via UART0.

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

adc_hardware_trigger_test.c: Main program file

3.1.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- AD0.2: ON
- INT0: ON
- Remain jumper: OFF

3.1.3.4 Running mode

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

3.1.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example

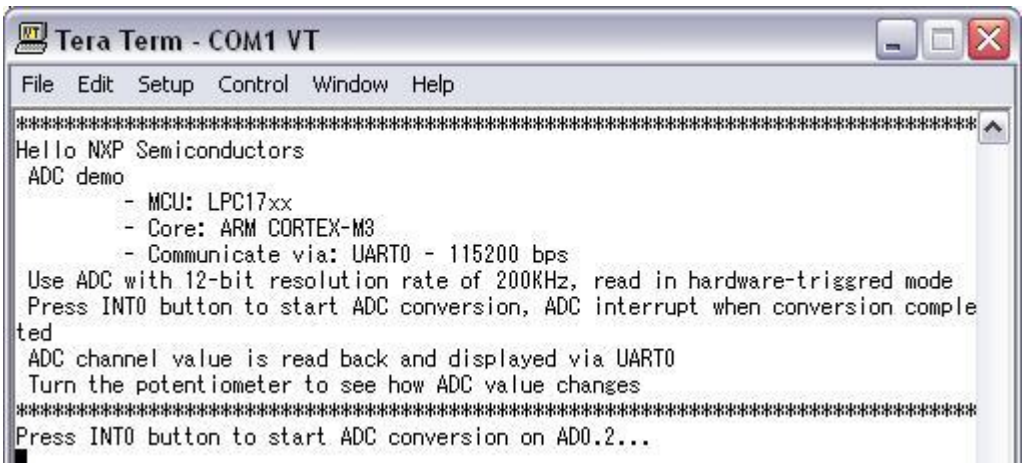

Fig 10. ADC Hardware Trigger welcome screen

Press INT0 button to start the ADC conversion

Turn potetiometer and press INT0 button again then observe data on serial display

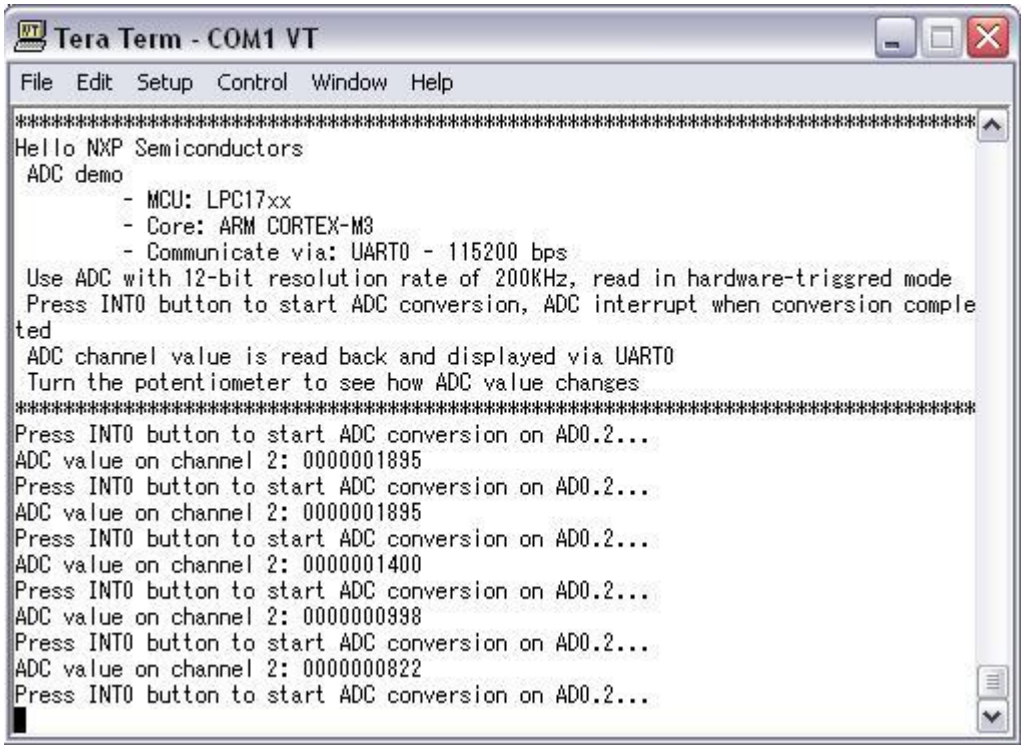

Fig 11. ADC hardware trigger demo

3.1.4 Interrupt

3.1.4.1 Example description

Purpose

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

Process

Because the potentiometer on different board connect to different ADC channel, so we have to configure correct ADC channel on each board respectively. In this case,

- MCB1700 board: ADC is configured to convert signal on channel 2
- IAR-LPC1768-KS board: ADC is configured to convert signal on channel 5

The ADC conversion rate is 200KHz. A fully A fully accurate conversion requires 65 of these clocks.

So ADC clock = 200KHz * 65 = 13MHz.

Note that maximum ADC clock input is 13MHz.

ADC will generate interrupt at the end of each conversion. ADC service routine will be invoked to check ADC status, if "DONE bit" 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 via UART0.

Turn potentiometer to change ADC signal input.

3.1.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

adc_interrupt_test.c: Main program file

3.1.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- AD0.2: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.1.4.4 Running mode

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

3.1.4.5 Step to run

Step 1: Choose working board by uncomment correct defined board in adc_interrupt_test.c file:

- MCB1700 board, uncomment “#define MCB_LPC_1768”
- IAR-LPC1768-KS board, uncomment “#define IAR_LPC_1768”

```
//#define MCB_LPC_1768
#define IAR_LPC_1768
```

```
#define MCB_LPC_1768
//#define IAR_LPC_1768
```

Fig 12. Choose working board

Step 2: Build example

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect UART0 on this board to COM port on your computer

Step 5: Configure hardware and serial display as above instruction

Step 6: Run example, turn potetiometer and observe data on serial display

The screen will display like this:

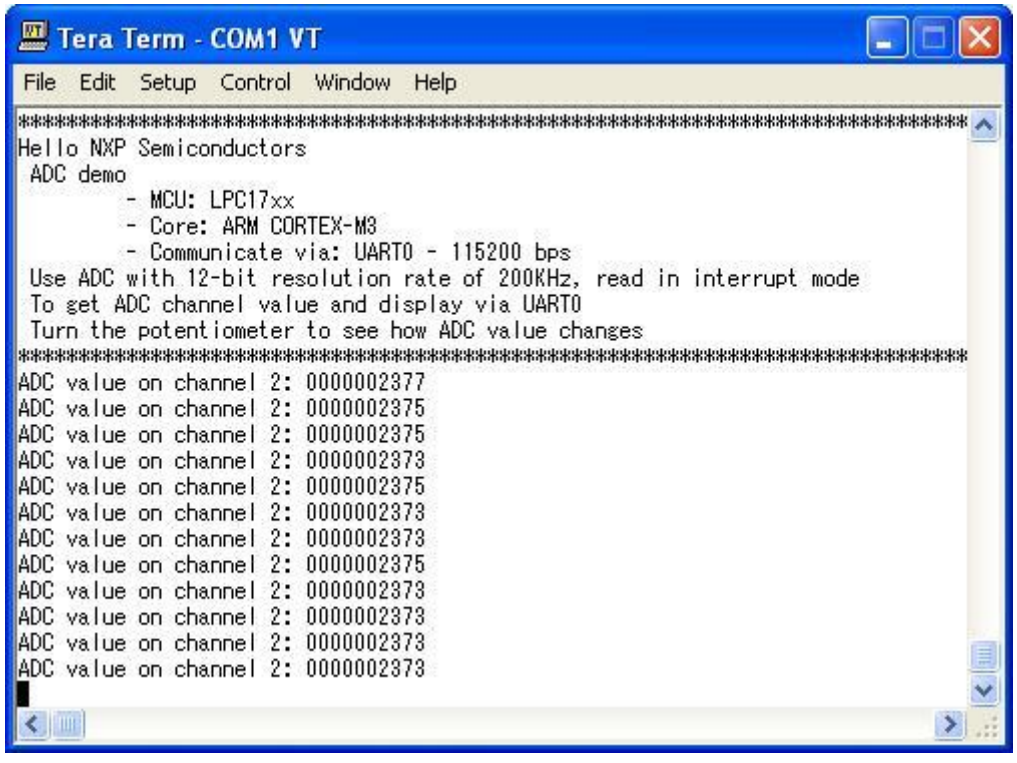

Fig 13. ADC interrupt demo

3.1.5 Polling

3.1.5.1 Example description

Purpose

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

Process

Because the potentiometer on different board connect to different ADC channel, so we have to configure correct

ADC channel on each board respectively. In this case,

- MCB1700 board: ADC is configured to convert signal on channel 2
- IAR-LPC1768-KS board: ADC is configured to convert signal on channel 5

DMA channel 0 was configured in this example.

The ADC conversion rate is 200KHz. A fully A fully accurate conversion requires 65 of these clocks.

So ADC clock = 200KHz * 65 = 13MHz.

Note that maximum ADC clock input is 13MHz.

After start ADC operation, polling "DONE" bit. If "DONE" bit is set, store ADC converted data in "adc_value" and display this data via UART0 and re-start ADC for next conversion.

Turn potentiometer to change ADC signal input.

3.1.5.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

adc_interrupt_test.c: Main program file

3.1.5.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- AD0.2: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source

- DBG_EN: ON
- Remain jumper: OFF

3.1.5.4 Running mode

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

3.1.5.5 Step to run

Step 1: Choose working board by uncomment correct defined board in adc_polling_test.c file:

+ Using MCB1700 board, uncomment “#define MCB_LPC_1768”

+ Using IAR-LPC1768-KS board, uncomment “#define IAR_LPC_1768”

```
//#define MCB_LPC_1768
#define IAR_LPC_1768
```

```
#define MCB_LPC_1768
//#define IAR_LPC_1768
```

Fig 14. Choose working board

Step 2: Build example

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect UART0 on this board to COM port on your computer

Step 5: Configure hardware and serial display as above instruction

Step 6: Run example, turn potetiometer and observe data on serial display

The screen will display like this:

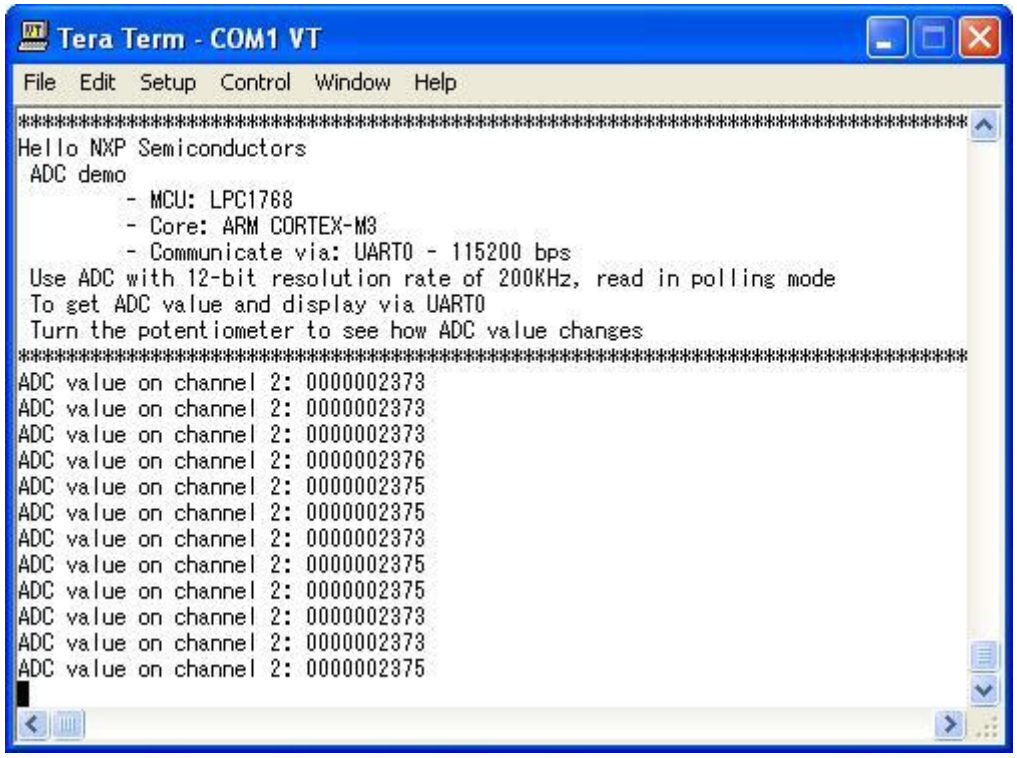

Fig 15. ADC polling demo

3.2 CAN

3.2.1 CAN_LedControl

3.2.1.1 Example description

Purpose

This example describes how to use CAN frames to control LED display.

Process

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

Two CAN used in bypass mode.

User will enter LED display value on serial display and this value will be saved

in TXMsg, this message will be sent from CAN1 to CAN2.

When CAN2 receive this message, it will display LEDs according to value in this message.

3.2.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

can_ledcontrol.c: Main program file

3.2.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

CAN connection

- CAN1-Pin2 connects to CAN2-Pin2 (CAN-L)
- CAN1-Pin7 connects to CAN2-Pin7 (CAN-H)

(See Fig18. CAN pin connection)

3.2.1.4 Running mode

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

3.2.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example

The screen will be displayed like this:

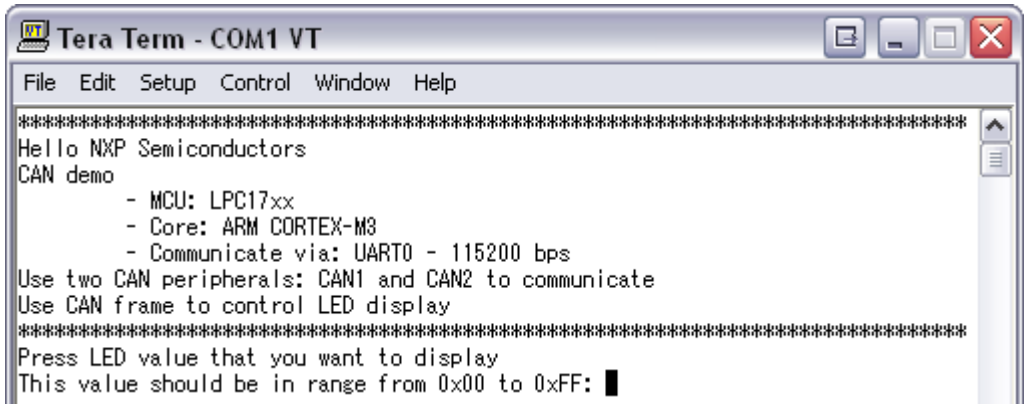

Fig 16. Welcome screen

- Press LED value on serial display. This value should be in range: 0x00 - 0xFF

Such as in this case, we typed: 55

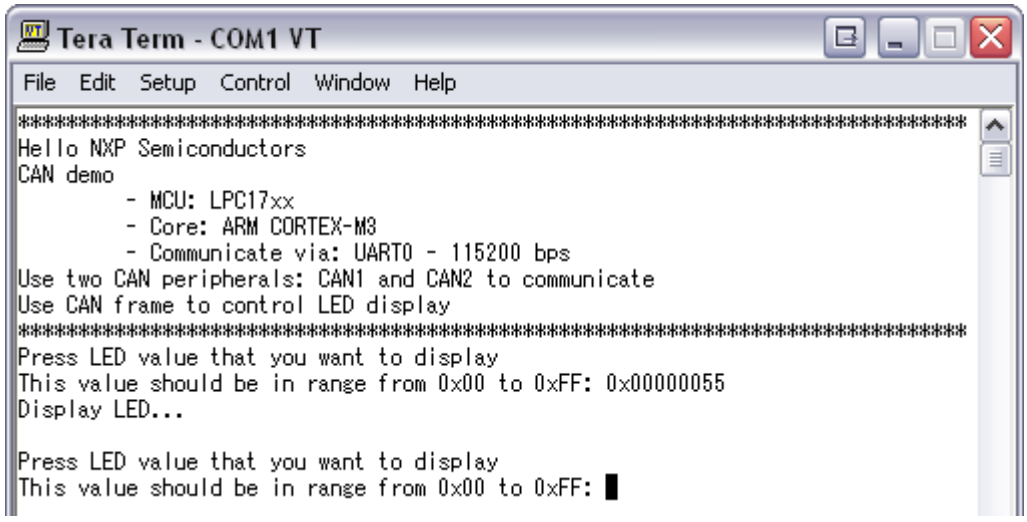

Fig 17. Welcome screen

- See LED displaying on board.

3.2.2 CAN_self_test

3.2.2.1 Example description

Purpose

This example describes how to test CAN self-test mode

Process

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

With the Self test mode, the transmitted message is also received and stored in the receive 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.

Pls observe process via serial display.

3.2.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

can_self_test.c: Main program file

3.2.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

CAN connection

No connection required

3.2.2.4 Running mode

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

3.2.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example

After reset, welcome screen appears like this:

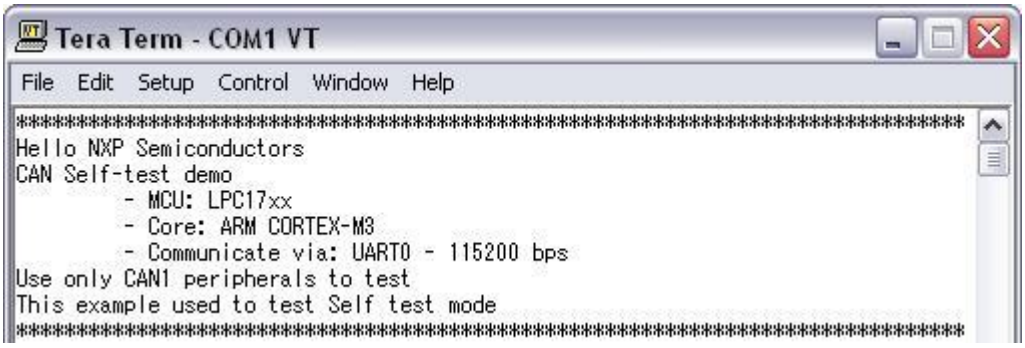

Fig 18. Welcome screen

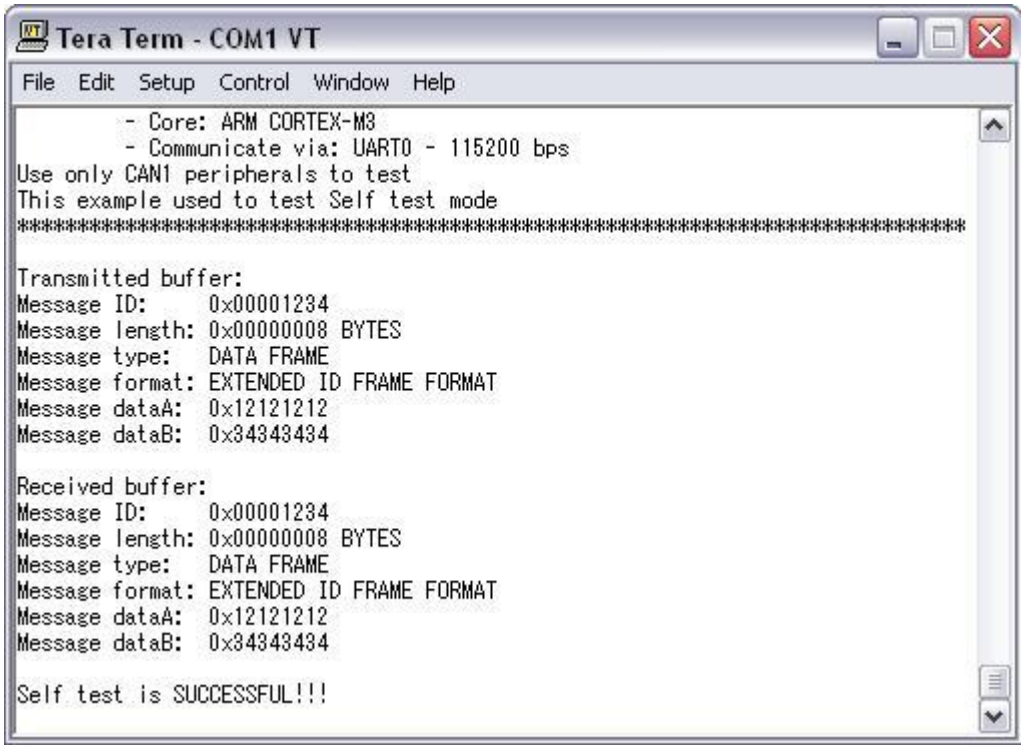

Fig 19. Welcome screen

3.2.3 CAN_test_aflut

3.2.3.1 Example description

Purpose

This example describes how to use CAN driver functions for setup and change AFLUT table dynamically.

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, settup 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 "AFRxMsg[]".

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 2,4,6,9,10 instead of 1,3,5,7,9.

Re-send 10 messages and re-received messages to check if AFLUT operation 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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

CAN_test_aflut.c: Main program file

3.2.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

CAN connection

- CAN1-Pin2 connects to CAN2-Pin2 (CAN-L)
- CAN1-Pin7 connects to CAN2-Pin7 (CAN-H)

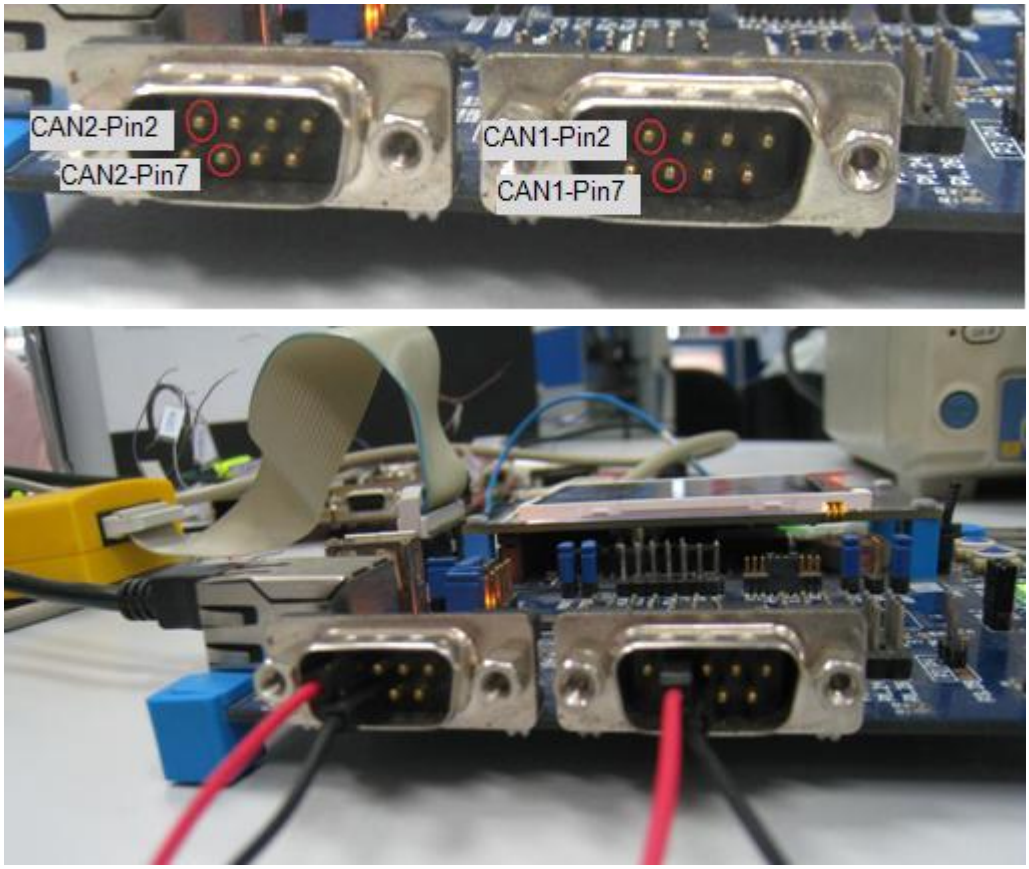

Fig 20. CAN pin connection

3.2.3.4 Running mode

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

3.2.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example:

After reset, the welcome screen appears like this:

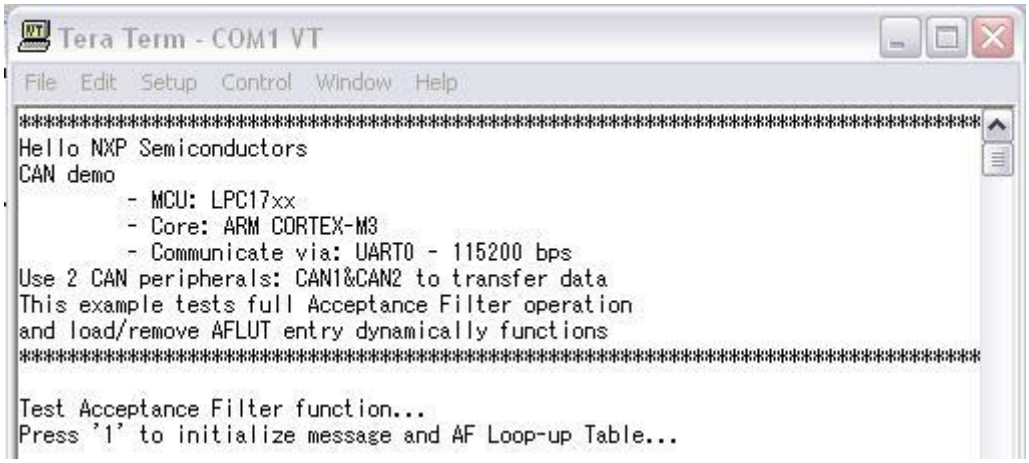

Fig 21. Welcome screen

- Press "1" to initialize message and AFLUT

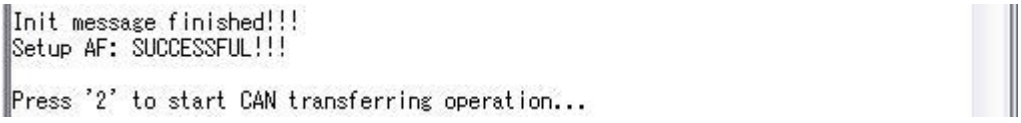

Fig 22. Initialize message

- Press "2" to start CAN operation

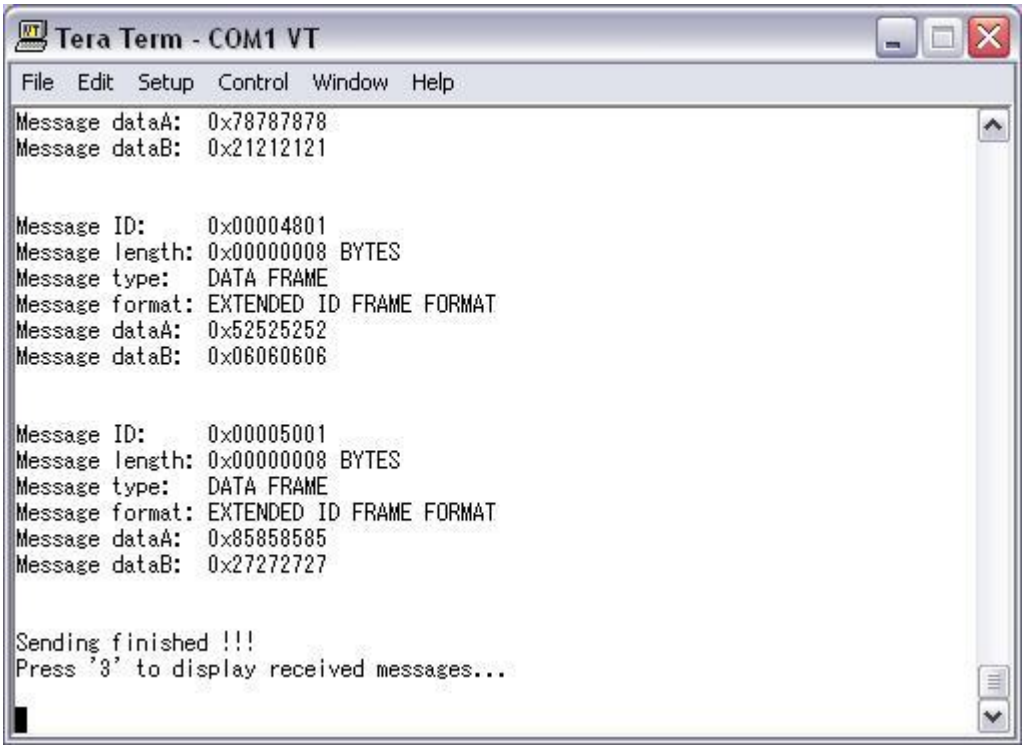

Fig 23. Send messages

- Press "3" to display received messages

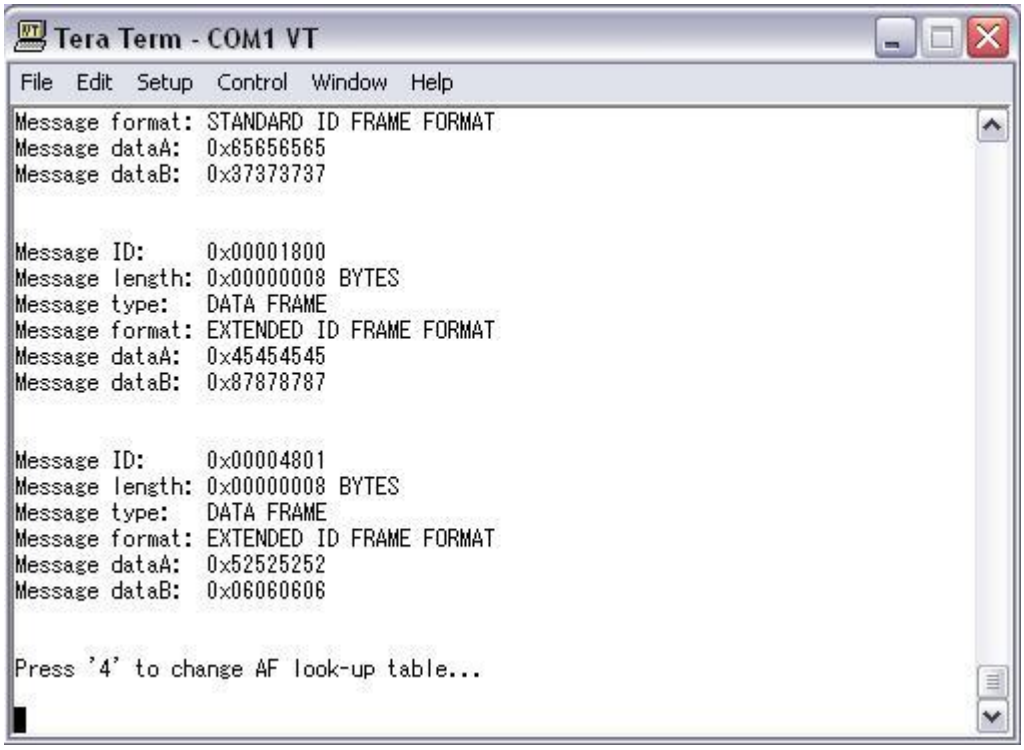

Fig 24. Received messages

- Press "4" to change AFLUT

Fig 25. Change AFLUT

- Press "5" to re-send message

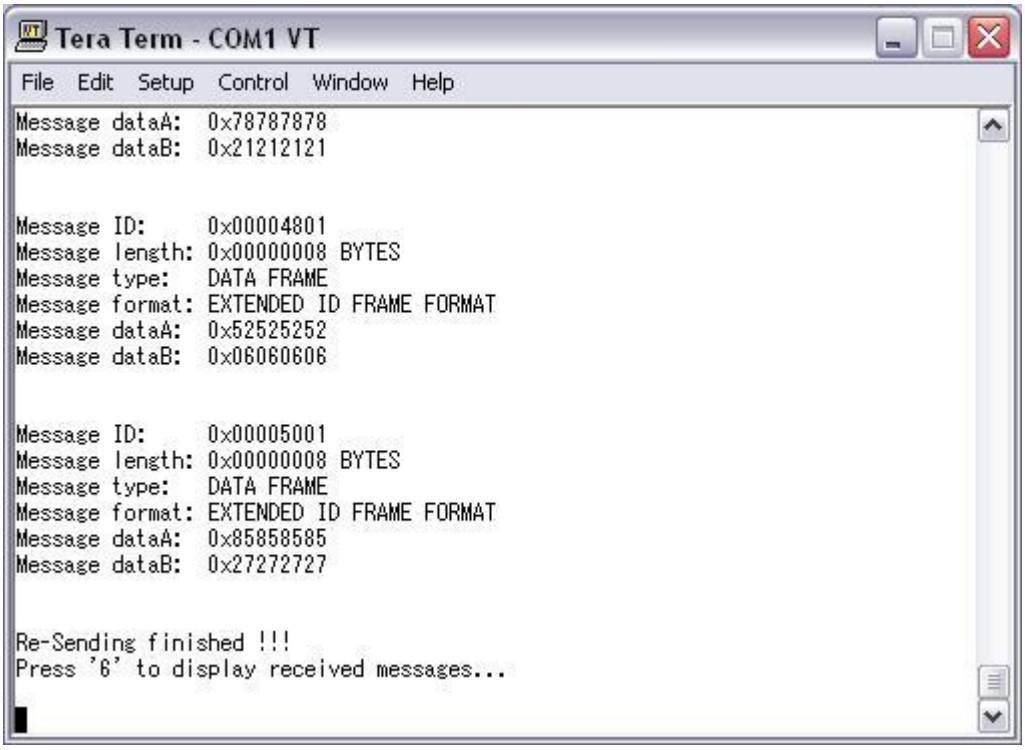

Fig 26. Re-send messages

- Press "6" to display received messages

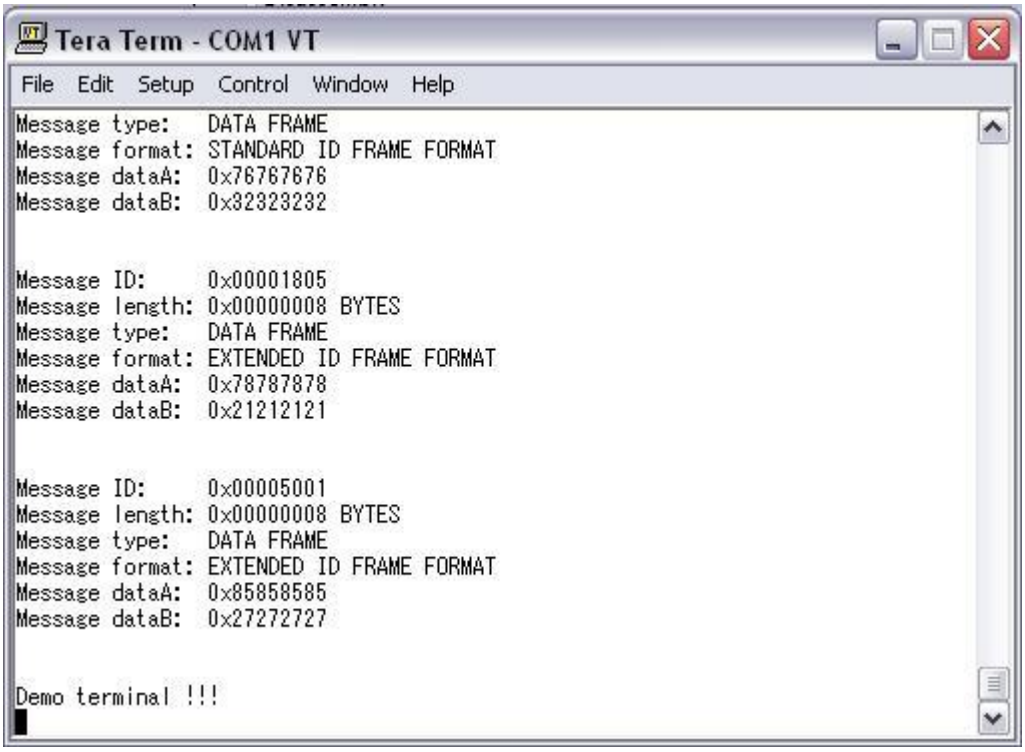

Fig 27. Re-receive message

3.2.4 CAN_test_bypass_mode

3.2.4.1 Example description

Purpose

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

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 disable, 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 transmittion by CAN1.This process is a endless loop.

Open serial display to observe CAN transfer processing.

3.2.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

CAN_test_bypass_mode.c: Main program file

3.2.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

CAN connection

- CAN1-Pin2 connects to CAN2-Pin2 (CAN-L)
- CAN1-Pin7 connects to CAN2-Pin7 (CAN-H)

(See “CAN_test_aflut” example for more information)

3.2.4.4 Running mode

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

3.2.4.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example

After reset, welcome screen appears like this:

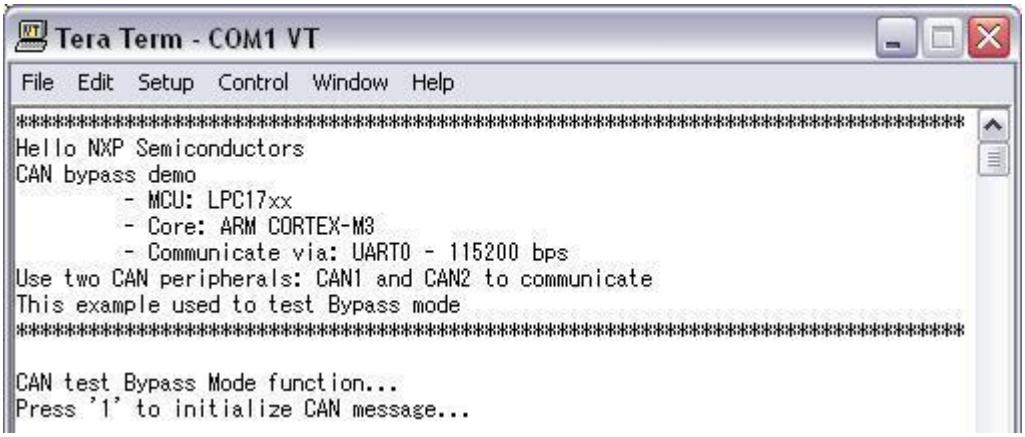

Fig 28. Welcome screen

- Press "1" to initialize message and AFLUT

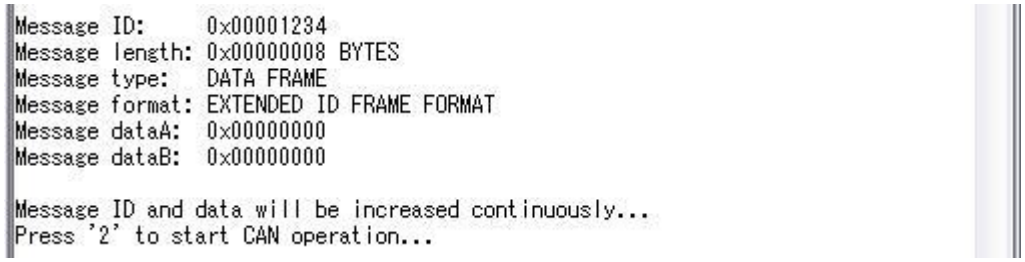

Fig 29. Initialize first messages

- Press "2" to start CAN operation

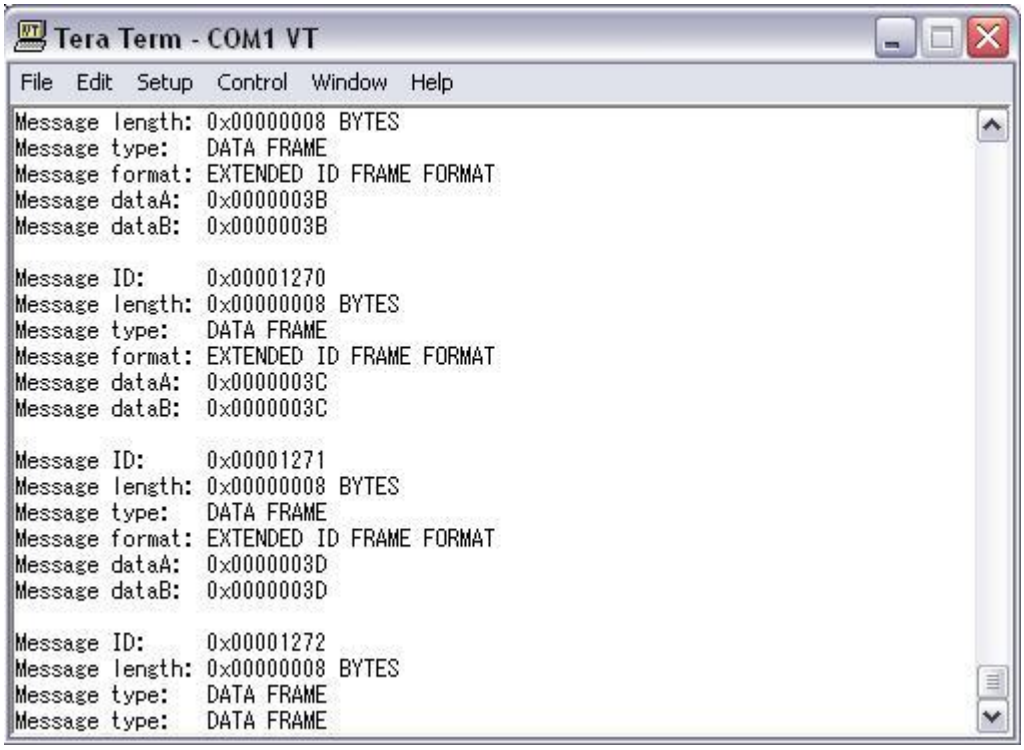

Fig 30. Received messages in bypass mode

3.2.5 CAN_test_two_kit

3.2.5.1 Example description

Purpose

This example describes how to configure CAN operation in two boards seperately

Process

Using 2 CAN1 peripheral on two boards to test.

This example just supports explicit standard/extended ID format.

Both CAN1 peripherals are set baudrate at 125KHz

CAN transmit:

Initialize 4 CAN messages:

- 1st message with 11-bit ID which exit in AF Look-up Table
- 2nd message with 11-bit ID which not exit in AF Look-up Table
- 3th Message with 29-bit ID which exit in AF Look-up Table
- 4th Message with 29-bit ID which exit in AF Look-up Table

These messages will be send to CAN peripheral in another board

CAN receive:

Setup a simple AFLUT. It just has two sections:

- Explicit Standard Frame Format Identifier Section
- Explicit Extended Frame Format Identifier Section

Receive message from another board. Whenever CAN1 receive message that has ID exit in its AFLUT, receive interrupt occurs, CAN service routine will be invoked to receive message and save its data in array "AFRxMsg[]" and print to serial display via UART0 port.

In this case, messages 1,3 will be received.

Open serial display to observe CAN transfer processing.

3.2.5.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

CAN_test_two_kit.c: Main program file

3.2.5.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

CAN connection

- Connect two pins CAN1-Pin2 between two boards
- Connect two pins CAN1-Pin7 between two boards

(See pictures of “CAN_test_aflut” example for more information)

3.2.5.4 Running mode

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

3.2.5.5 Step to run

Step 1: Setting "CAN_TRANSMIT = 1" (in main.c file) -> build and burn hex file in the board transmitter

Step 2: Setting "CAN_TRANSMIT = 0" (in main.c file) -> build and burn hex file in the board receiver

```
#define CAN_TRANSMIT                0
#define CAN_RECEIVE                 !CAN_TRANSMIT
#define TX_BUFFER_SIZE              4
#define RX_BUFFER_SIZE              2
```

Fig 31. Setting macro

Note: receive program also can run in RAM mode. In this case, not burn hex file but run it with debugger.

Step 3: Connect UART0 on board receiver to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example:

- Re-set board receiver.

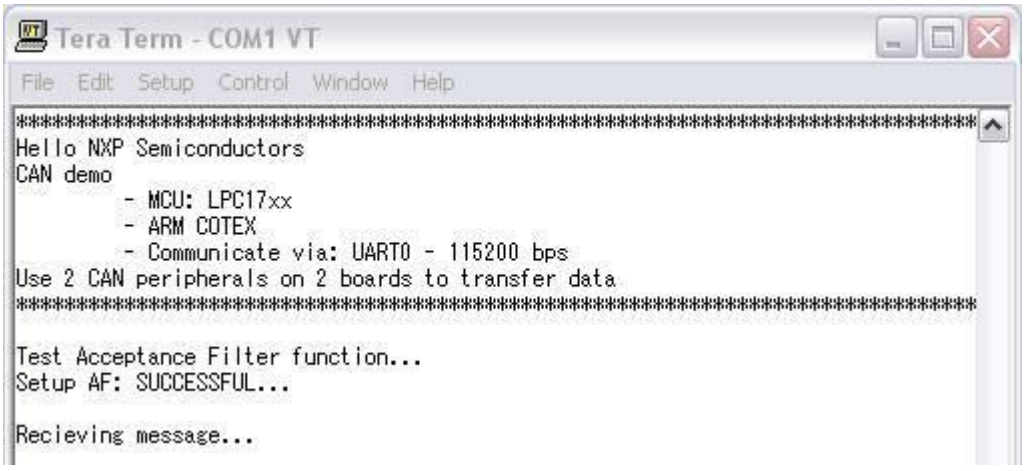

Fig 32. CAN MASTER welcome screen

- Re-set board transmitter. After reset, CAN sends messages immediately…

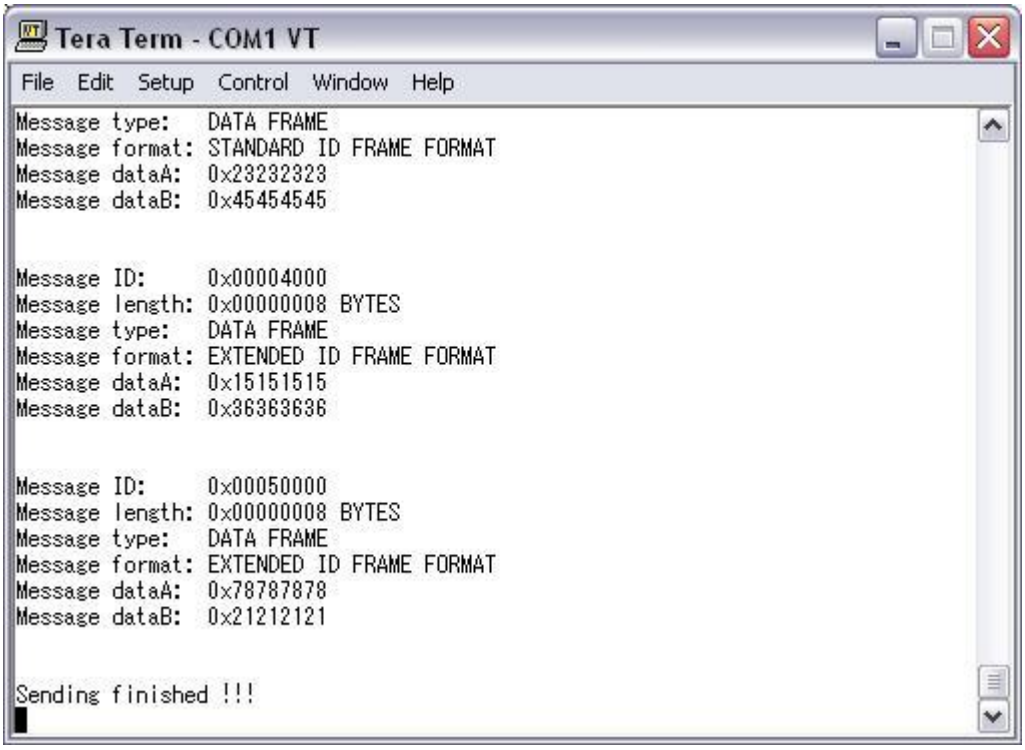

Fig 33. CAN SLAVE sends messages

At received board side, received message will be displayed:

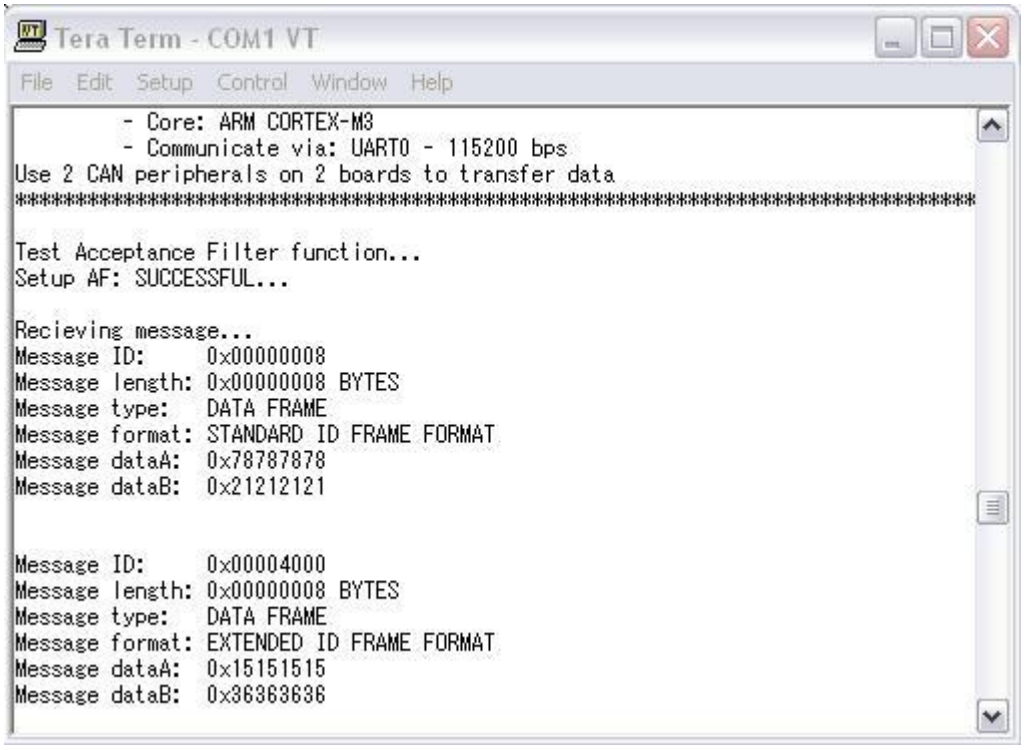

Fig 34. CAN messages were received after transferring finished

3.3 Cortex-M3

3.3.1 Bit-banding

3.3.1.1 Example description

Purpose

This example describes how to test Bit-banding feature of Cortex-M3 processor

Process

The processor memory map includes two bit-band regions. These occupy the lowest 1MB of the SRAM and peripheral memory regions respectively.

- SRAM
  - Bit-band region: 0x20000000 - 0x20100000
  - Bit-band alias: 0x22000000 - 0x23FFFFFF
- PERIPHERAL
  - Bit-band region: 0x40000000 - 0x40100000
  - Bit-band alias: 0x42000000 - 0x43FFFFFF

The mapping formula:

bit_word_offset = (byte_offset * 32) + (bit_number * 4)

bit_word_address = bit_band_base + bit_word_offset

where:

- bit_word_offset: the position of the target bit in the bit-band memory region
- bit_word_addr: the address of the word in the alias memory region that maps to the target bit
- bit_band_base: the starting address of the alias region
- byte_offset: the number of byte in the bit-band region that contains the targeted bit
- bit_number: is the bit position (0-7) of the targeted bit

Note: In fact, the SRAM on LPC1768 just available in two ranges:

- 0x2007C000 - 0x2007FFFF: for SRAM – bank 0
- 0x20080000 – 0x20083FFF: for SRAM – bank 1

Just can set 'VAR_ADDR' with value that exits in two ranges above.

Beside, the range: 0x2009C000 - 0x2009FFF is assigned for GPIO peripheral. So you can use VAR_ADDR in this range for bit-modifying GPIO registers.

First, the program test SRAM bit-banding:

- read the value at VAR_ADDRESS

Using bit-band access to:

- read the value at 'VAR_BIT'.
- clear the value at 'VAR_BIT' and print new value at VAR_ADDRESS
- re-set the value at 'VAR_BIT' and print new value at VAR_ADDRESS

Then, the program test PERIPHERAL bit-banding:

In this case, use SPCR register for testing

- Assigned value SPCR: 0x00000A28

Use bit-band access to:

- read the value at 'PERI_BIT'
- clear the value at 'PERI_BIT' and print new value of SPCR
- re-set the value at 'PERI_BIT' and print new value of SPCR

3.3.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

bitband.c: Main program file

3.3.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.3.1.4 Running mode

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

3.3.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe result on serial display

The screen will be displayed like this:

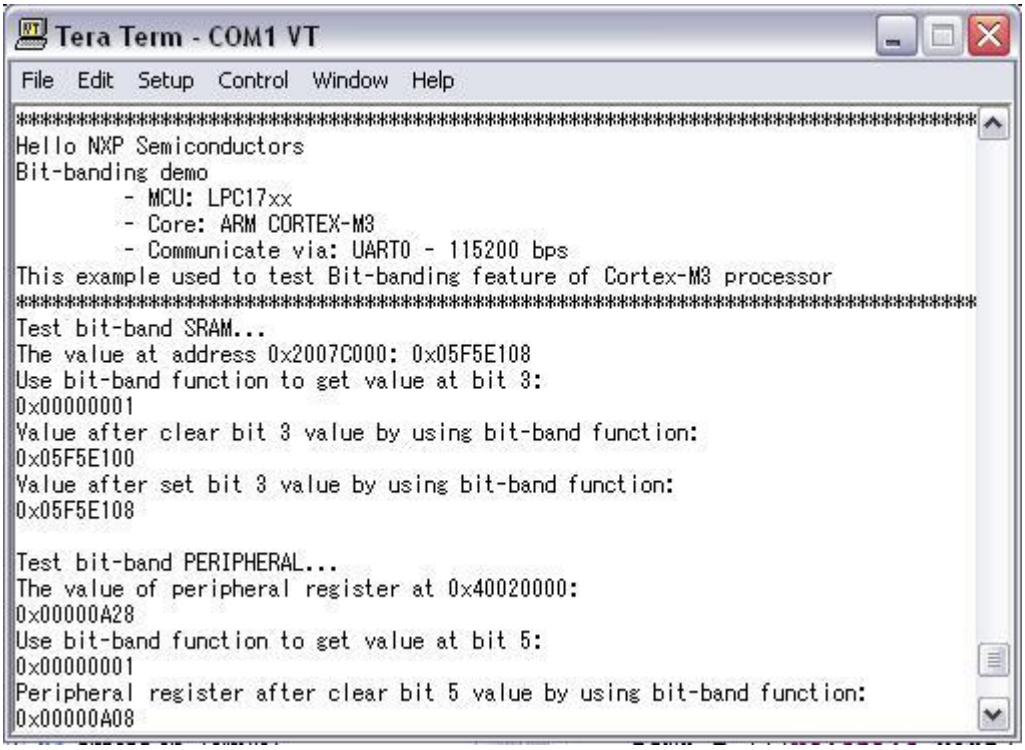

Fig 35. Bit-banding demo

3.3.2 MPU

3.3.2.1 Example description

Purpose

This example describes how to use MPU to protect memory region.

Process

Use MPU to set up 6 memory regions as follows:

- Region 0 - Privileged code: 0x00000000 - 0x0007FFFF(512kB)
- Region 1 - Privileged data: 0x10000000 - 0x10007FFF(32kB)
- Region 2 - APB Peripheral: 0x40000000 - 0x400FFFFF(1MB)
- Region 3 - AHB peripheral: 0x50000000 - 0x501FFFFF(2MB)
- Region 4 - System control: 0xE0000000 - 0xE00FFFFF(1MB)
- Region 5 - GPIO peripheral: 0x2009C000 - 0x2009FFFF(16kB)
- Region 6 - Private SRAM: 0x20080000 - 0x20083FFF(16kB)

Except region 6 can not access, remain regions can access normally.

After setup:

- First, we try to access memory at address: 0x10000000 -> it will allow to access
- Last, we try to access memory at address: 0x20080000 -> it will not allow to access

At the time we access to memory that disallow, Memory Management Handler will be invoked and we blink LED P1.28 to announce.

3.3.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

mpu.c: Main program file

3.3.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- LED: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.3.2.4 Running mode

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

3.3.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example

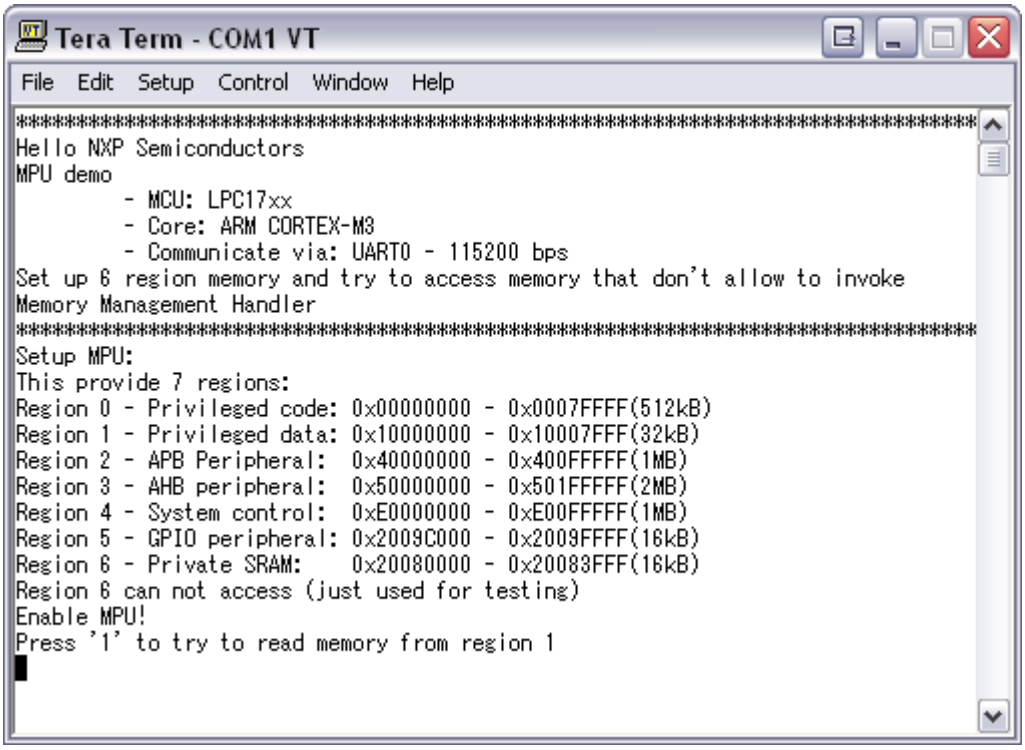

Fig 36. MPU demo

- Press ‘1’

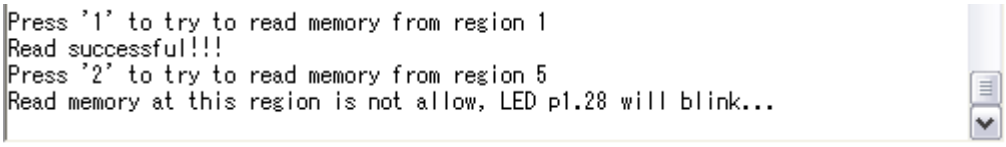

Fig 37. Acess to illegal memory

- Press ‘2’ and see the led blinking

3.3.3 Privilege_mode

3.3.3.1 Example description

Purpose

This example describes how to change privilege to unprivilege mode and vice vesa.

Process

CONTROL[0] register controls the privilege level for software execution when the processor is in Thread mode. So, we can access to get/set CONTROL[0] to change Thread mode privilege level.

Out of reset, Thread mode is privilege.

Read CONTROL[0] to check if it is truly privilege or not. If Thread mode is not privilege, so it's not compliant with Cortex-M3 Technical, the program will enter into infinite error loop.

Then, set CONTROL[0] = 1, changing Thread mode to unprivilege.

Re-check by re-read CONTROL[0] again.

A system service will be call __SCV(). In this exception mode, we can switch back Thread mode into privilege

(Note that:

When Thread mode has been changed from privilege to use, it cannot change ifself back to privilege. Only a handler can change the privilege of Thread mode. Handler mode is always privilege.)

After call handler, get CONTROL[0] to check if it is truly privilege or not.

If not, program will enter to error loop.

We use two LED to display status of Thread mode.

- LED1: turn on if Thread mode is privilege.
- LED2: turn on if Thread mode is unprivilege.
  - ❖ Using MCB1700 board: LED1 is P1.28, LED2 is P1.29
  - ❖ Using IAR1700 board: LED1 is P1.25, LED2 is P0.4

3.3.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

privilege.c: Main program file

3.3.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- LED: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- ACC_IRQ/LED2: 2-3 (LED2)

- Remain jumper: OFF

3.3.3.4 Running mode

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

3.3.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe result on serial display

The screen will be displayed like this:

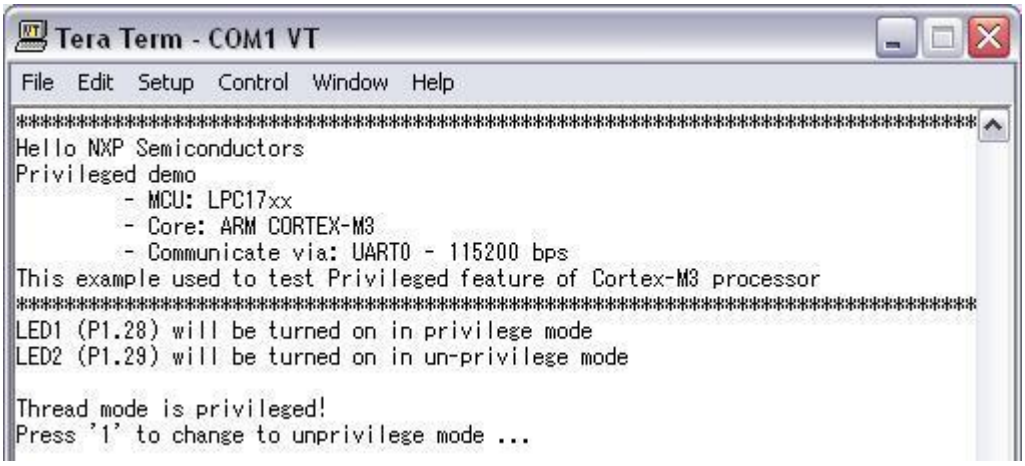

Fig 38. Privilege_mode welcome screen (using MCB1700 board)

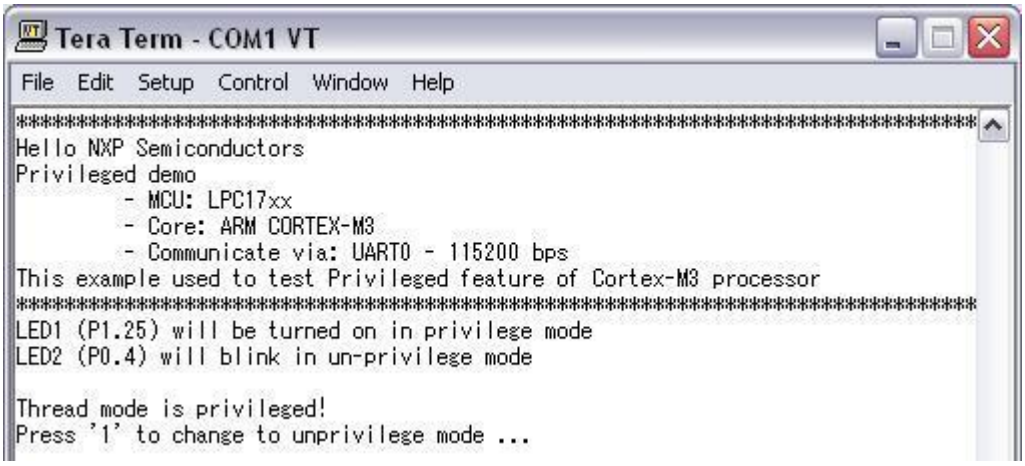

Fig 39. Privilege_mode welcome screen (using IAR1768KS board)

Press ‘1’ to change Thread mode to unprivileged

```
Changed to unprivilege mode!
Check: Thread mode change to unprivilege successful!
Press '2' to change to privilege mode by calling system call exception...
```

Fig 40. Change to unprivileged

Press ‘2’ to change Thread mode to privilege by calling handler.

```
Called system call exception!
Check: Thread mode change to privilege successful!
Demo terminate!
```

Fig 41. Switch back Thread mode to privilege

3.4 DAC

3.4.1 DMA

3.4.1.1 Example description

Purpose

This example describes how to use DMA to transfer data to DAC peripheral

Process

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

Formula for ouput voltage on AOUT is:

AOUT = VALUE x ((Vrefp - Vrefn)/1024)+Vrefn

in which:

- Vrefp: tied to VDD(3.3V)
- Vrefn: tied to Vss

GPDMA channel 0 is configured in this example.

GPDMA channel 0 will tranfer "dac_value" to DAC peripheral. DAC updated values have range from 0 to 0x3FF. So AOUT ouput voltage will change from: Vss to VDD.

Observe AOUT(P0.26) signal by oscilloscope.

3.4.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

dac_dma.c: Main program file

3.4.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.4.1.4 Running mode

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

3.4.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode).

Step 3: Configure hardware and serial display as above instruction.

Step 4: Run example and observe AOUT signal by oscilloscope.

DAC signal displays on oscillopscope like this:

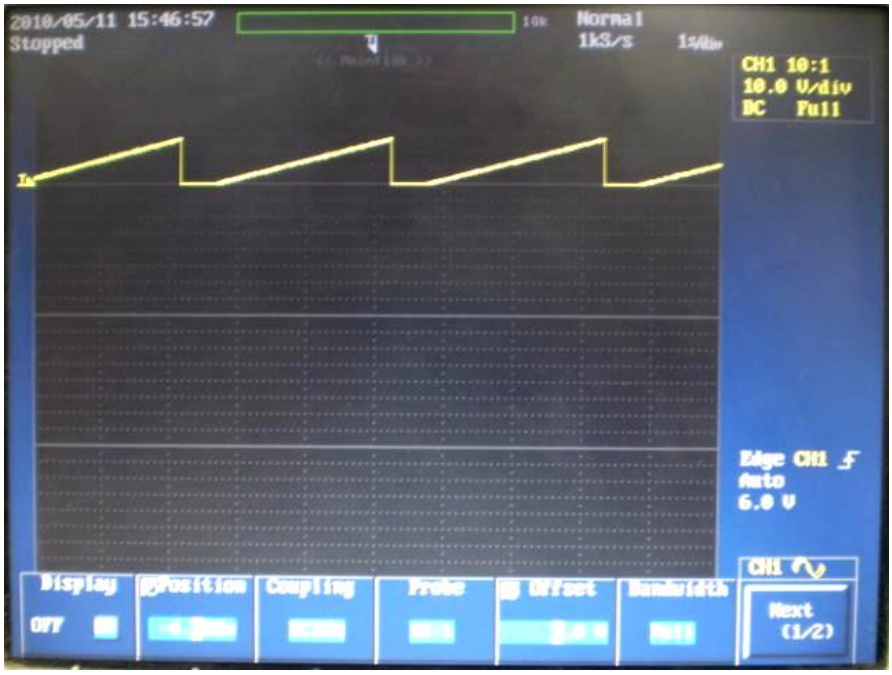

Fig 42. DAC AOUT(P0.26) signal

3.4.2 SineWave

3.4.2.1 Example description

Purpose

This example describes how to use DMA to generate a sinewave signal

Process

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

Formula for ouput voltage on AOUT is:

AOUT = VALUE x ((Vrefp - Vrefn)/1024)+Vrefn

in which:

- Vrefp: tied to VDD(3.3V)

- Vrefn: tied to Vss

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

```
for(i=0;i<NUM_SINE_SAMPLE;i++) //NUM_SINE_SAMPLE = 60
{
  dac_sine_lut[i] = 512 + 512*sin(i);
  dac_sine_lut[i] = (dac_sine_lut[i]<<6);
}
```

GPDMA channel 0 is configured in this example and used to transfer dac_sine_lut[i] to DAC peripheral. When the last item of dac_sine_lut is transfered, 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 sinewave frequency:

time out =(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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

dac_sinewave.c: Main program file

3.4.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

3.4.2.4 Running mode

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

3.4.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware as above instruction

Step 4: Run example and observe AOUT(P0.26) signal by oscilloscope

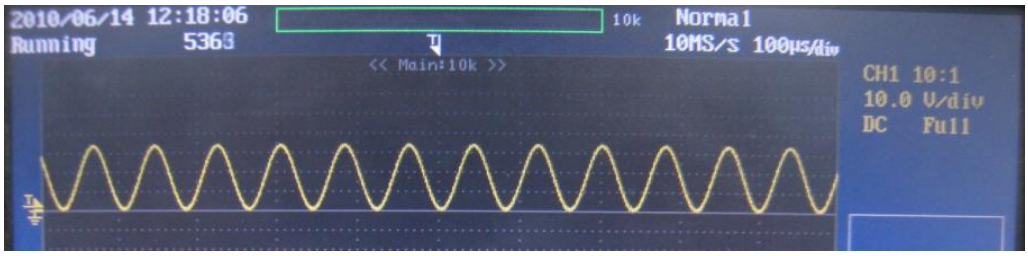

Fig 43. DAC sinewave signal

3.4.3 WaveGenerate

3.4.3.1 Example description

Purpose

This example describes how to use DMA to generate multi signal forms.

Process

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

Formula for ouput voltage on AOUT is:

AOUT = VALUE x ((Vrefp - Vrefn)/1024)+Vrefn

in which:

- Vrefp: tied to VDD(3.3V)
- Vrefn: tied to Vss

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

```
for(i=0;i<NUM_SAMPLE_SINE;i++) //NUM_SAMPLE_SINE = 60
{
  dac_lut[i] = 512 + 512*sin(i);
  dac_lut[i] = (dac_sine_lut[i]<<6);
}
```

For generating triangle wave signal, the following formula is used:

```
for(i=0;i<NUM_SAMPLE;i++) //NUM_SAMPLE=64
{
  if(i<32) dac_lut[i]= 32*i;
  else if (i==32) dac_lut[i]= 1023;
  else dac_lut[i] = 32*(NUM_SAMPLE-i);
  dac_lut[i] = (dac_lut[i]<<6);
}
```

This will create a balance triangle.

Below is the formula for escalator case:

```
for(i=0;i<NUM_SAMPLE;i++) //NUM_SAMPLE=64
{
  dac_lut[i] = (1023/3)*(i/16);
  dac_lut[i] = (dac_lut[i]<<6);
}
```

This will create an escalator with 4 steps.

GPDMA channel 0 is configured in this example and used to transfer dac_lut[i] to DAC peripheral.When the last item of dac_lut is transfered, 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 signal frequency:

time out = (PCLK_DAC_IN_MHZ*1000000)/(SIGNAL_FREQ_IN_HZ*NUM_SAMPLE);

Where:

- PCLK_DAC_IN_MHZ = 25
- SIGNAL_FREQ_IN_HZ = 60
- NUM_SAMPLE = 60 / 64

UART0 will display a menu, asks user to select the signal to generate:

1) Sine

2) Triangle

3) Escalator

Select the signal type and observe AOUT signal by oscilloscope.

Press ESC if you want to terminate trasmitting the currect signal type and generate other signal types.

3.4.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

dac_wave_generate.c: Main program file

3.4.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.4.3.4 Running mode

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

3.4.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware and serial display as above instruction

Step 4: Run example, hear sound from speaker/headphone and observe AOUT signal (P0.26) by oscilloscope.

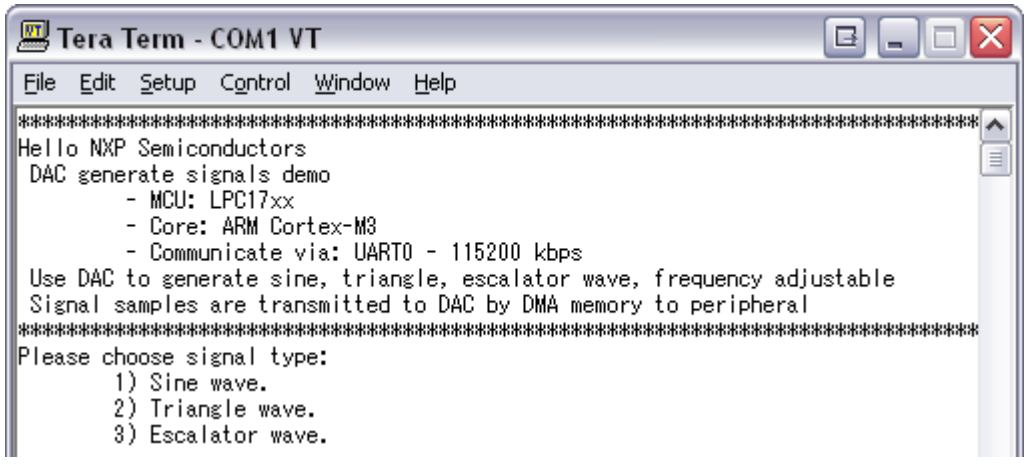

Fig 44. DAC generate wave form demo

Press ‘1’ or ‘2’ or ‘3’ to choose signal type, such as ‘1’ to generate sine wave

Fig 45. DAC generate wave form demo

Press ‘ESC’ to terminal and choose other form.

3.4.4 Speaker

3.4.4.1 Example description

Purpose

This example describes how to test DAC through speaker

Process

DAC will be initialized with maximum current is 700uA.

Formula for ouput voltage on AOUT is:

AOUT = VALUE x ((Vrefp - Vrefn)/1024)+Vrefn

in which:

- Vrefp: tied to VDD(3.3V)
- Vrefn: tied to GND

The DAC peripheral will generate the sin tone, in which will increase step by step to max and reduce to 0 and continue to increase to max again in the same frequence continuously.

We are able to hear the tone from the speaker for many cycles.

3.4.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

dac_speaker.c: Main program file

3.4.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- SPK: ON (speaker)

- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

Listen through headphone port

3.4.4.4 Running mode

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

3.4.4.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware and serial display as above instruction

Step 4: Run example, hear sound from speaker/headphone and observe AOUT signal (P0.26)by oscilloscope.

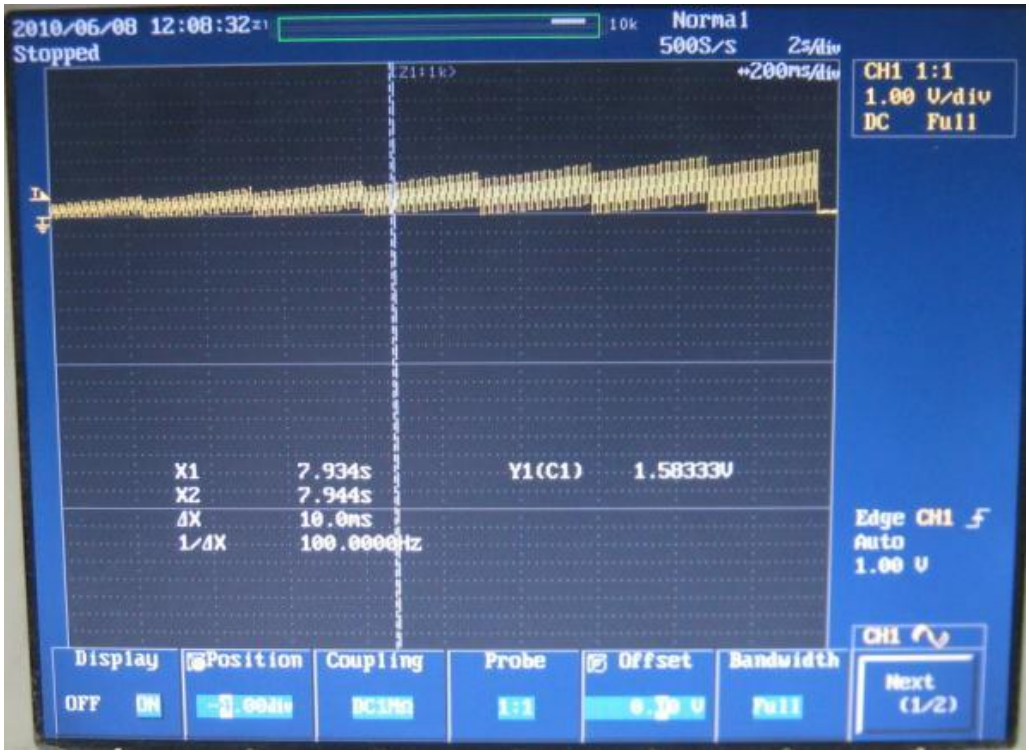

Fig 46. DAC speaker signal on oscilloscope

3.5 EMAC

3.5.1 Easy_Web

3.5.1.1 Example description

Purpose

This example describes how to implement an 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 is: 192.168.0.100

3.5.1.2 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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

Retarget.c: target-dependent low level functions

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

makefile: Example's makefile (to build with GNU toolchain)

webpage.h: webpage html source

3.5.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON

- LED: ON
- E/C: 2-3 (Ethernet)
- E/U: 2-3 (Ethernet)
- Remain jumper: OFF

Use Ethernet Physical Layer Transceiver: DP83848C

MAC address: 1E-30-6C-A2-45-5E

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- ACC_IRQ/LED2: 2-3 (LED2)
- Remain jumper: OFF

Use Ethernet Physical Layer Transceiver: FSZ8721BL

MAC address: 0-FF-FF-FF-FF-FF

3.5.1.4 Running mode

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

3.5.1.5 Step to run

Step 1: Choose correct working board by uncomment correct defined board in lpc17xx_emac.h file

- MCB1700 board, uncomment "#define MCB_LPC_1768"
- IAR-LPC1768-KS board, uncomment "#define MCB_LPC_1768"

  (Should not uncomment both symbols at the same time)

```
//#define MCB_LPC_1768
#define IAR_LPC_1768
```

```
#define MCB_LPC_1768
//#define IAR_LPC_1768
```

Fig 47. Choose working board

Step 2: Build example.

Step 3: Burn hex file into board.

Step 4: Use CrossOver cable to connect from your PC to ETH port on eval board

Step 5: Connect UART0 on this board to COM port on your computer

Step 6: Configure hardware and serial display as above instruction

Step 7: Re-config IP address on PC:

- IP address: 192.168.0.x (x != 100)
- Subnet mask: 255.255.255.0

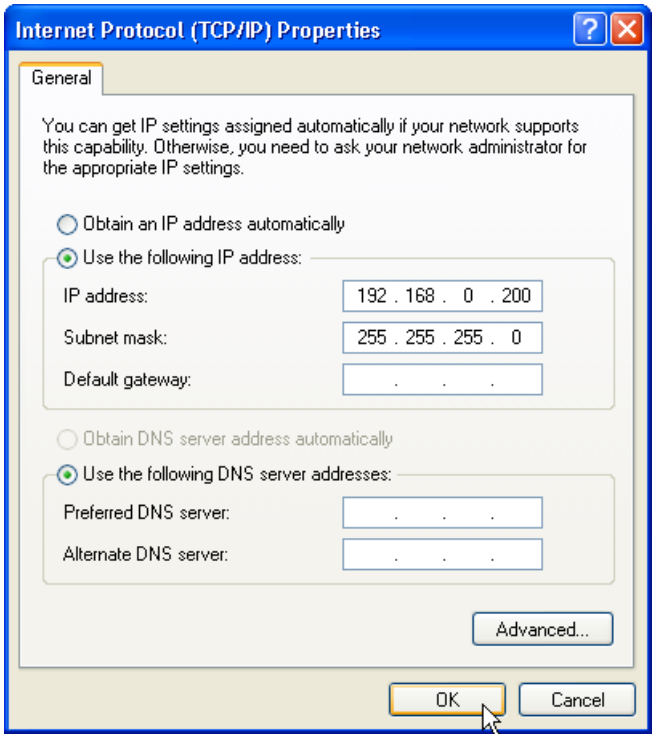

Fig 48. Configure IP address on your PC

Step 8: Reset board, monitor the status via serial display until EMAC initialized

Step 9: Open command prompt window, execute 'ping 192.168.0.100' command

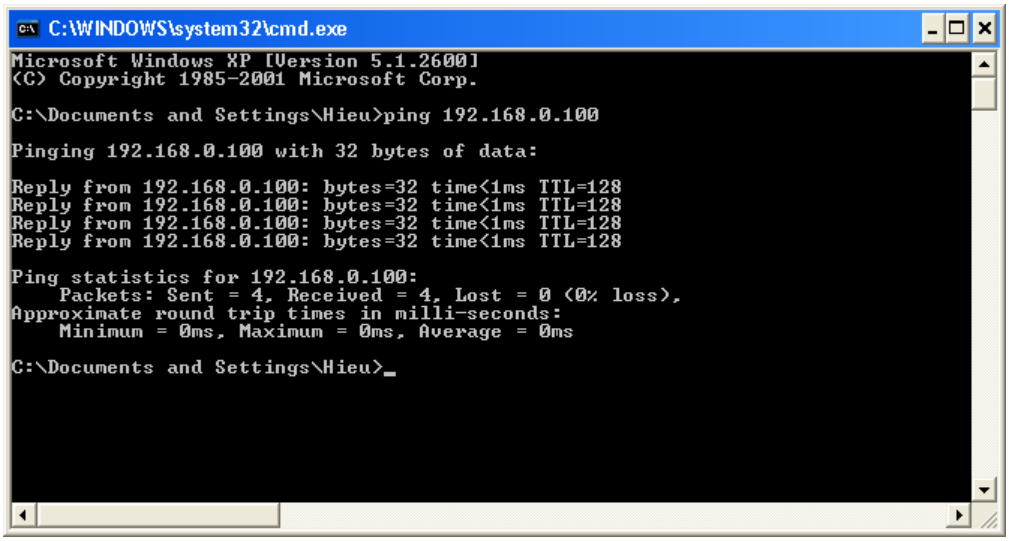

Fig 49. 'ping' command

Step 10: Open web browser, access to address "http://192.168.0.100" to display the content of webserver. Turn potentiometer and see the update ADC value on this web

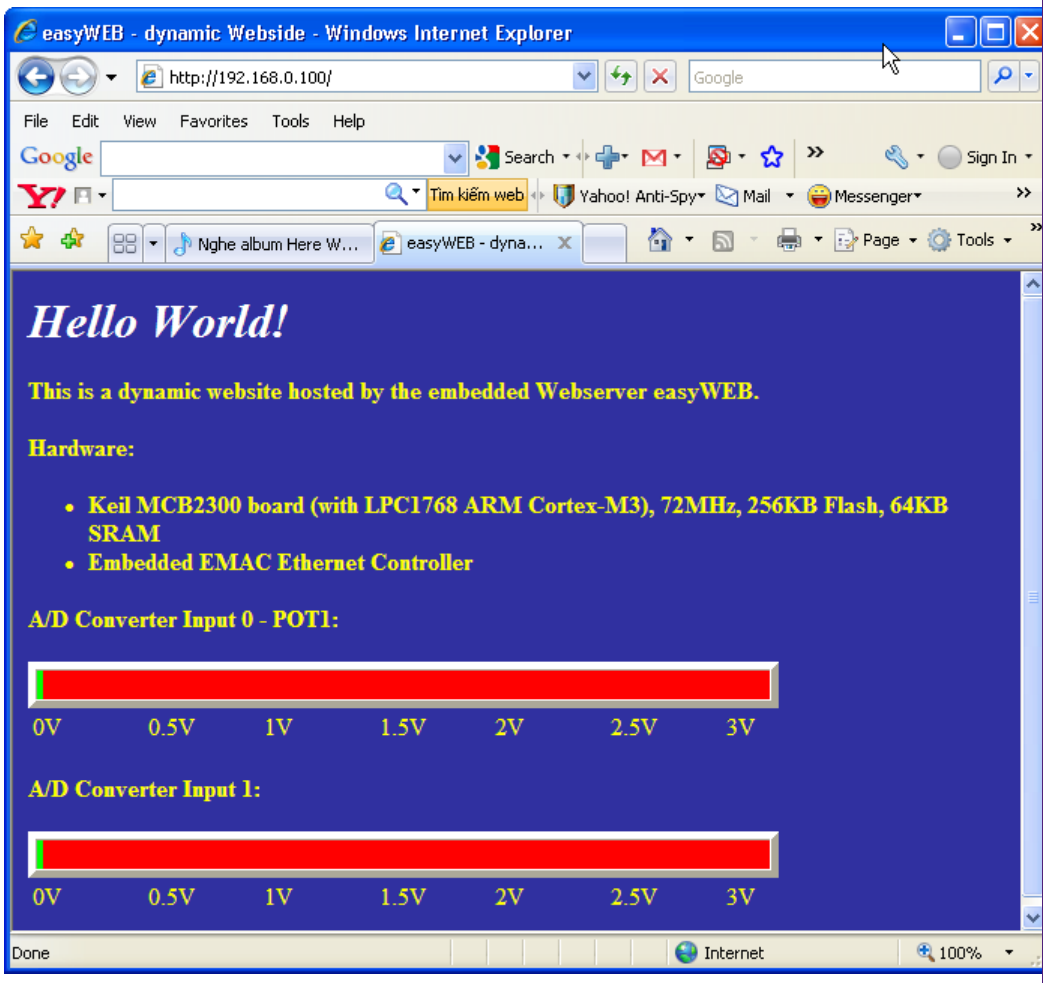

Fig 50. Work with web browser

3.5.2 EmacRaw

3.5.2.1 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 a test to test both TX and RX, use the traffic generator/analyzer, you can creat 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.

3.5.2.2 Directory contents

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

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

crc32c.h/.c: Ethernet CRC module

emactest.c: main program

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

3.5.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- LED: ON
- E/C: 2-3 (Ethernet)
- E/U: 2-3 (Ethernet)
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- ACC_IRQ/LED2: 2-3 (LED2)
- Remain jumper: OFF

3.5.2.4 Running mode

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

3.5.2.5 Step to run

Step 1: Choose correct working board by uncomment correct defined board in lpc17xx_emac.h file

- MCB1700 board, uncomment "#define MCB_LPC_1768"

- IAR-LPC1768-KS board, uncomment "#define MCB_LPC_1768"

  (Should not uncomment both symbols at the same time)

```
//#define MCB_LPC_1768
#define IAR_LPC_1768
```

```
#define MCB_LPC_1768
//#define IAR_LPC_1768
```

Fig 51. Choose working board

Step 2: setting flag (in emactest.c):

- TX_ONLY = 1
- BOUNCE_RX = 0

Build example and burn into first board.

Step 3: setting flag:

- TX_ONLY = 0
- BOUNCE_RX = 1

Build example and burn into second board.

Step 4: Use CrossOver cable to connect two boards.

Step 5: Connect UART0 on two boards to COM ports on your computer.

Step 6: Configure hardware and serial display as above instruction.

Step 7: Hit reset button on two boards.

Step 8: Wait for EMAC initilization completes on two board.

Step 9: If ENABLE_WOL is enabled on board 'BOUNCE_RX' side, after initializing EMAC,it will enter sleep mode to be waked-up on LAN (WoL)

Step 10: On 'TX_ONLY' side, hit INT0 button to send a frame.

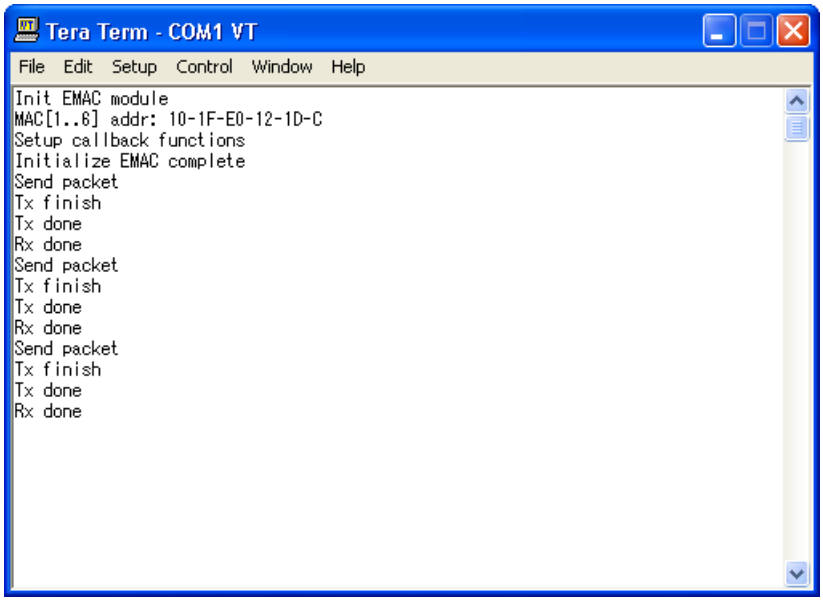

Fig 52. Status on 'TX_ONLY' side

Step 11: After receiving frame, 'BOUNCE_RX' side will be waked-up and operates properly.

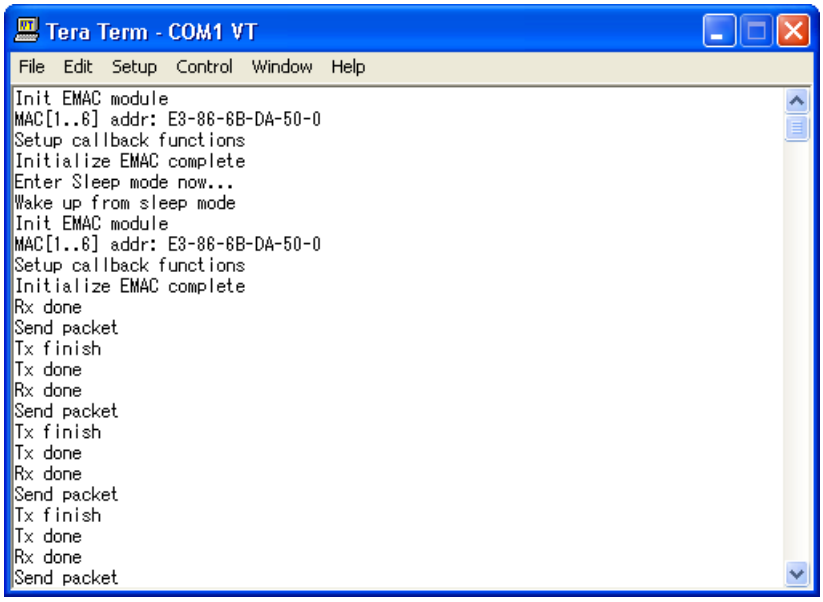

Fig 53. Status on 'BOUNCE_RX' side

3.5.3 uIP

3.5.3.1 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

3.5.3.2 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 DHCPC protocol
- \hello-world: A small example showing how to write applications with protosockets.
- \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

\lpc17xx_port: include main program

makefile: Example's makefile (to build with GNU toolchain)

3.5.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- LED: ON
- E/C: 2-3 (Ethernet)
- E/U: 2-3 (Ethernet)
- Remain jumper: OFF

Use Ethernet Physical Layer Transceiver: DP83848C

MAC address: 1E-30-6C-A2-45-5E

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- ACC_IRQ/LED2: 2-3 (LED2)
- Remain jumper: OFF

Use Ethernet Physical Layer Transceiver: FSZ8721BL

MAC address: 0-FF-FF-FF-FF-FF

3.5.3.4 Running mode

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

3.5.3.5 Step to run

Step 1: Choose correct working board by uncomment correct defined board in lpc17xx_emac.h file

- MCB1700 board, uncomment "#define MCB_LPC_1768"
- IAR-LPC1768-KS board, uncomment "#define MCB_LPC_1768"

  (Should not uncomment both symbols at the same time)

```
//#define MCB_LPC_1768
#define IAR_LPC_1768
```

```
#define MCB_LPC_1768
//#define IAR_LPC_1768
```

Fig 54. Choose working board

Step 2: Build example.

Step 3: Burn hex file into board

Step 4: Use CrossOver cable to connect from your PC to ETH port on eval board

Step 5: Connect UART0 on this board to COM port on your computer

Step 6: Configure hardware and serial display as above instruction

Step 7: Re-config IP address on PC:

- IP address: 192.168.0.x (x != 1, 100)
- Subnet mask: 255.255.255.0

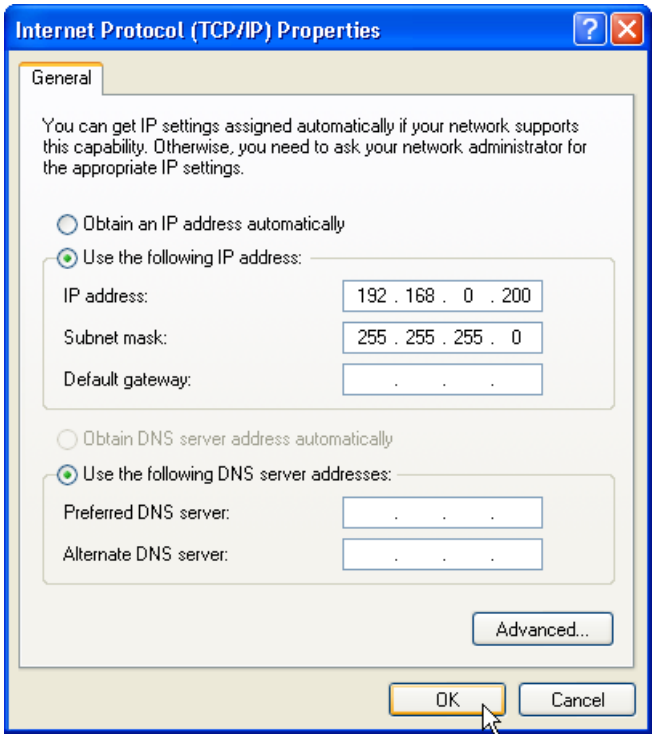

Fig 55. Configure IP address on your PC

Step 8: Reset board, monitor the status via serial display until EMAC initialized

Step 9: Open command prompt window, execute 'ping 192.168.0.100' command

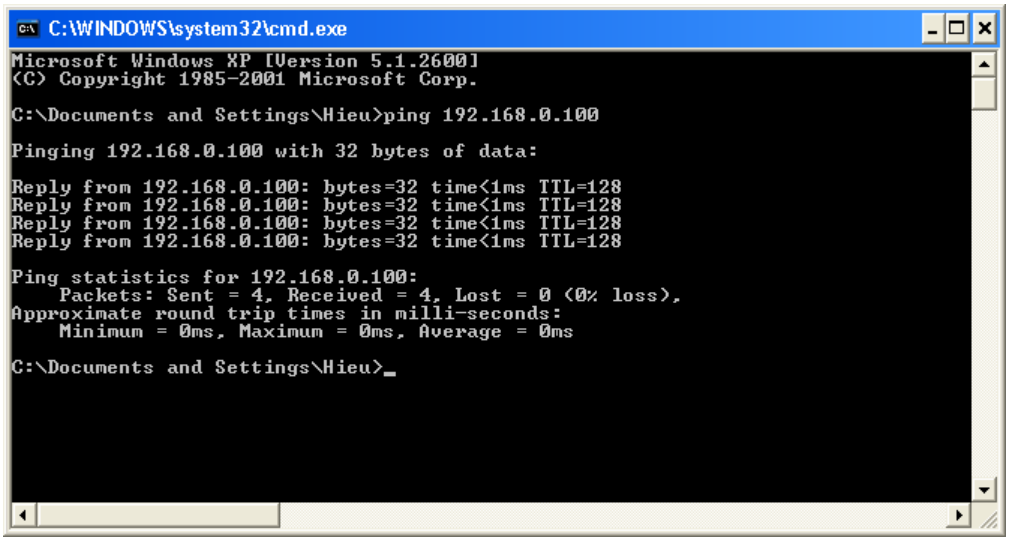

Fig 56. 'ping' command

Step 10: Open web browser, access to address "http://192.168.0.100" to display the content of the webpage.

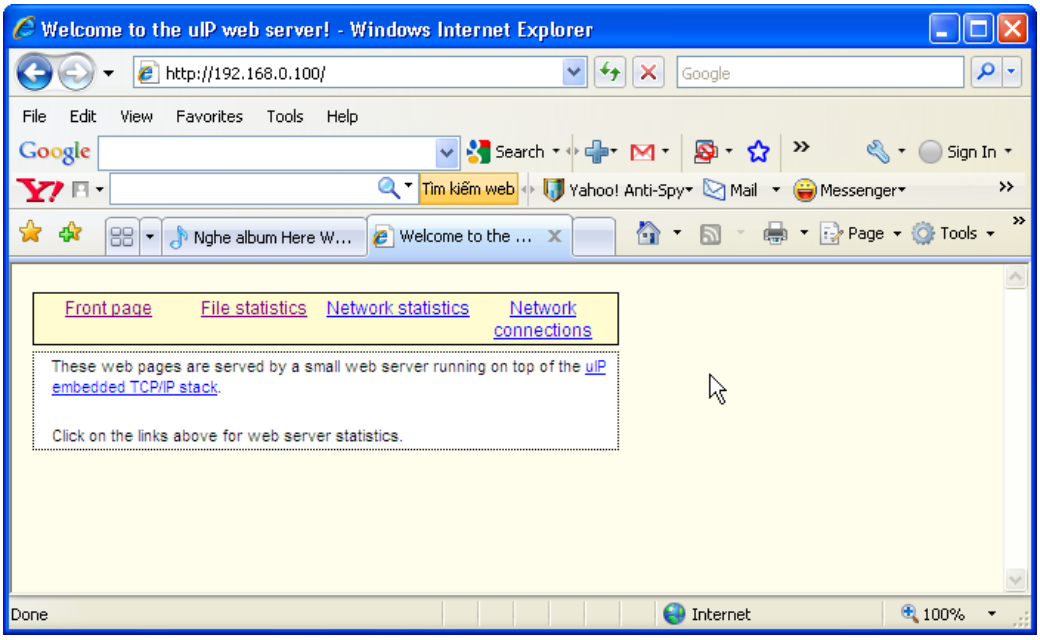

Fig 57. Work with Web browser

3.6 GPDMA

3.6.1 Flash_2_Ram_Test

3.6.1.1 Example description

Purpose

This example describes how to test GPDMA 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 const array 'DMAScr_Buffer'. This buffer will be burned into flash when compile. Received block data is 'DMADest_Buffer' will be stored in ram when compile.

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..

3.6.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

flash_2_ram.c: Main program

3.6.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.6.1.4 Running mode

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

3.6.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe result on serial display

The screen will be displayed like this:

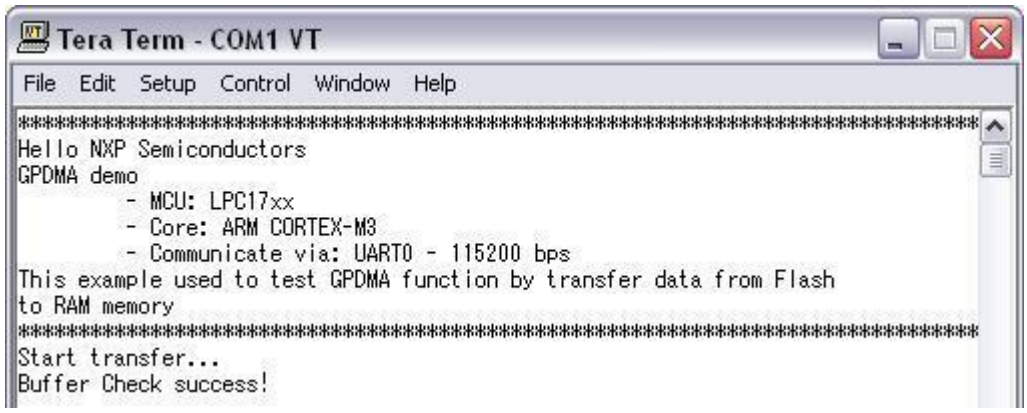

Fig 58. GPDMA Flash-2-Ram demo

3.6.2 GPDMA_Sleep

3.6.2.1 Example description

Purpose

This example describes how to test GPDMA operation in sleep mode

Process

GPDMA perpiheral in LPC17xx can operate in Sleep mode.

Note that: in Sleep mode, the GPDMA can not access the flash memory.

At first, source and destinate buffers were initialized.

After that, GPDMA channel 0 is initialized as follows:

- Channel: 0
- Transfer size: 0x100
- Source address: DMA_SRC = LPC_AHBRAM1_BASE = 0x20080000
- Destination address: DMA_DST = DMA_SRC+DMA_SIZE = 0x20080100
- Transfer width = WORD
- Transfer type: M2M (memory-to-memory)
- No use link list

After enable GPDMA interrupt, call "CLKPWR_Sleep()" function for entering system in sleep mode.

GPDMA will transfer 2 block of data from memory boundary to the other memory boundary on RAM by using interrupt mode in sleep mode.

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 observe DMA transfer process.

3.6.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

gpdma_sleep.c: Main program

3.6.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.6.2.4 Running mode

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

3.6.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe result on serial display

The screen will display like this:

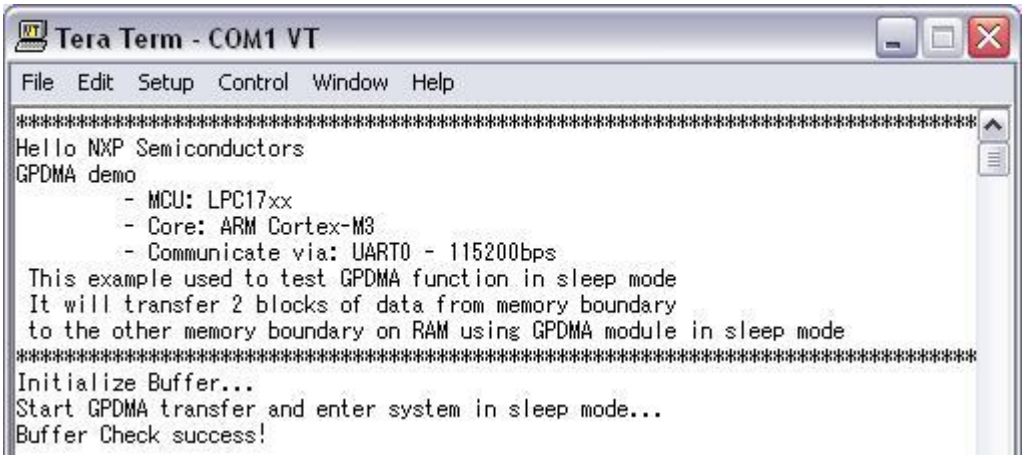

Fig 59. GPDMA sleep demo

3.6.3 Link_list

3.6.3.1 Example description

Purpose

This example describes how to use GPDMA Link-list function

Process

To test GPDMA link-list function, we initialize two source buffers at two different memory blocks (With project run in IAR enviroment, we load two buffers in .data_init section)

Set up two link-list DMA_LLI_Struct[0], DMA_LLI_Struct[1] structs and configure GPDMA peripheral to transfer data of two block to destination buffer.

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

Open serial display to see DMA transfer result.

3.6.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

link_list.c: Main program

3.6.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.6.3.4 Running mode

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

3.6.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe result on serial display

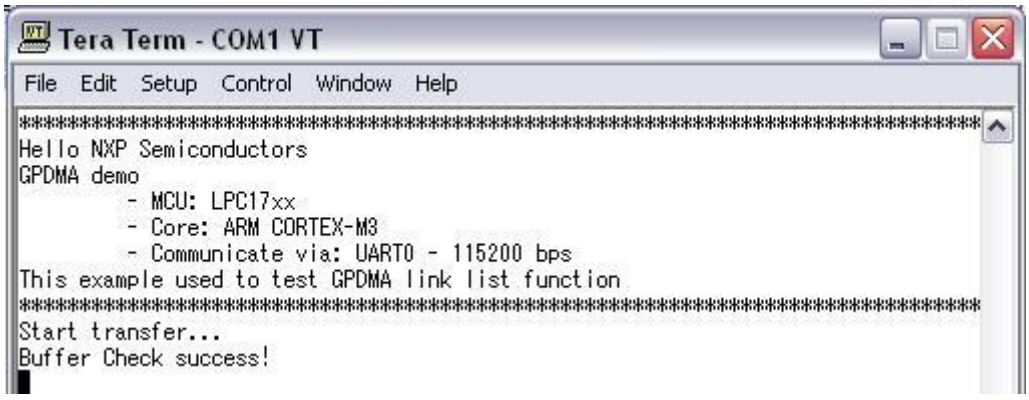

Fig 60. GPDMA Link list demo

3.6.4 Ram_2_Ram_Test

3.6.4.1 Example description

Purpose

This example describes how to configure GPDMA to transfer data from internal RAM memory to internal RAM memory, using interrupt mode

Process

This example will transfer 2 block of data from memory boundary (AHBRAM1_BASE - USB RAM) to the other memory boundary on RAM using GPDMA module with interrupt.

GPDMA channel 0 is configured in this example.

Transfer size is 0x100 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 observe DMA transfer result.

3.6.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

gpdma_m2m_test.c: Main program

3.6.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.6.4.4 Running mode

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

3.6.4.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe result on serial display

The screen will be displayed like this:

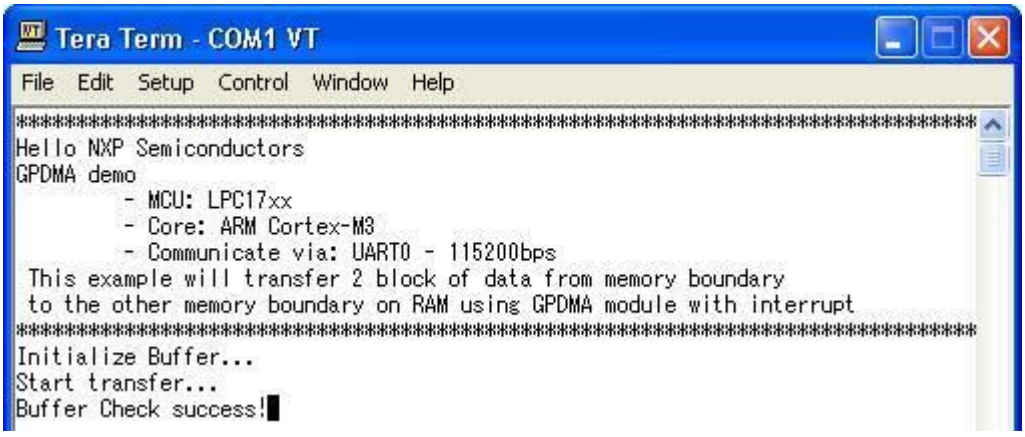

Fig 61. GPDMA Ram-to-Ram demo

3.7 GPIO

3.7.1 GPIO_Interrupt

3.7.1.1 Example description

Purpose

This example describes how to use GPIO interrupt function..

Process

Different board have different layout, so we need to have different configuration

- MCB1700 board: Interrupt is configured to use P0.25 pin (ACD potentiometter)
- IAR-LPC1768-KS board: External interrupt is configured to use P0.23 pin (BUT1 button)

Turn the ADC potentiometer according to clock direction until interrupt happen (If using MCB1700 board).

Hit the BUT1 button (If using IAR-LPC1768-KS board)to make interrupt happen.

3.7.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

gpio_int.c: Main program

3.7.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- LED: ON
- INT0: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- ACC_IRQ/LED2: 2-3 (LED2)
- Remain jumper: OFF

3.7.1.4 Running mode

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

3.7.1.5 Step to run

Step 1: Choose correct working board by uncomment correct defined board in gpio_int.c file

- MCB1700 board, uncomment "#define MCB_LPC_1768"
- IAR-LPC1768-KS board, uncomment "#define MCB_LPC_1768"

  (Should not uncomment both symbols at the same time)

```
//#define MCB_LPC_1768
#define IAR_LPC_1768
```

```
#define MCB_LPC_1768
//#define IAR_LPC_1768
```

Fig 62. Choose working board

Step 2: Build example.

Step 3: Burn hex file into board (if run in ROM mode)

Step 4: Reset the board

Step 5:

- MCB1700: Turn ACD potentiometter until interrupt occurs, then we see LED P1.29 blinking 8 times.
- IAR-LPC1768-KS board: Press BUT1 button then LED2 will be blinking 8 times

3.7.2 LEDBlinky

3.7.2.1 Example description

Purpose

A simple program to test GPIO interrupt functionality to drive LED

Process

- Keil MCB1700 with LPC1768: After reset software all LED from P1.28 to P2.6 will be blinking one after another
- IAR LPC1768 KickStart: After reset software LED1 will be blinking

3.7.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

LEDBlinky.c: Main program

3.7.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON

- VDDREG: ON
- VBUS: ON
- LED: ON
- INT0: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.7.2.4 Running mode

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

3.7.2.5 Step to run

Step 1: Choose correct working board by uncomment correct defined board in LEDBlinky.c file

- MCB1700 board, uncomment "#define MCB_LPC_1768"
- IAR-LPC1768-KS board, uncomment "#define MCB_LPC_1768"

  (Should not uncomment both symbols at the same time)

```
//#define MCB_LPC_1768
#define IAR_LPC_1768
```

```
#define MCB_LPC_1768
//#define IAR_LPC_1768
```

Fig 63. Choose working board

Step 2: Build example.

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Reset the board then we can see the LED blinking

3.8 I2C

3.8.1 Master

3.8.1.1 Example description

Purpose

This example describes how to use I2S as master device to communicate with I2S slave device in polling mode

Note that: this example will run combine with slave example at folder \slave.

(Pls see "Step to run" part in this file for more information)

Process

This example uses I2C as master device to transfer data from/to I2C slave device

- First, the master transmit to slave a number of data bytes
- Then, the master receive a number of data bytes from slave.
- Finally, the master send two bytes to slave, send repeat start immediately and receive from slave a number of data byte.

Using in polling mode.

I2C Clock Rate is set at 100K.

UART0 is configure for display message and control I2C.

After reset UART0 will send the welcome message and guide user how to control the I2C transmition and reception via UART

3.8.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

master.c: Main program

3.8.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

I2C connection

- If using I2C0:
  - SDA -> P0.27
  - SCL -> P0.28
- If using I2C2
  - SDA -> P0.10
  - SCL -> P0.11

SDA pin on master connects with SDA pin on slave

SCL pin on master connects with SCL pin on slave

3.8.1.4 Running mode

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

3.8.1.5 Step to run

Step 1: Choose what I2C peripheral use in this case by setting "USEDI2CDEV_M"

- If using I2C0, setting

  #define USEDI2CDEV_M 0

- If using I2C2, setting

  #define USEDI2CDEV_M 2

Step 2: Build example.

Step 3: Burn hex file into master board (if run on ROM mode)

Step 4: Choose correct working board and I2C peripheral in \slave\slave.c file

Build and burn this example into slave board (if run on ROM mode)

Step 5: Connect UART0 on master and slave boards to COM ports on your computer

Step 6: Configure hardware and serial display as above instruction

Step 7: Run example

- Hit reset button on slave board
- At slave side: Press '1' to start
- Hit reset button on master board
- At master side:
  - Press '1' to transmit
  - Press '2' to receive
  - Press '3' to transmit, then repeat start and receive

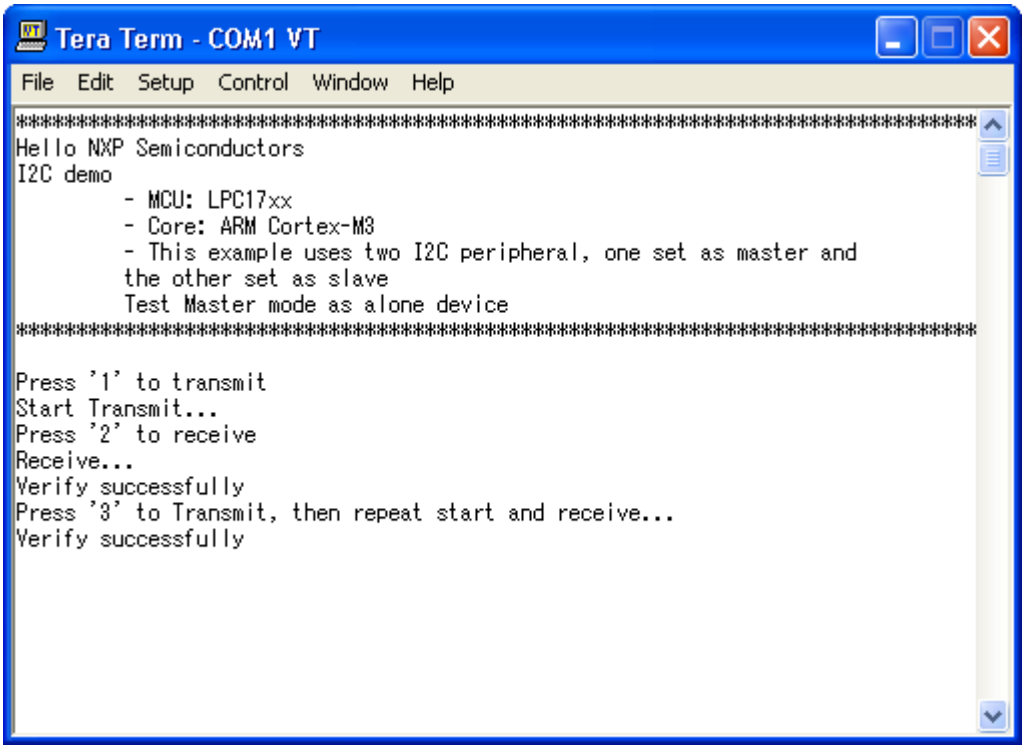

Fig 64. I2C Master demo

(Please refrerence “I2C > slave” for more information about slave operation)

3.8.2 Master_Slave_Interrupt

3.8.2.1 Example description

Purpose

This example describes how uses I2C as master device to transfer data from/to I2C slave device

Process

I2C0 is configured as slave and I2C2 is configured as master.

Both I2C using polling mode.

I2C Clock Rate is set at 100K.

UART0 is configure for display transmit process.

After reset software will run the following steps:

- First, the master transmit to slave a number of data bytes
- Then, the master receive a number of data bytes from slave.
- Finally, the master send two bytes to slave, send repeat start immediately and receive from slave a number of data byte

3.8.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

i2c_master_slave_int_test.c: Main program

3.8.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

I2C connection

- For I2C0:
  - SDA -> P0.27
  - SCL -> P0.28
- For I2C2:
  - SDA -> P0.10
  - SCL -> P0.11

SDA pin of I2C master connects to SDA pin of I2C slave peripheral

SCL pin of I2C master connects to SCL pin of I2C slave peripheral

3.8.2.4 Running mode

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

3.8.2.5 Step to run

Step 1: Choose I2C peripheral for master and slave by setting 'USEDI2CDEV_M' and 'USEDI2CDEV_S'

Step 2: Build example.

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect UART0 on this board to COM port on your computer

Step 5: Configure hardware and serial display as above instruction

Step 6: Run example, and see the result in UART display tool.

The screen will be like this:

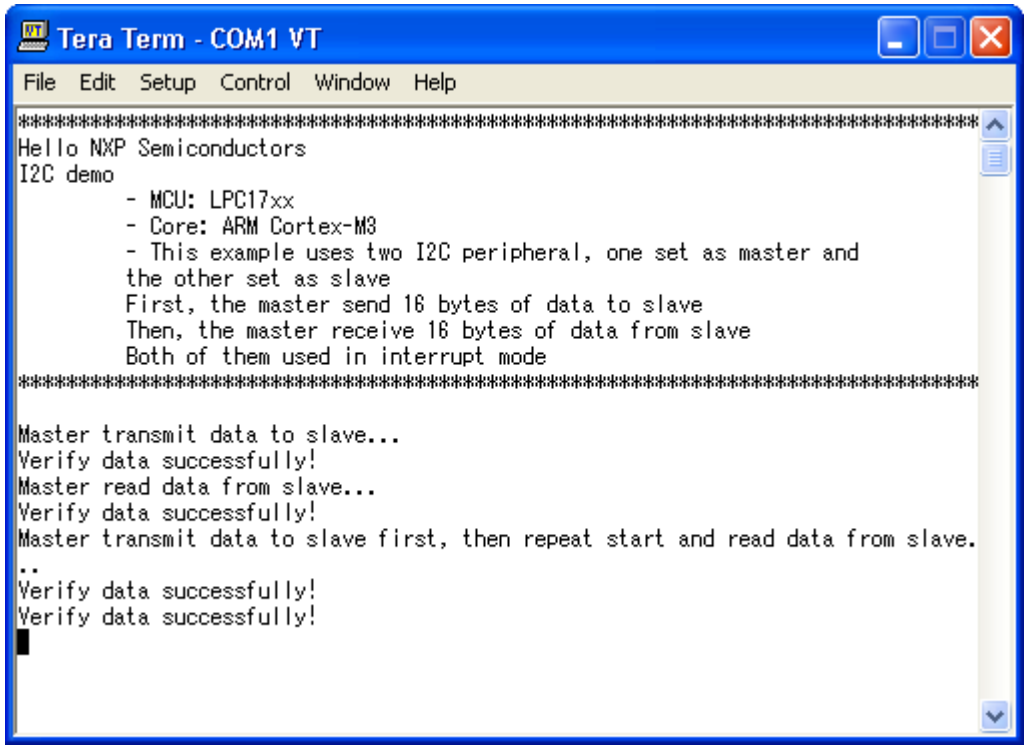

Fig 65. I2C Master-Slave interrupt demo

3.8.3 Monitor

3.8.3.1 Example description

Purpose

This example describes how to use I2C to monitor traffic on I2C bus

Process

I2C0 is configured in monitor mode.

UART0 is configure for displaying captured data.

1. After reset software will run the following steps:
2. Display introducion menu.
3. Ask user input the length of monitor buffer, in hex.
4. Start capturing I2C data, when buffer length exceeded, the whole captured data is displayed.
5. Roll back to step 2.

3.8.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

i2c_monitor.c: Main program

3.8.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

I2C connection

- For I2C0:
    - SDA -> P0.27
    - SCL -> P0.28

SDA, SCL connect to SDA, SCL of the I2C bus which we intend to capture data.

Must be carefull that SDA, SCL signals on demo board are 3.3V pulled up resistor.

In this example, we connect SDA, SCL signals to SDA, SCL of the other MCB1700 (or IAR LPC1768) board running Master_Slave_Interrupt example.

3.8.3.4 Running mode

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

3.8.3.5 Step to run

Step 1: Define the maximum monitor buffer size 'BUFFER_SIZE'

Step 2: Build example.

Step 3: Burn hex file into board (if run on ROM mode)

Burn hex file of Master_Slave_Interrupt example into the other board.

Step 4: Connect UART0 on this board to COM port on your computer

Step 5: Configure hardware and serial display as above instruction

Step 6: Run example

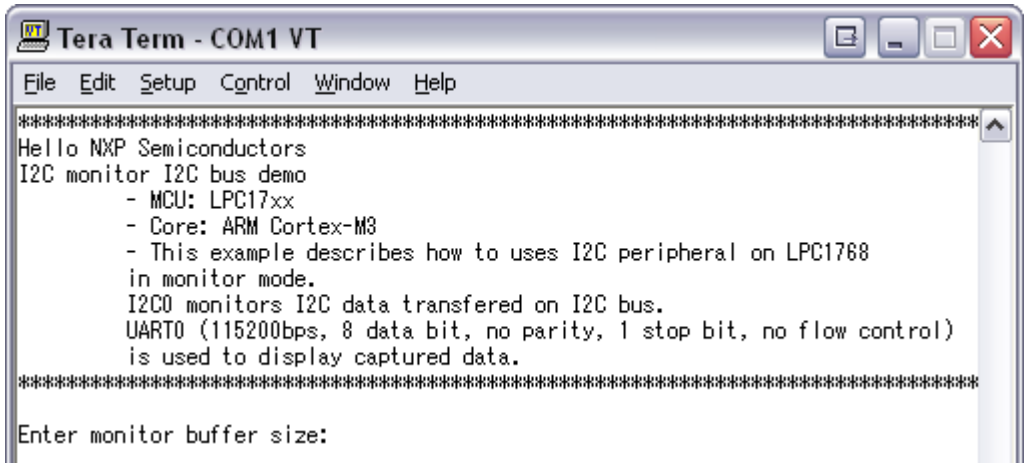

Fig 66. I2C Monitor demo

Input a buffer size (smaller than BUFFER_SIZE), such as: 0x10 (type: 10)

Note that: Input is an hex number. Maximum number is 0x80 bytes.

Fig 67. Enter monitor buffer size

Press RESET button on the board running Master_Slave_Interrupt example

Look at the PC's terminal screen to see the captured data.

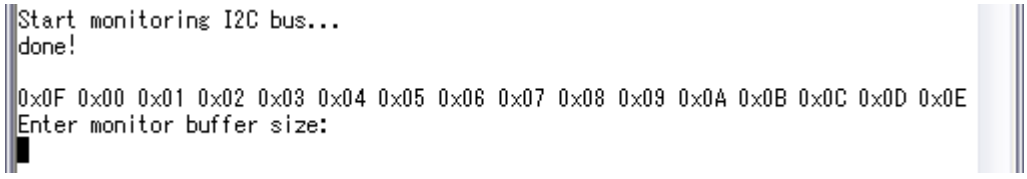

Fig 68. Captured data

3.8.4 pca8581_polling

3.8.4.1 Example description

Purpose

This example describes how to use I2C to communicate with EEPROM PCA8581

Process

I2C is configured as master using polling mode to send/receive data.

I2C Clock Rate is set at 200K.

PCA8581 Slave address is 0xA0 (8-bit format)

After initialize and enable I2C, I2C writes 8 data to PCA8581 at address: (0x01<<3) by using 'PCA8581_Write( )'function and then read back again these data from PCA8581.

After transmittion finished, read and write buffer will be compared, if not familiar, the program will display an error message.

(Please see 'PCA8581.pdf' for more information about EEPROM PCA8581)

3.8.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

pca8581_polling.c: Main program

PCA8581.pdf: pca8581's datasheet file

3.8.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

EEPROM PCA8581 connection

- A0, A1 and A2 - Pin 1, 2, 3: must be connect with GND
- SCL - Pin 6: connects with SCL pin of I2C peripheral on board
- SDA - Pin 5: connects with SDA pin of I2C peripheral on board
- VDD - Pin 8: supply voltage, connects to 5V power source
- Vss - Pin 4: connect to GND on board
- TEST - Pin 7: not used

I2C pin selection:

- For I2C0:
    - SDA -> P0.27

- SCL -> P0.28

- For I2C2:
  - SDA -> P0.10
  - SCL -> P0.11

3.8.4.4 Running mode

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

3.8.4.5 Step to run

Step 1: Choose I2C peripheral by setting 'USEDI2CDEV'

Step 2: Build example.

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect PCA8581 and configure serial display as above instruction

Step 5: Connect UART0 on this board to COM port on your computer

Step 6: Run example, and control the I2C transmition result on UART display tool.

The screen will be displayed like this:

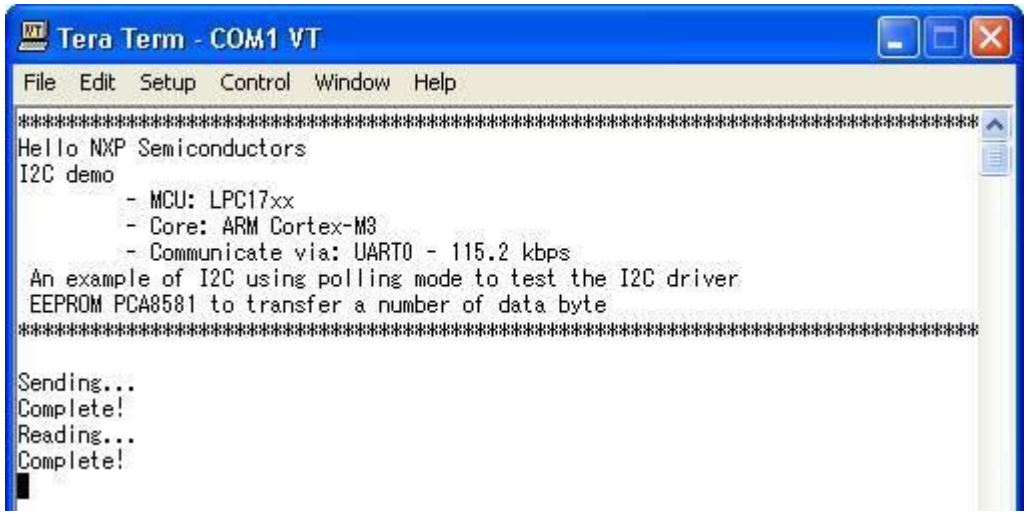

Fig 69. I2C pca8581 polling test demo

3.8.5 sc16is750_int

3.8.5.1 Example description

Purpose

This example describes how to use I2C to communicate with SC16IS750/760 Demo Board in interrupt mode.

Process

I2C is configured as master using interrupt mode.

I2C Clock Rate is set at 100K.

First, I2C send commands to reset, config direction, and validate value on SC16IS740 chip in interrupt mode.

Then, start to use I2C polling mode to handle SC16IS740 board.

On serial display:

- Press 'r' to print menu
- Press '1': send 0x00 value to IOStat register to turn on 8 LEDs on SC16IS740 board.
- Press '2': send 0xFF value to IOStat register to turn off 8 LEDs on SC16IS740 board.

3.8.5.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

i2c_interrupt_test.c: Main program

3.8.5.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SC16IS750 Demo board

- JP2: 1-2 (I2C)
- JP16: 2-3 (hard reset)
- Remain jumper: OFF

I2C connection

- SDA and SCL pin in two board connect in "pin-to-pin" style:
    - SDA <-> SDA
    - SCL <-> SCK
- Common power source 3.3V and ground must be connected together between two board.

I2C pin selection

- For I2C0:
  - SDA -> P0.27
  - SCL -> P0.28
- For I2C2:
  - SDA -> P0.10
  - SCL -> P0.11

3.8.5.4 Running mode

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

3.8.5.5 Step to run

Step 1: Choose I2C peripheral by setting 'USEDI2CDEV'

Step 2: Build example.

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect SC16IS750 with eval board and configure serial display as above instruction

Step 5: Connect UART0 on this board to COM port on your computer

Step 6: Run example.

- Press 'r' to display welcome screen
- Press '1' to turn on LEDs
- Press '2' to turn off LEDs

The screen will be displayed like this:

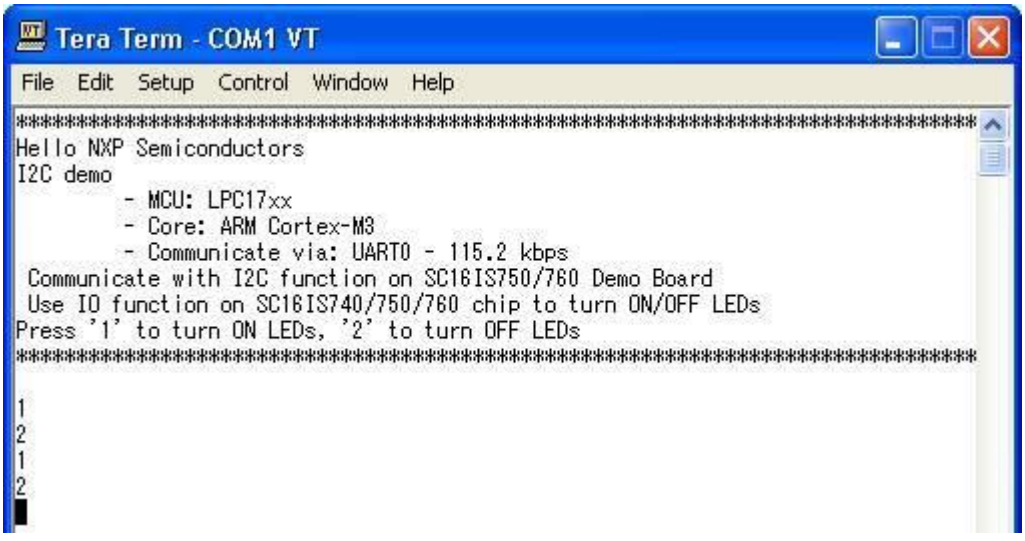

Fig 70. I2C SC16IS750 interrupt demo

3.8.6 sc16is750_polling

3.8.6.1 Example description

Purpose

This example describes how to use I2C to communicate with SC16IS750/760 Demo Board in polling mode.

Process

I2C is configured as master using polling mode.

I2C Clock Rate is set at 100K.

I2C send commands to reset, config direction, validate value and handle SC16IS740 chip using polling mode.

On serial display:

- Press 'r' to print menu
- Press '1': send 0x00 value to IOStat register to turn on 8 LEDs on SC16IS740 board.
- Press '2': send 0xFF value to IOStat register to turn off 8 LEDs on SC16IS740 board.

3.8.6.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

i2c_polling_test.c: Main program

3.8.6.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SC16IS750 Demo board

- JP2: 1-2 (I2C)
- JP16: 2-3 (hard reset)
- Remain jumper: OFF

I2C connection

- SDA and SCL pin in two board connect in "pin-to-pin" style:
  - SDA <-> SDA
  - SCL <-> SCKss
- Common power source 3.3V and ground must be connected together between two board.

I2C pin selection

- For I2C0:
  - SDA -> P0.27
  - SCL -> P0.28
- For I2C2:
  - SDA -> P0.10
  - SCL -> P0.11

3.8.6.4 Running mode

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

3.8.6.5 Step to run

Step 1: Choose I2C peripheral by setting 'USEDI2CDEV'

Step 2: Build example.

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect SC16IS750 with eval board and configure serial display as above instruction

Step 5: Connect UART0 on this board to COM port on your computer

Step 6: Run example.

- Press 'r' to display welcome screen
- Press '1' to turn on LEDs
- Press '2' to turn off LEDs

The screen will be displayed like this:

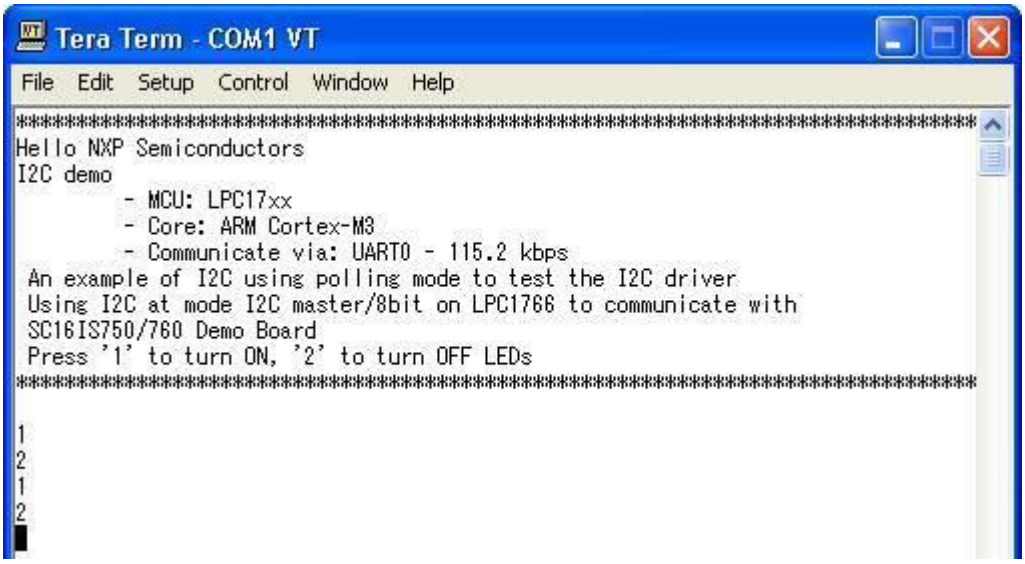

Fig 71. I2C SC16IS750 polling demo

3.8.7 Slave

3.8.7.1 Example description

Purpose

This example describes how to use I2S as slave device to communicate with I2S master device in polling mode

Note that: this example will run combine with slave example at folder \slave.

Process

This example uses I2C as master device to transfer data from/to I2C slave device

- First, the slave receive from master a number of data bytes
- Then, the slave transmit a number of data bytes to master.
- Finally, the master send two bytes to slave, send repeat start immediately and receive from slave a number of data byte.

Using in polling mode.

UART0 is configure for display message and control I2C.

After reset UART0 will send the welcome message and guide user how to control the I2C transmition and reception via UAR

3.8.7.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

slave.c: Main program

3.8.7.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

I2C connection

- For I2C0:
  - SDA -> P0.27
  - SCL -> P0.28
- For I2C2:
  - SDA -> P0.10
  - SCL -> P0.11

SDA pin on master connects with SDA pin on slave

SCL pin on master connects with SCL pin on slave

3.8.7.4 Running mode

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

3.8.7.5 Step to run

Step 1: Choose what I2C peripheral use in this case by setting "USEDI2CDEV_M"

- If using I2C0, setting

  #define USEDI2CDEV_M 0

- If using I2C2, setting

  #define USEDI2CDEV_M 2

Step 2: Build example.

Step 3: Burn hex file into slave board (if run on ROM mode)

Step 4: Choose correct working board and I2C peripheral in \master\master.c file

Build and burn this example into master board (if run on ROM mode)

Step 5: Connect UART0 on master and slave boards to COM ports on your computer

Step 6: Configure hardware and serial display as above instruction

Step 7: Run example

- Hit reset button on slave board
- At slave side: Press '1' to start
- Hit reset button on master board
- At master side:
    - Press '1' to transmit
    - Press '2' to receive
    - Press '3' to transmit, then repeat start and receive

The screen will be like this on slave side:

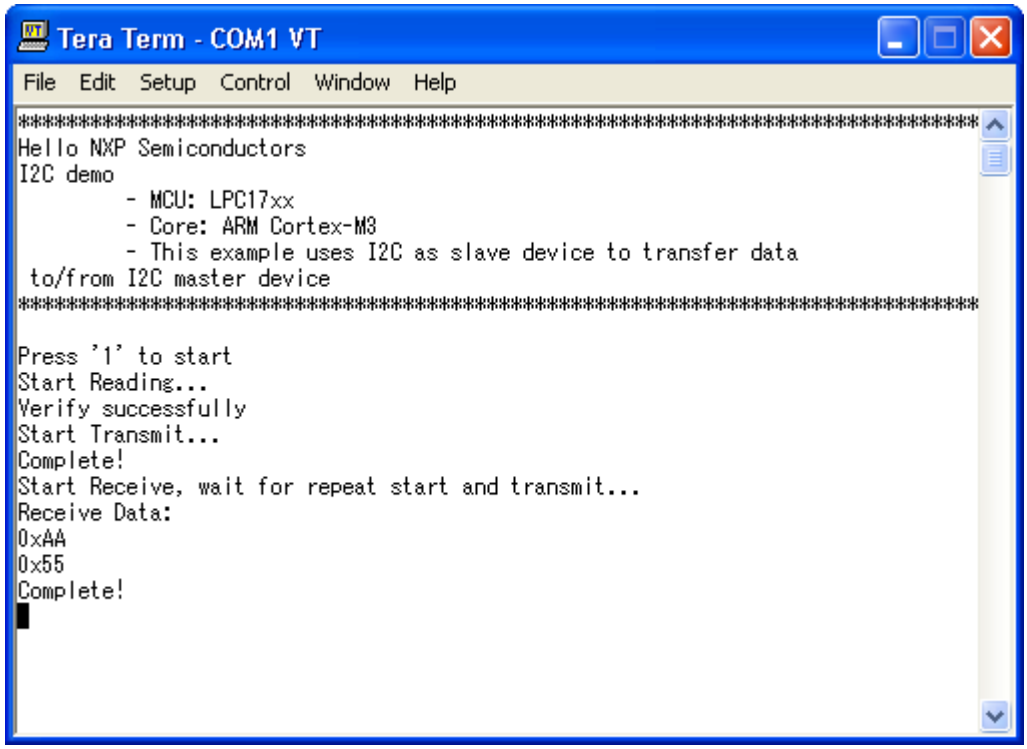

Fig 72. I2C slave demo

3.9 I2S

3.9.1 I2S_DMA

3.9.1.1 Example description

Purpose

This example describes how to use I2S in DMA mode

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)

GPDMA channel 0 and 1 are configured in this example.

DMA channel 0 used to transfer data from interal 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 recieve 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 transmittion finised, "Buffer_Verify(void)" 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.

Pls 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.

3.9.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

i2s_dma_test.c: Main program

3.9.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

I2S connection

- 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

3.9.1.4 Running mode

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

3.9.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe data on serial display

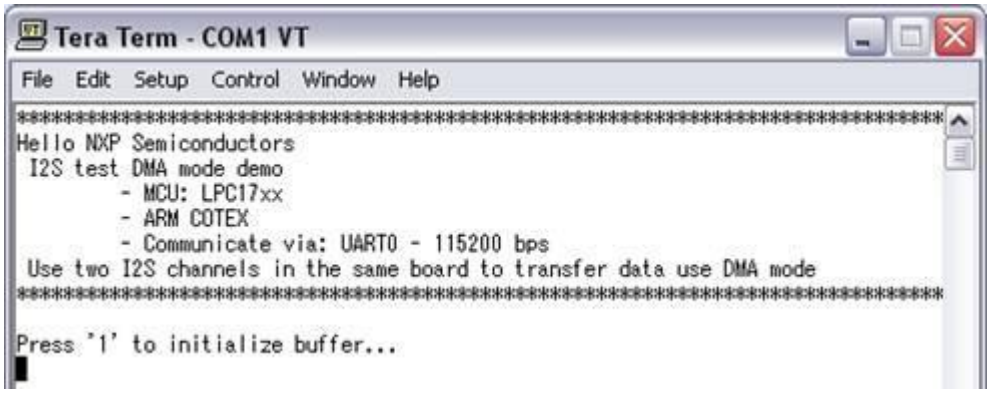

Fig 73. I2S welcome screen in DMA mode

- Press '1' to initialize buffer

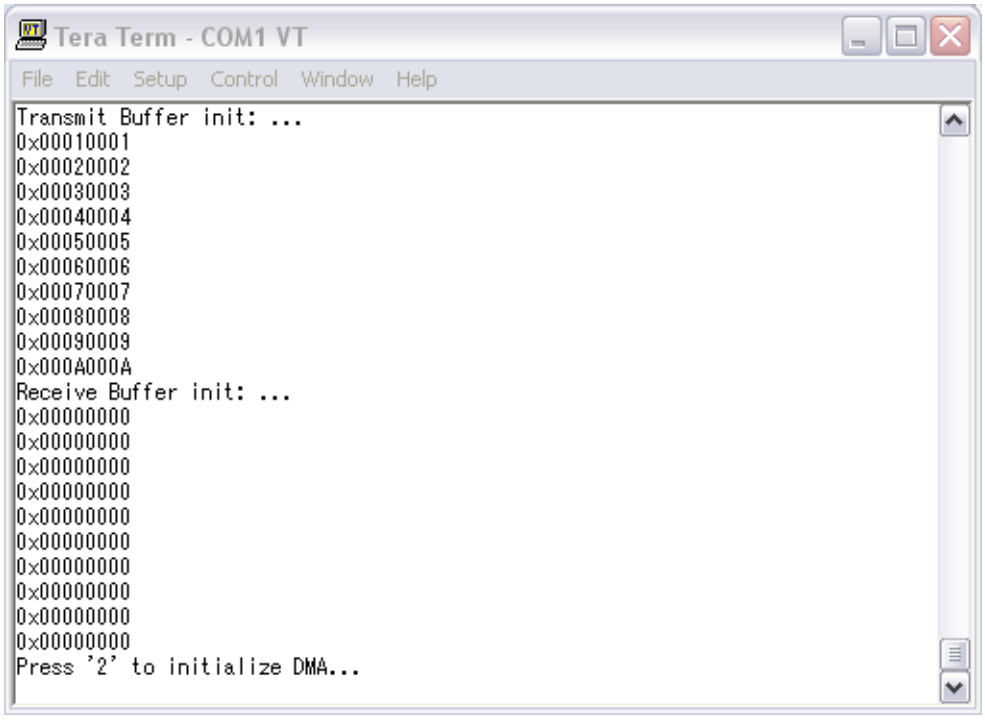

Fig 74. I2S Transmit and Receive Buffer initialized after press '1'

- Press '2' to initialize DMA

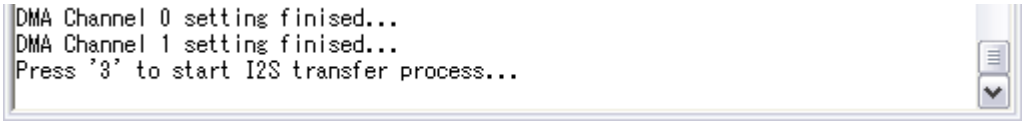

Fig 75. Setting 2 DMA channels

- Press '3' to start I2S operation

After the I2S process is finished, the Receive Buffer data will be displayed as follows:

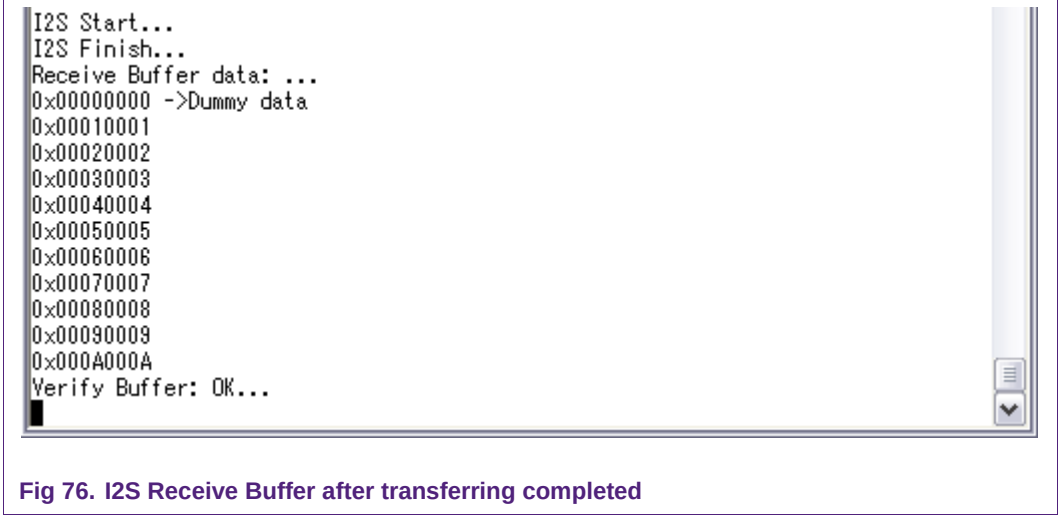

Fig 76. I2S Receive Buffer after transferring completed

3.9.2 I2S_IRQ

3.9.2.1 Example description

Purpose

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

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 config.

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 transmittion finish, "Buffer_Verify()" will check RX and TX buffer data. The result will be print out serial display.

Pls 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.

3.9.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

i2s_irq_test.c: Main program

3.9.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

I2S connection

- 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

3.9.2.4 Running mode

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

3.9.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe data on serial display

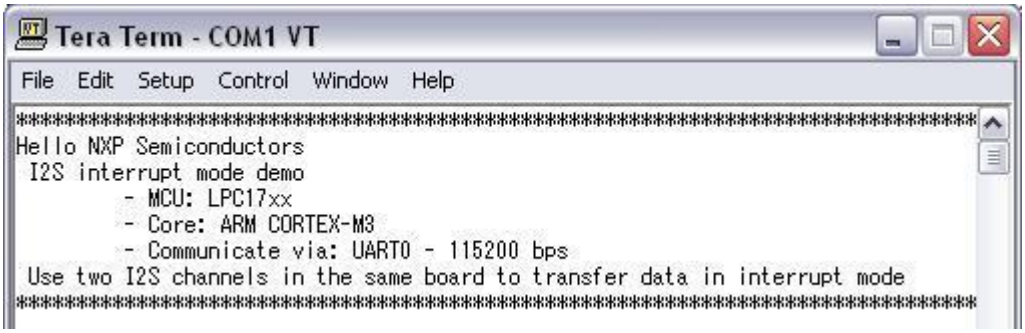

Fig 77. I2S Welcome Screen in interrupt mode

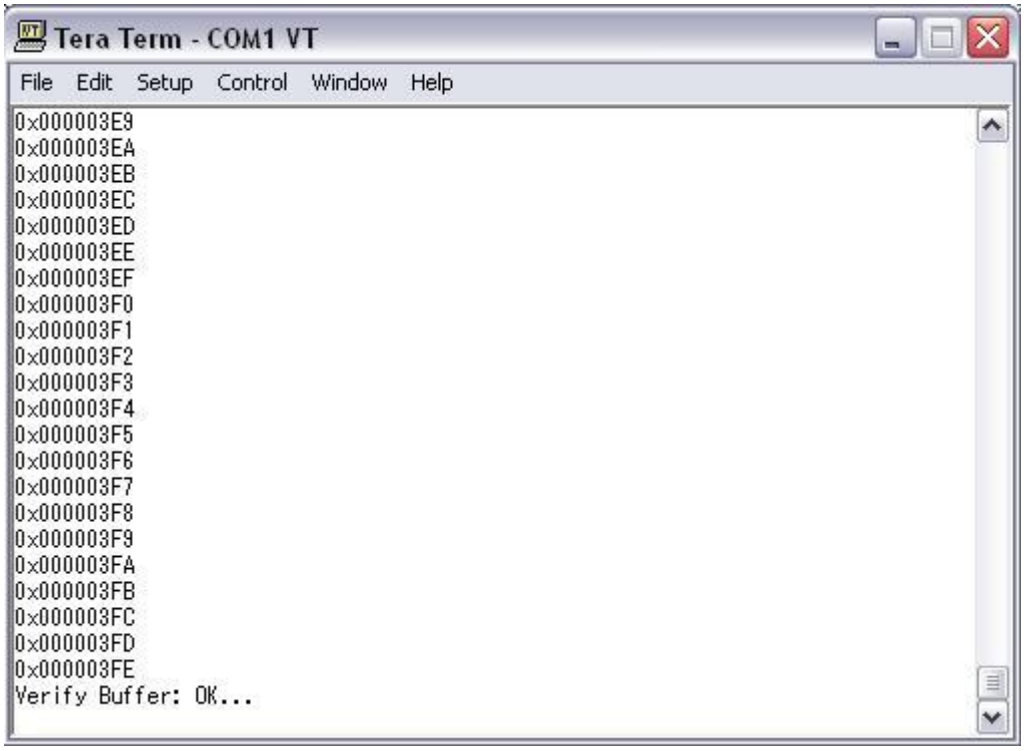

Fig 78. Display and validate received data

3.9.3 I2S_MCLK

3.9.3.1 Example description

Purpose

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

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)

I2S Clock setting:

- Transmitter:
  - Select the RX fractional rate divider clock ouput 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 calculate 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 not must have to connect wire between I2SRX_CLK and I2STX_CLK.

I2S transmit data in interrupt mode. After finished, received and transmited buffer will be verified, the result will be print out serial display.

Pls 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.

3.9.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

i2s_mclk.c: Main program

3.9.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

I2S connection

- P0.5-I2SRX_WS connects to P0.8-I2STX_WS
- P0.6-I2SRX_SDA connects to p0.9-I2STX_SDA

3.9.3.4 Running mode

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

3.9.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe data on serial display

Fig 79. I2S MCLK welcome screen

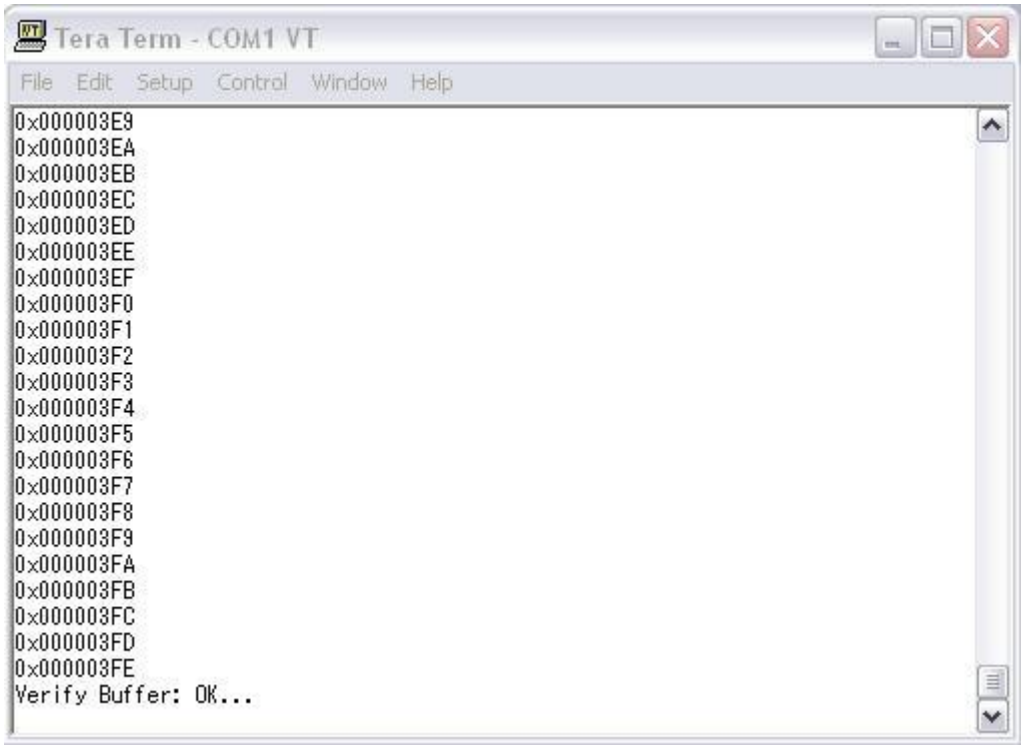

Fig 80. I2S MCLK received and validate data

3.9.4 I2S_test_4_wire

3.9.4.1 Example description

Purpose

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

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 config.

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

Transfer size = 0x200 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 transmittion finised, "Buffer_Verify(void)" 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.

Pls 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.

3.9.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

i2s_test_4_wire.c: Main program

3.9.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

I2S connection

- P0.6-I2SRX_SDA connects to p0.9-I2STX_SDA

3.9.4.4 Running mode

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

3.9.4.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe data on serial display

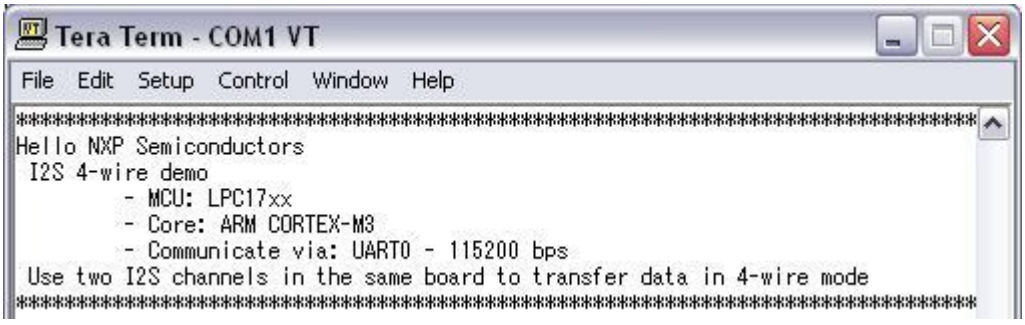

Fig 81. I2S 4-wire welcome screen

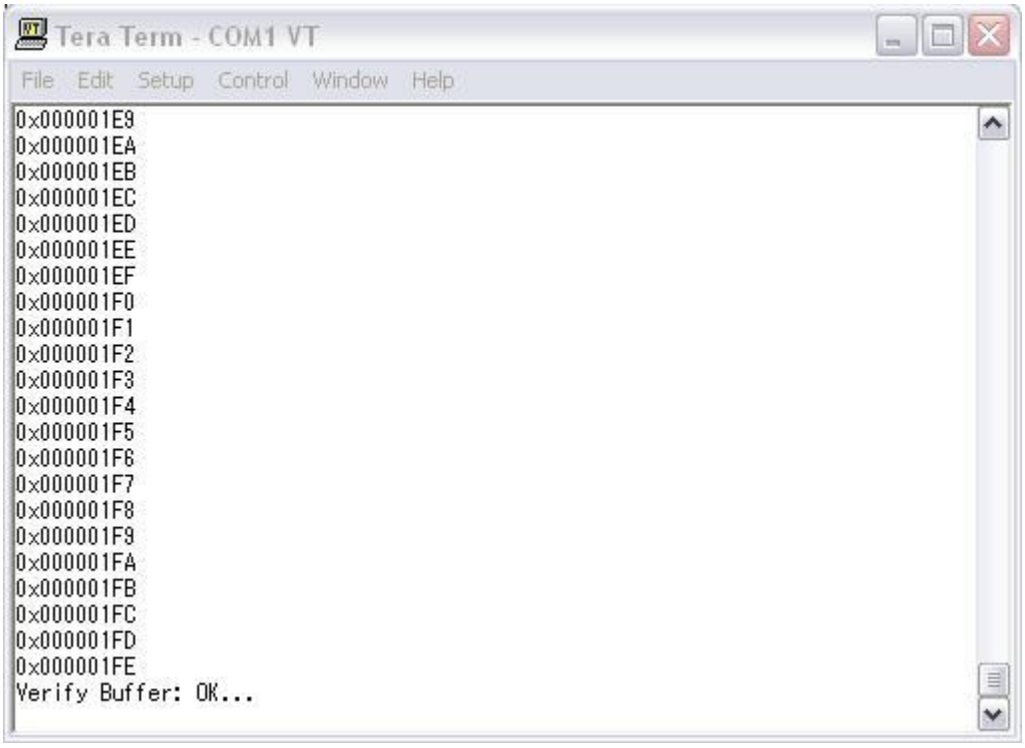

Fig 82. Display and validate received data

3.9.5 I2S_two_kit

3.9.5.1 Example description

Purpose

This example describes how to configure I2S peripheral to communication between two board.

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 config.

I2S peripheral on transmit board will send infinite data to I2S peripheral on receive board.

Data will be increase after each transmittion.

tx_depth_irq = rx_depth_irq = 4.

So whenever FIFO is haft full, it will create IRQ request, I2S interrupt service routine "I2S_IRQHandler()" will be called to send/receive data.

Open serial terminal to observe I2S transfer process.

Pls 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.

3.9.5.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

i2s_two_kit.c: Main program

3.9.5.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

I2S connection

- P0.4(J.116)-I2SRX_CLK on first board connects to P0.7(J.112)-I2STX_CLK on second board.

- P0.5(J.115)-I2SRX_WS on first board connects to P0.8(J.111)-I2STX_WS on second board.
- P0.6(J.113)-I2SRX_SDA on first board connects to p0.9(J.109)-I2STX_SDA on second board.

Common ground must be connected together between two board.

3.9.5.4 Running mode

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

3.9.5.5 Step to run

Step 1: Setting: "I2S_TRANSMIT = 1", build example and burn hex file into transmit board.

Step 2: Setting: "I2S_TRANSMIT = 0", build example and burn hex file into receive board.

```
#include "lpc17xx_i2s.h"
#include "lpc17xx_libcfg.h"
#include "lpc17xx_nvic.h"
#include "lpc17xx_pinsel.h"
#include "debug_frmwrk.h"

/************************** PRIVATE MACROS **************************/
#define I2S_TRANSMIT            1
#define I2S_RECEIVE             !I2S_TRANSMIT
```

Fig 83. Setting I2S_TRANSMIT macro

Note that receive program also can run on RAM mode with debugger if you want.

Step 3: Connect UART0 on receive board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe data on serial display

- Run receive board first
  - Press '1' to start I2S operation

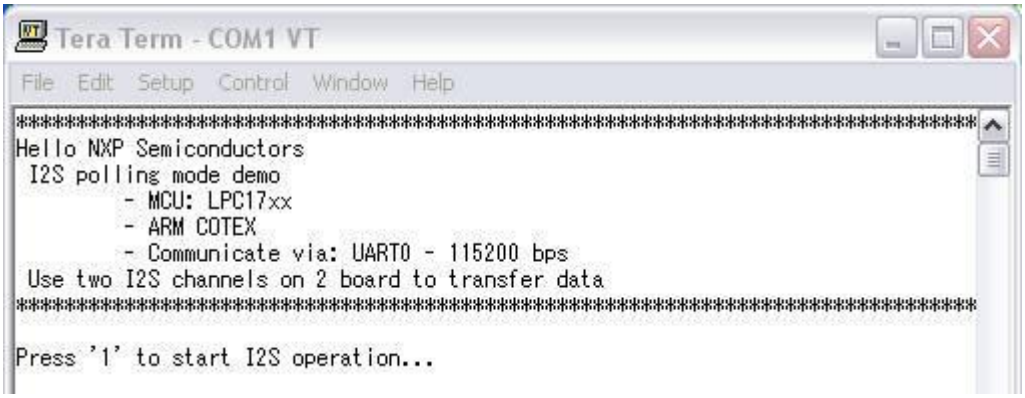

Fig 84. I2S MASTER Welcome Screen

- Then, hit reset button on transmit board

The received data will be display as follows:

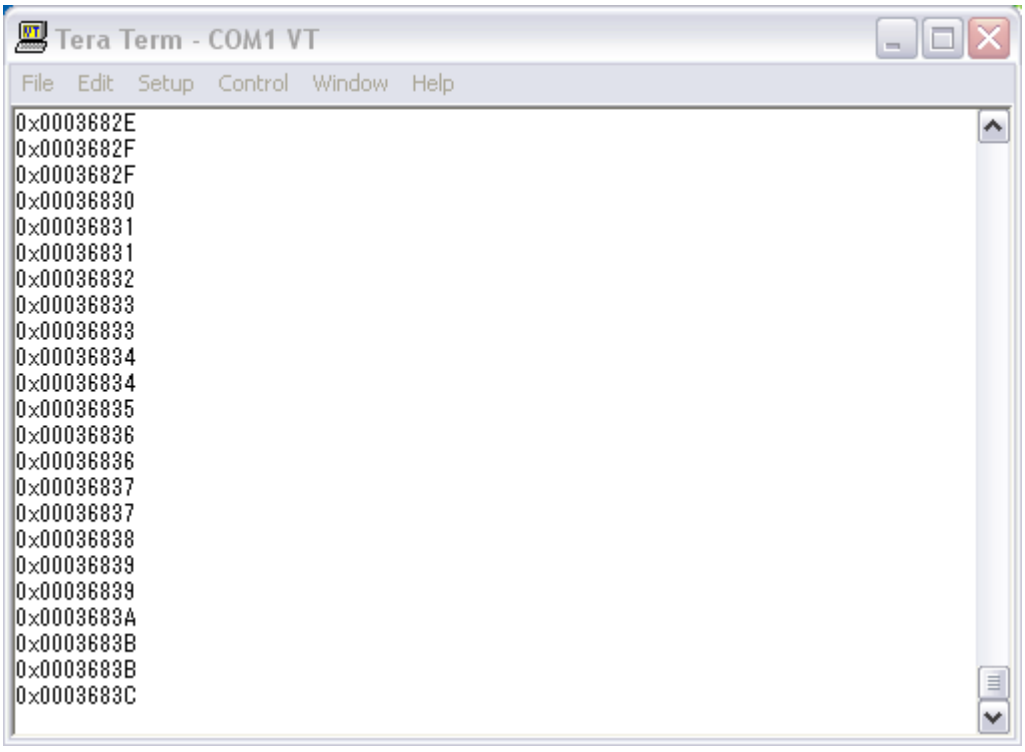

Fig 85. I2S Received data after press '1'

3.9.6 Polling

3.9.6.1 Example description

Purpose

This example describes how to use I2S to transfer data in polling mode

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 config.

Source data are initialized at 'I2S_BUFFER_SRC' and will be copy to 'I2S_BUFFER_DST' by I2S peripheral in polling mode.

After transmittion finised, "Buffer_Verify(void)" 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.

Pls 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.

3.9.6.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

i2s_polling_test.c: Main program

3.9.6.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

I2S connection

- 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

3.9.6.4 Running mode

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

3.9.6.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe data on serial display

Fig 86. I2S Welcome Screen in polling mode

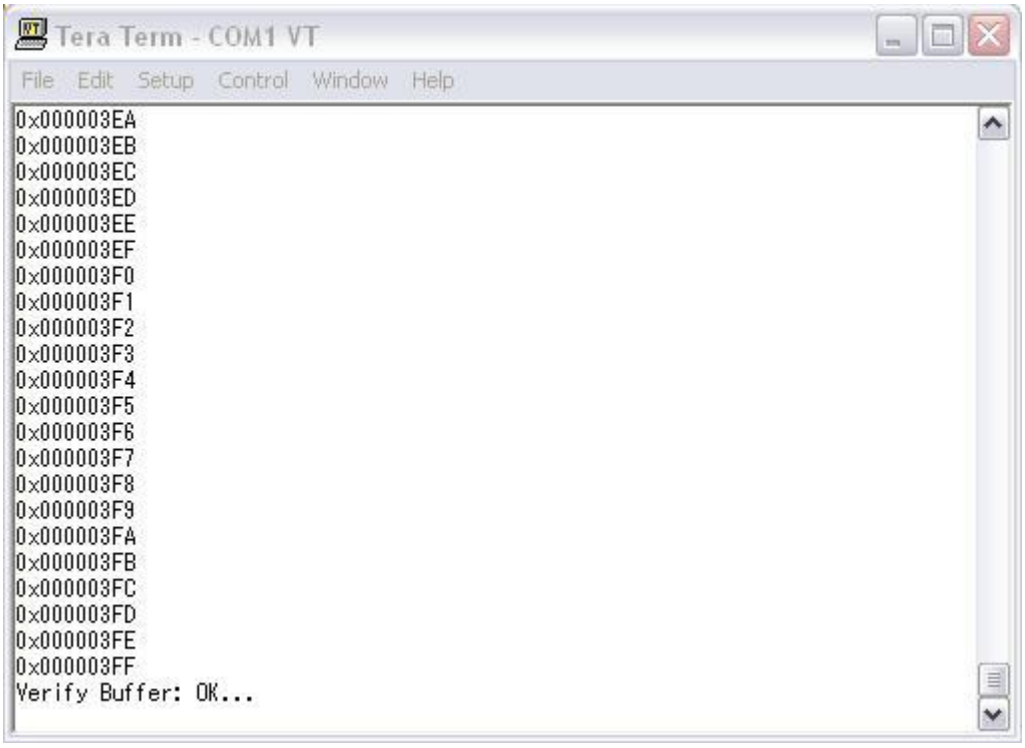

Fig 87. Display and validate received data

3.10 LCD

3.10.1 NOKIA6610_LCD

3.10.1.1 Example description

Purpose

This example describes how to use LCD NOKIA6610 on IAR-LPC1768-KS board

Process

This example use LCD NOKIA6610 driver library that deriverd from IAR.

Using SPI protocol to communicate with LCD controller chip.

It starts by show NXP icon.

LCD operation can be controlled by using:

- AN_TRIM: ADC potentiometer - increase/decrease LCD backlight or contrast.
- BUT1: used to adjust backlight.
- BUT2: used to adjust LCD contrast.

3.10.1.2 Directory contents

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

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

drv_glcd_cbcf.h: Graphic LCD configuration file

drv_glcd.h/.c: Graphic LCD driver

glcd_ll.h/.c: Graphic LCD low level functions

lcdtest.c: Main program

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

NXP_logo.h/.c: NXP icon data

Terminal_9_12x6: font (12x6) data

makefile: Example's makefile (to build with GNU toolchain)

3.10.1.3 Hardware configuration

These jumper must be configured as follows:

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.10.1.4 Running mode

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

3.10.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware as above instruction

Step 4: Hit reset button to run example:

- It shows NXP icon first.
- Hit "BUT1" if want to adjust backlight, then turn ADC potentiometer AN_TRIM to change LCD backlight.
- Hit "BUT2" if want to adjust contrast, then turn ADC potentiometer AN_TRIM to change LCD contrast.

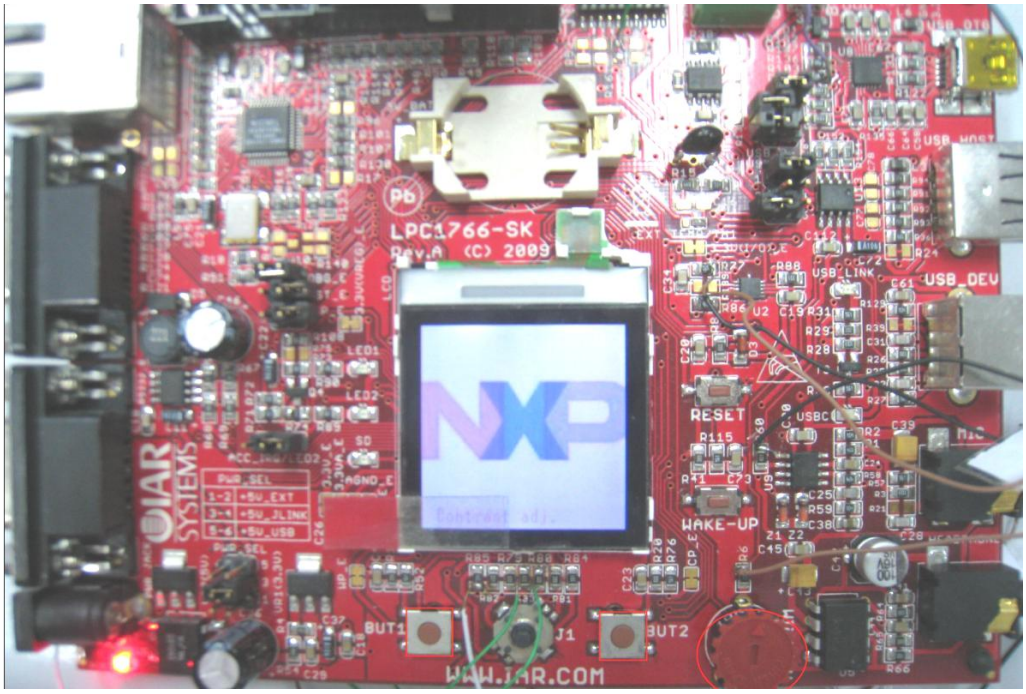

Fig 88. LCD-IAR1768KS board display

3.10.2 QVGA_TFT_LCD

3.10.2.1 Example description

Purpose

This example describes how to configure LCD on MCB1700 board

Process

This example uses Graphic LCD driver library that derivered by Keil.

Using SPI protocol to communicate with LCD controller chip.

This example just uses LCD driver to display simple text on LCD screen.

3.10.2.2 Directory contents

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

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

debug_frmwrk.h/.c: Debug framework header/source file - support utilities that used for debugging through UART port

Font_24x16.h: Data of font 24x16

GLCD_SPI_LPC1700.c: LPC1700 low level Graphic LCD driven with SPI functions

GLCD.h: Graphic LCD function prototypes and defines

lcdtest.c: Main program

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

3.10.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

3.10.2.4 Running mode

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

3.10.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware as above instruction

Step 4: Run example, see simple text displays on LCD screen

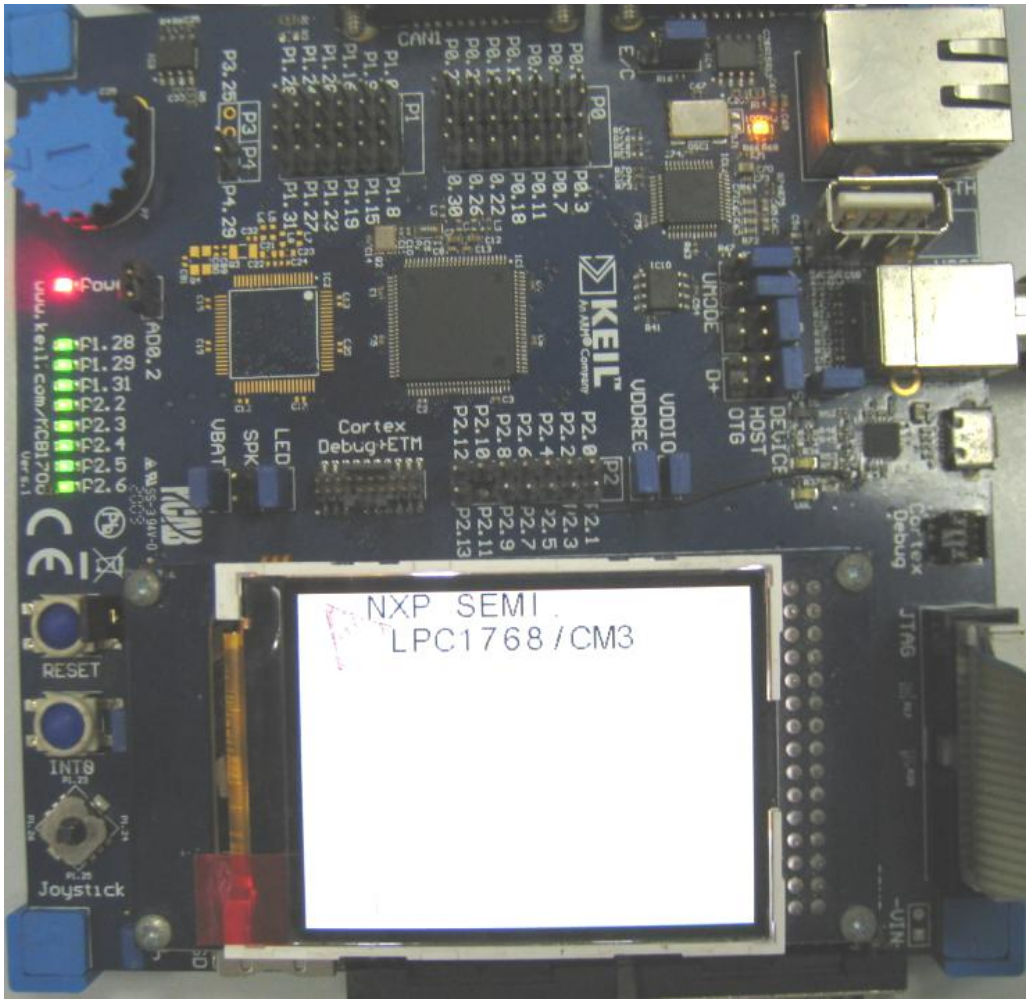

Fig 89. Display and validate received data

3.11 MCPWM

3.11.1 MCPWMSimple

3.11.1.1 Example description

Purpose

This example describes how to test Motor Control PWM module in LPC17xx.

Process

Tested 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 channel 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.

3.11.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

mcpwm_simple.c: Main program

3.11.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.11.1.4 Running mode

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

3.11.1.5 Step to run

Step 1: Chose MCPWM mode that want to test

- If want to test 3-phase DC mode, setting:

  #define DC_MODE_TEST 1

- If want to test 3-phase AC mode, setting:

  #define AC_MODE_TEST 1

- If want to test Capture MCPWM mode, setting:

  #define CAPTURE_MODE_TEST 1

  (Should not enable DC mode and AC mode at the same time)

Step 2: Build example.

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect UART0 on this board to COM port on your computer

Step 5: Configure hardware and serial display as above instruction

Step 6: Run example. See capture result on serial display (if use Capture mode)

And use oscilloscope to monitor the wave form.

- Enable AC_MODE_TEST

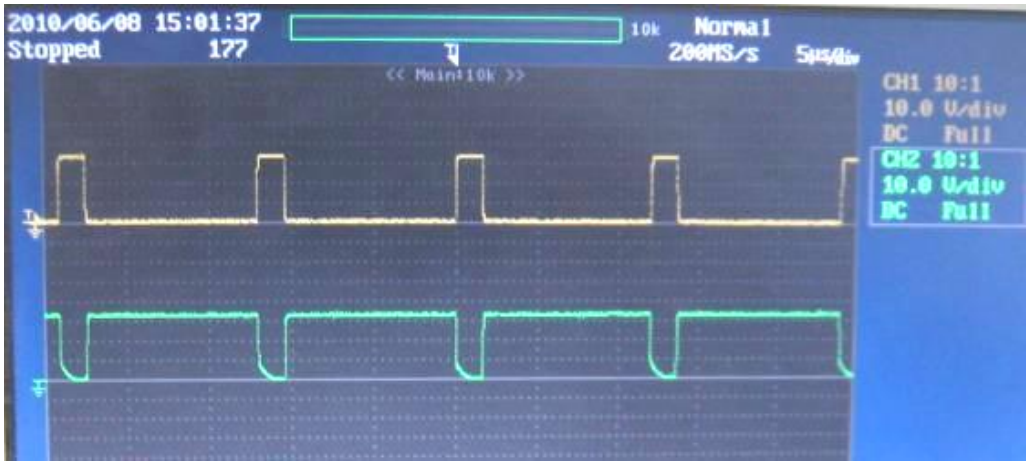

Fig 90. MCOA0 (P1.19) and MCOB0 (P1.22) signal display

- Enable DC_MODE_TEST

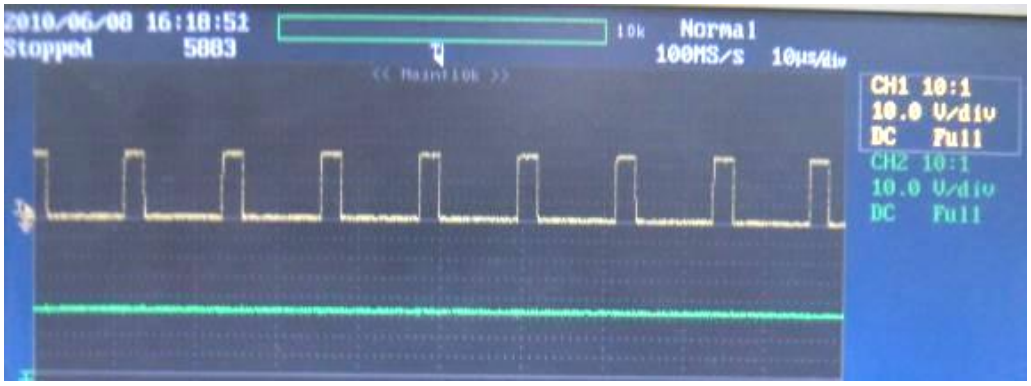

Fig 91. MCOA0 (P1.19) and MCOB0 (P1.22) signal display

- Enable CAPTURE_MODE_TEST

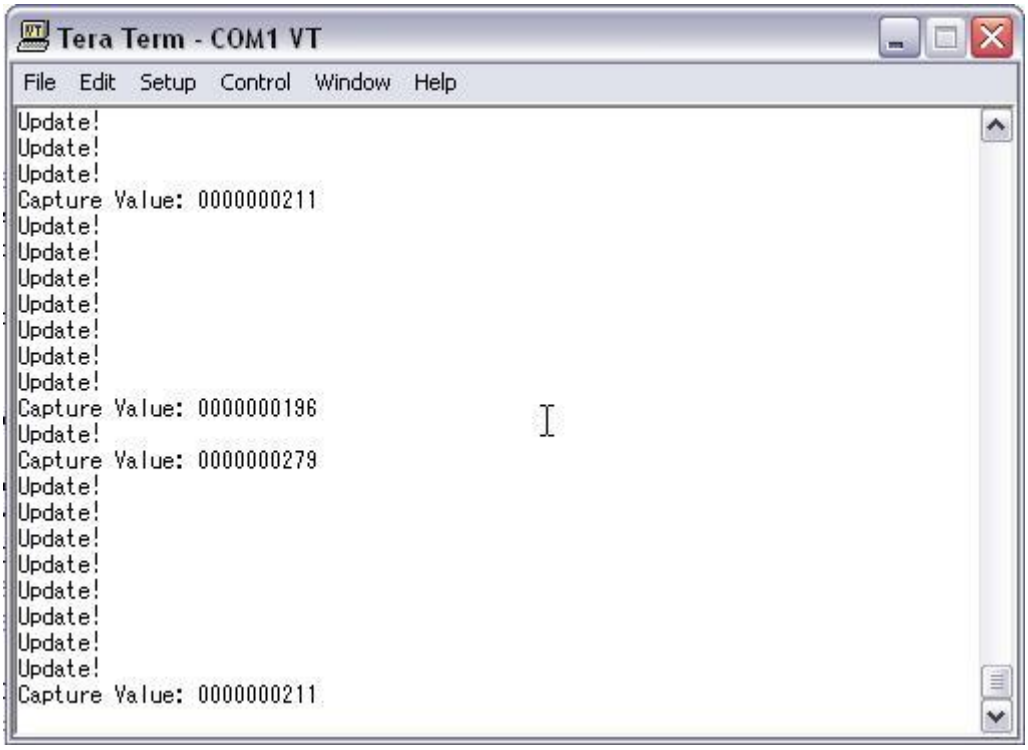

Fig 92. Capture value on serial display

3.12 NVIC

3.12.1 Priority

3.12.1.1 Example description

Purpose

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

Process

This example uses 2 external interrupt 0 and 3 as IRQ channels.

Setting "INT_MODE" macro to chose interrupt mode test.

- INT_MODE = 0: Tail-chainning interrupt testing
    - EXT0 is assigned group-priority = 0, sub-priority = 2
    - EXT3 is assigned group-priority = 0, sub-priority = 1

Two IRQ channels has same group.So, new ISR can not pre-empt previous interrupt when it's executing even if new ISR has higher priority than curren ISR

- INT_MODE = 1: Late-arriving interrupt testing.
    - EXT0 is assigned group-priority = 0, sub-priority = 0
    - EXT3 is assigned group-priority = 1, sub-priority = 0

EXT0 has higher group-priority than EXT3, so EXT0 can pre-empt EXT3 when it's executing

In this example:

- EXT0 occurs when pressing button INT0.

EXT0 interrupt will blink LED P1.29 10 times.

- EXT3 occurs when turning ADC potentiometer until pull-down GPIO P0.25 pin (ADC0.2).

EXT3 interrupt will blink LED P1.28 10 times

3.12.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

nvic_priority.c: Main program

3.12.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON

- INT0: ON
- Remain jumper: OFF

3.12.1.4 Running mode

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

3.12.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardward as above instruction

Step 4: Run example

- Test tail-chaining interrupt mode:

Hit INT0 button to generate EXT0. Led P1.29 will blink 10 times

When P1.29 is blinking, turn ADC potentiometer and see if EXT3 pre-empt EXT0 or not.

When EXT0 executes finised, EXT3 will bink led P1.28 10 times.

- Test late-arriving interrupt mode:

Turn ADC potentiometer to generate EXT3, Led P1.28 will blink 10 times.

When P1.28 is blinking, press INT0 button to see EXT0 pre-empt EXT3.

Led P1.29 will blink instead led P1.28. After P1.29 blink 10 times, P1.28 continues blink remain times.

3.12.2 VecTable_Relocation

3.12.2.1 Example description

Purpose

This example describes how to relocation vetor table.

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 init address to new address.

Check VT remapping successful or not, use SysTick interrupt to blink LEDs on board.

3.12.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

vt_relocation.c: Main program

3.12.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

3.12.2.4 Running mode

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

3.12.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and see LED P1.28 blink or not.

If it is blinking, Vector Table remapping is successful.

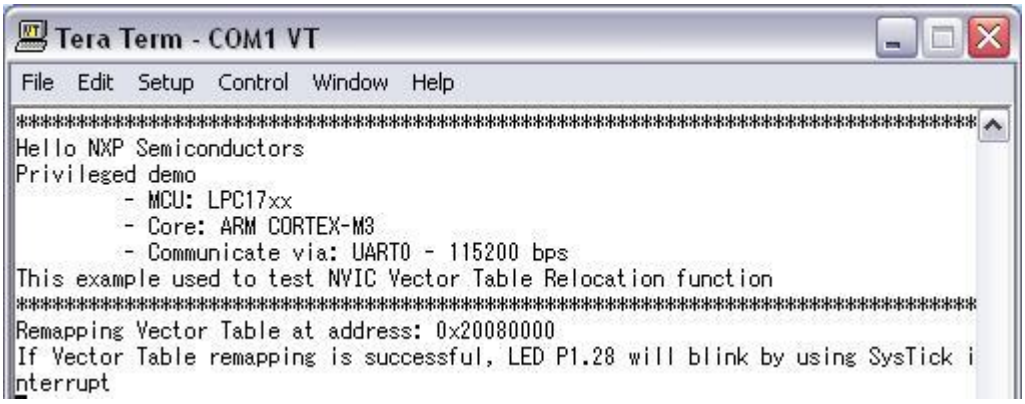

Fig 93. Vector Table Relocation demo

3.13 PWM

3.13.1 Dual_Edge

3.13.1.1 Example description

Purpose

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

Process

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

Peripheral clock for PWM: PWM_PCLK = CCLK / 4 = 72MHz/4 = 18MHz and there is no prescale for PWM. The PWM timer/counter clock is at 18MHz. The base rate is set to 100

The base PWM frequency is at 18MHz/100 = 180 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

3.13.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

pwm_dual_edge.c: Main program

3.13.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON

- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

PWM Pin selected

Observe PWM wave signal on these pin

- PWM1.2 (channel 2): P2.1
- PWM1.4 (channel 4): P2.3
- PWM1.5 (channel 5): P2.4

3.13.1.4 Running mode

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

3.13.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware and serial display as above instruction

Step 4: Run example

Use oscilloscope to monitor the wave form

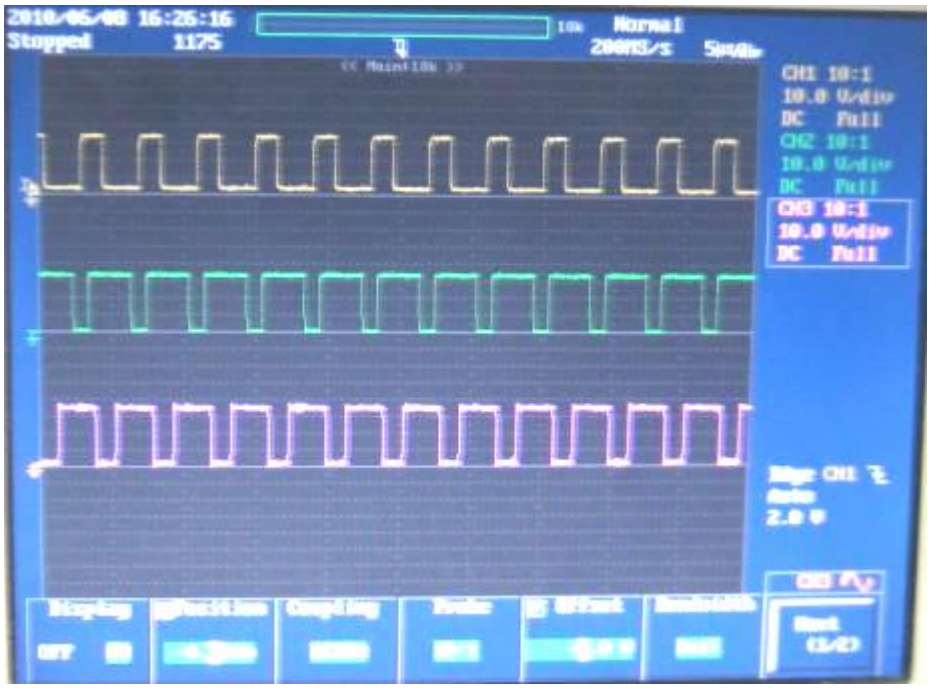

Fig 94. PWM1.2, PWM1.4 and PWM1.5 signals

3.13.2 Match_Interrupt

3.13.2.1 Example description

Purpose

This example describes how to use PWM Match function in interrupt 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 and there is no prescale 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 = 0.56 microsecond) (P2.0)
- PWM1.2 = (20/256) (period = 1.11 microsecond) (P2.1)
- PWM1.3 = (30/256) (period = 1.67 microsecond) (P2.2)
- PWM1.4 = (40/256) (period = 2.22 microsecond) (P2.3)
- PWM1.5 = (50/256) (period = 2.78 microsecond) (P2.4)
- PWM1.6 = (60/256) (period = 3.33 microsecond) (P2.5)

Using Oscilloscope to observe the PWM signals

Here, PWM1.1 value keeps changing, it will increase 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), an interrupt for matching will generate and update the value of PWM1.1, this value will be updated every 4096 match interrupts or:

Period * 4096 = 14.22 * 4096 = 58,245 (microsecond)

And this value will be reset to 0 after:

Period * 4096 * 256 = 14,910,750.72 (microsecond) = ~15 (second)

3.13.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

pwm_match_int.c: Main program

3.13.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON

- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

PWM Pin selected

- PWM1.1 (channel 1): P2.0
- PWM1.2 (channel 2): P2.1
- PWM1.3 (channel 3): P2.2
- PWM1.4 (channel 4): P2.3
- PWM1.5 (channel 5): P2.4
- PWM1.6 (channel 6): P2.5

3.13.2.4 Running mode

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

3.13.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware and serial display as above instruction

Step 4: Run example

Use oscilloscope to monitor the wave form

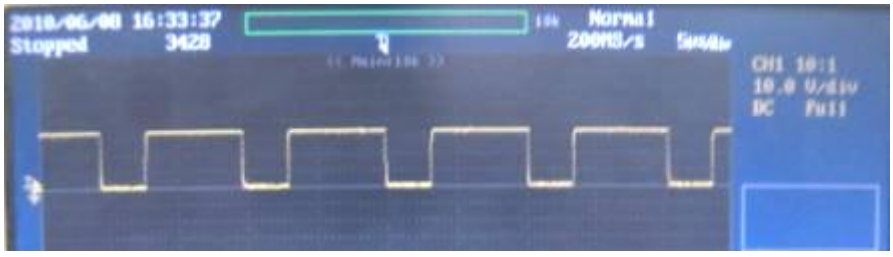

Fig 95. PWM1.1 signals, this signal change continuously

Fig 96. PWM1.2, PWM1.3, PWM1.4 signals

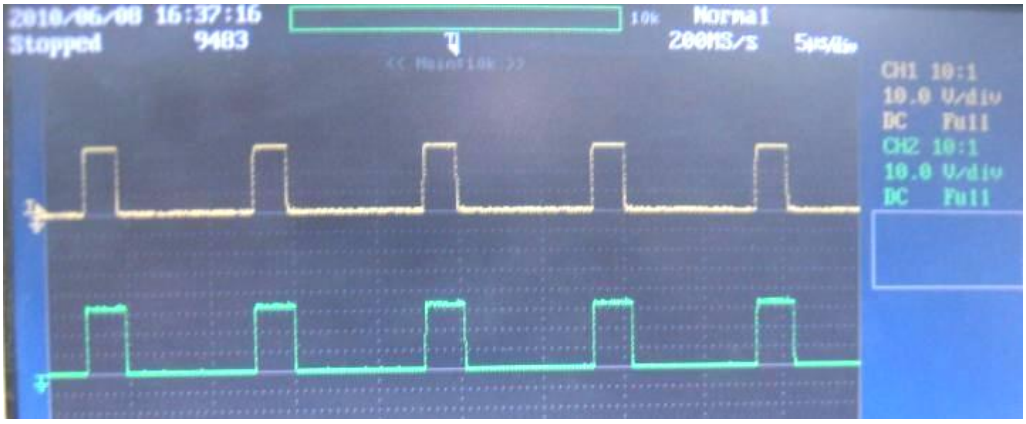

Fig 97. PWM1.5, PWM1.6 signals

3.13.3 Single_Edge

3.13.3.1 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 and there is no prescale 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 = 0.56 microsecond)(P2.0)
- PWM1.2 = (20/256) (period = 1.11 microsecond)(P2.1)
- PWM1.3 = (30/256) (period = 1.67 microsecond)(P2.2)
- PWM1.4 = (40/256) (period = 2.22 microsecond)(P2.3)
- PWM1.5 = (50/256) (period = 2.78 microsecond)(P2.4)
- PWM1.6 = (60/256) (period = 3.33 microsecond)(P2.5)

Using Oscilloscope to observe the PWM signals

3.13.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

pwm_single_edge.c: Main program

3.13.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

PWM Pin selected

- PWM1.1 (channel 1): P2.0
- PWM1.2 (channel 2): P2.1
- PWM1.3 (channel 3): P2.2
- PWM1.4 (channel 4): P2.3
- PWM1.5 (channel 5): P2.4
- PWM1.6 (channel 6): P2.5

3.13.3.4 Running mode

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

3.13.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware as above instruction

Step 4: Run example

Use oscilloscope to monitor the wave form on pin P2.0, P2.1, P2.2, P2.3, P2.4, P2.5

Fig 98. PWM1.1, PWM1.2, PWM1.3 signals

Fig 99. PWM1.4, PWM1.5, PWM1.6 signals

3.14 PWR

3.14.1 EXTINT_Sleep

3.14.1.1 Example description

Purpose

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

Process

It has some difference about testing on MCB1700 and IAR1700 board:

- MCB1700:
  - LEDs:
    - LED1 - P1.28
    - LED2 - P1.29
  - EXTINT: use external interrupt 0
  - INT button: INT0
- IAR1700:
  - LEDs:
    - LED1 - P1.25
    - LED2 - P0.4
  - EXTINT: use external interrupt 2
  - INT button: WAKE-UP

First LED will be blinked in normal mode. And when receive '1' character from serial display, the system call 'CLKPWR_Sleep()' function to enter sleep mode. Then, when press INT button to to generate external interrupt, it will wake-up system and blink second LED.

3.14.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

extint_sleep.c: Main program

3.14.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- INT0: ON

- LED: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- ACC_IRQ/LED2: 2-3 (LED2)
- Remain jumper: OFF

3.14.1.4 Running mode

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

3.14.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe result on serial display

- Hit RESET button to run example

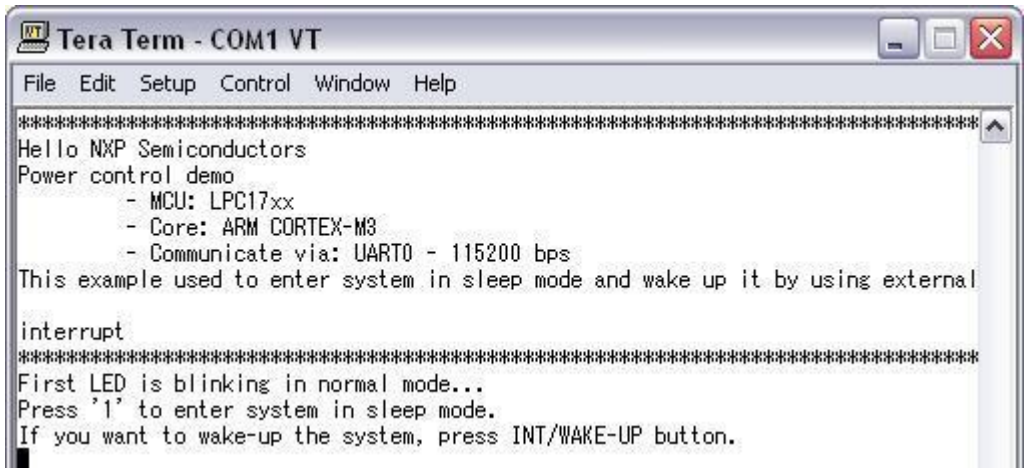

Fig 100. In normal mode

- Press '1' to enter sleep mode

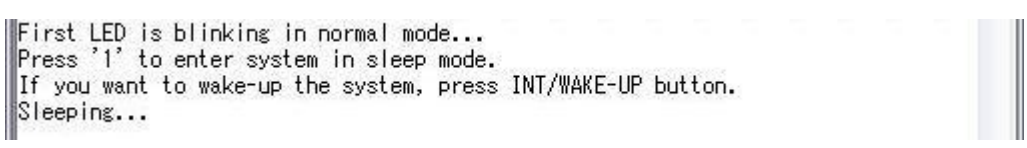

Fig 101. In sleep mode

- Press INT button:

- MCB1700: INT0 button
- IAR1700: WAKE-UP button

to wake-up the system

```
First LED is blinking in normal mode...
Press '1' to enter system in sleep mode.
If you want to wake-up the system, press INT/WAKE-UP button.
Sleeping...
System wake-up! Second LED is blinking...
```

Fig 102. After wake-up

3.14.2 WDT_DeepSleep

3.14.2.1 Example description

Purpose

This example describes how to enter system in DeepSleep mode and wake up it by using WDT Interrupt

Process

WDT setting:

- Source clock: IRC oscillator
- Interrupt mode
- Timeout = 2000000 us = 2s

Enter system in DeepSleep by using CLKPWR_DeepSleep function (call __WFI())

After 2s, WDT timeout and generate interrupt to wake up system.

3.14.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

wdt_deepsleep.c: Main program

3.14.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.14.2.4 Running mode

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

3.14.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe result on serial display

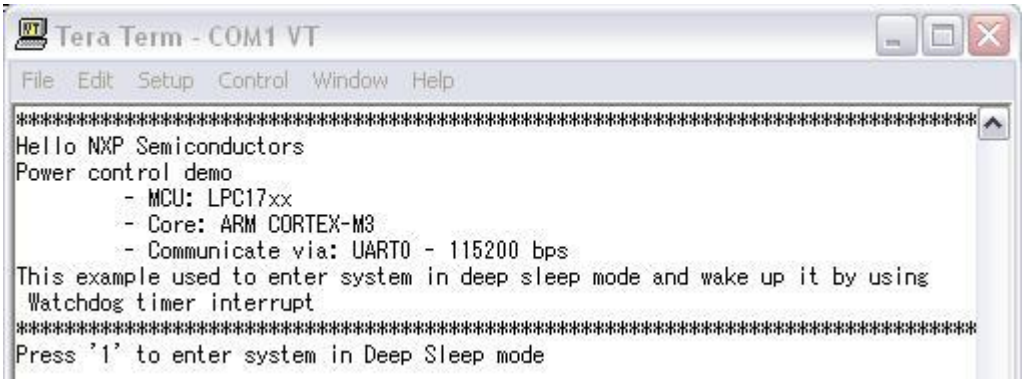

Fig 103. DeepSleep welcome screen

Press ‘1’ to enter system in DeepSleep mode

Fig 104. DeepSleeping…

Wait 2s…

WDT timeout and wake-up system.

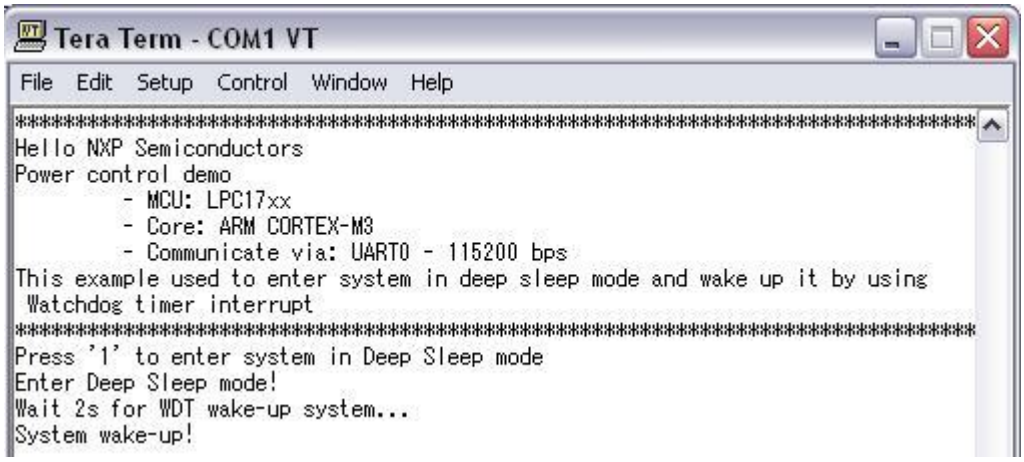

Fig 105. System wake-up from deepsleep mode

3.14.3 NMI_PowerDown

3.14.3.1 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. When connect, a logic 1 on the pin will cause the NMI to be processed.

Select P2.10 as NMI pin

At the first time, sure that NMI pin has logic 1 because if it connects with ground -> ISP is pulled low, system will enter in ISP mode. After that, call CLKPWR_PowerDown() to enter system in PowerDown mode

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

When system is PowerDowning..., pulls up NMI to logic 1, NMI occurs and wakes-up system.

3.14.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

nmi_powerdown.c: Main program

3.14.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.14.3.4 Running mode

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

3.14.3.5 Step to run

- ***Step 1:*** Build example.
- ***Step 2:*** Burn hex file into board (if run on ROM mode)
- ***Step 3:*** Connect UART0 on this board to COM port on your computer
- ***Step 4:*** Configure hardware and serial display as above instruction
- ***Step 5:*** Run example and observe on serial display
  - First, sure that NMI pin (P2.10) has logic 1
  - Press "RESET" button to run example

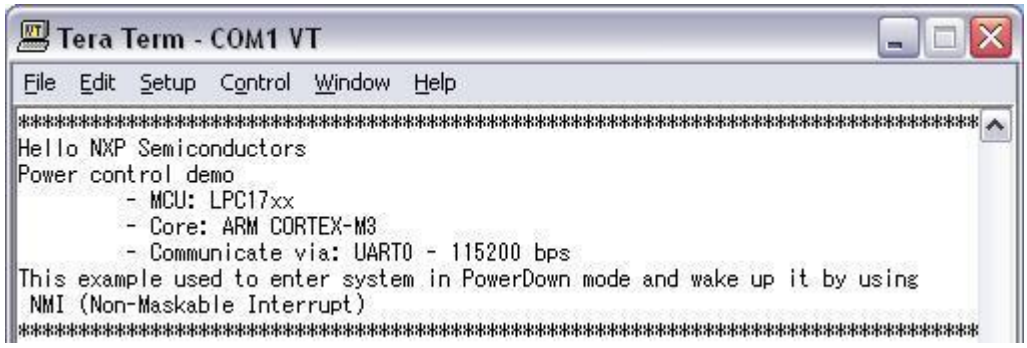

Fig 106. NMI-PowerDown welcome screen

  - Connect P2.10 with ground to disable NMI interrupt

Note: MCB1700 board doesn't wire ground pin output, so you can use UMODE.3 instead

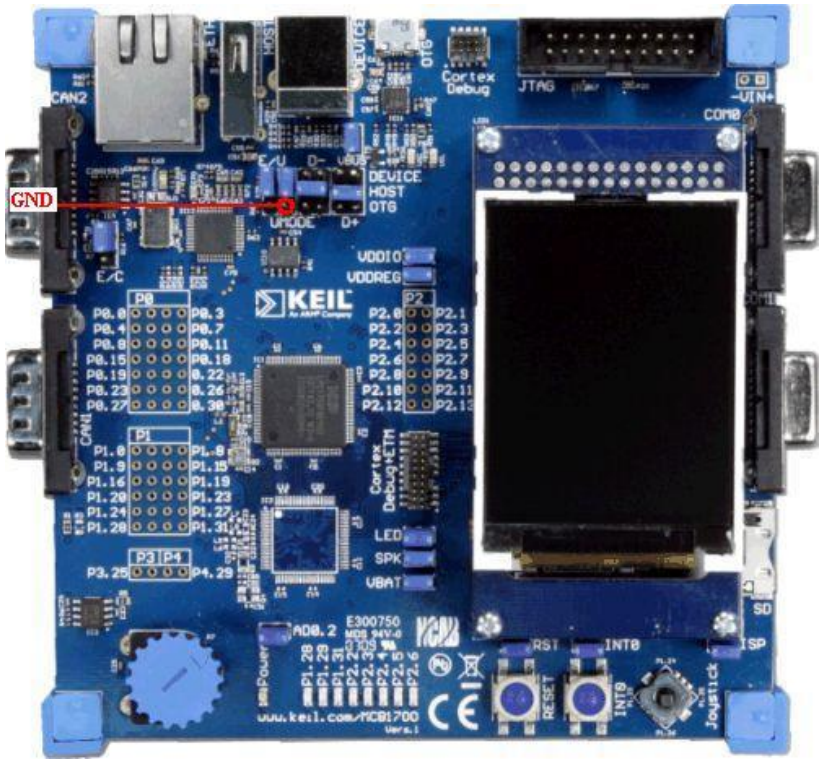

Fig 107. UMODE.3 (ground pin)

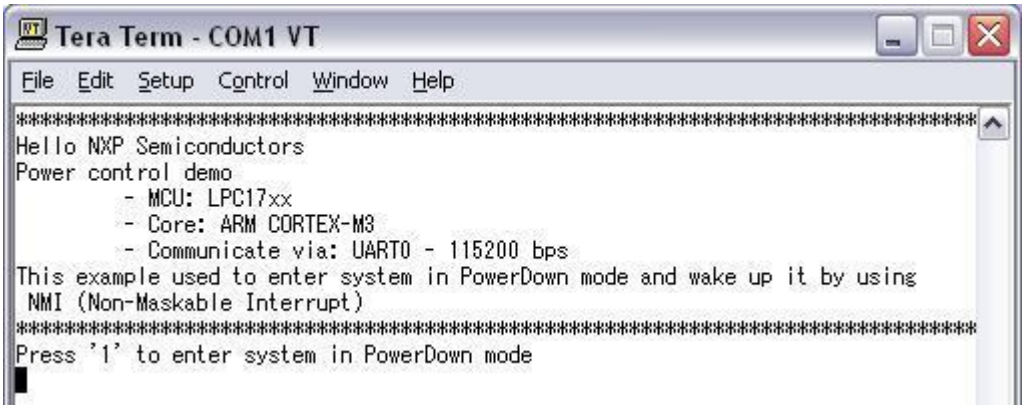

Fig 108. NMI-PowerDown after connect NMI pin with ground

- Press '1' to enter system in DeepSleep mode

Fig 109. Enter system to deep-sleep mode

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

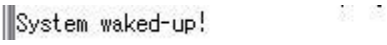

Fig 110. System waked-up

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

3.14.4 RTC_DeepPWD

3.14.4.1 Example description

Purpose

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

Process

When system enter 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 both 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.

3.14.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

rtc_deeppwd.c: Main program

3.14.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.14.4.4 Running mode

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

3.14.4.5 Step to run

- ***Step 1:*** Build example.
- ***Step 2:*** Burn hex file into board (if run on ROM mode)
- ***Step 3:*** Connect UART0 on this board to COM port on your computer
- ***Step 4:*** Configure hardware and serial display as above instruction
- ***Step 5:*** Run example and observe on serial display

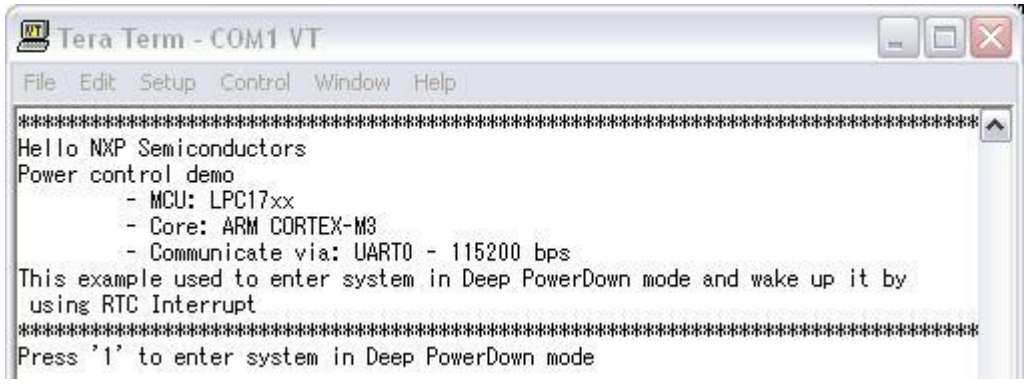

Fig 111. Deep PowerDown demo

- Press '1' to enter system in Deep PowerDown mode

Fig 112. System enter in Deep PowerDown mode

- Wait 5s to RTC wake-up system automatically or press "RESET" button to wake it up immediatly.

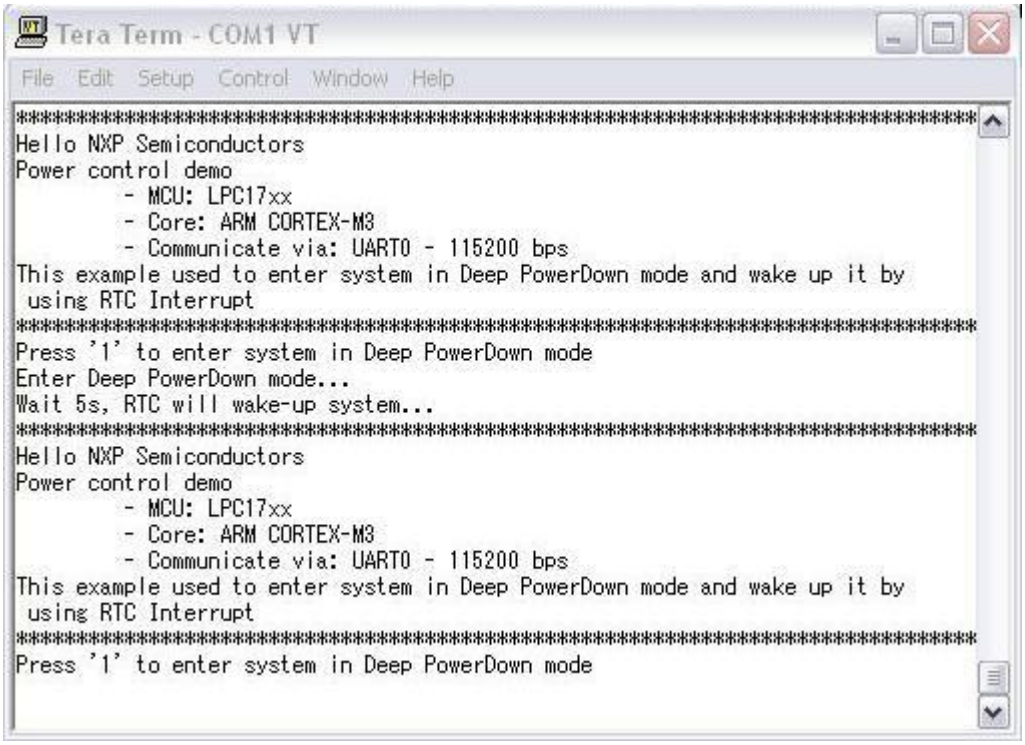

Fig 113. System waked-up

3.15 QEI

3.15.1 QEI_Velo

3.15.1.1 Example description

Purpose

This example describes how to use Quadrature Encoder Interface module to calculate velocity

Process

This is just a simple QEI demo for demonstrate QEI operation on LPC17xx

This example use a timer match interrupt to generate a virtual QEI signal output for QEI module to capture (Phase A and Phase B channel). In case of using QEI virtual signal, symbol VIRTUAL_QEI_SIGNAL must be set to 1.

In this case, two GPIO output pin on port 0 can be used to toggle state that defined through PHASE_A_PIN and PHASE_B_PIN. The MCFB0 (PHA - Phase A input) and MCFB1 (PHB - Phase B input) therefore must be connect to these GPIO pins.

In this case, a 'virtual encoder' that has these following parameter:

- 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 (Hz)

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

3.15.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

qei_test_velo.c: Main program

3.15.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

QEI connection

Use P0.19 and P0.21 to generate virtual signal supply for QEI peripheral by using timer match interrupt output.

Connect:

- P0.19 to P1.20 (MCI0)
- P0.21 to P1.23 (MCI1)

3.15.1.4 Running mode

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

3.15.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe data on serial display

Screen wil be displayed like this:

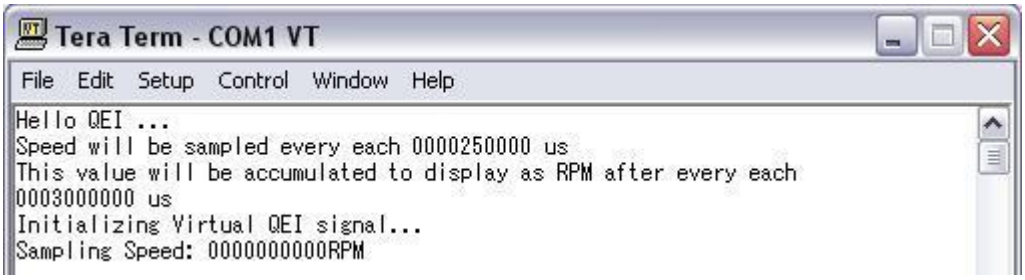

Fig 114. QEI init

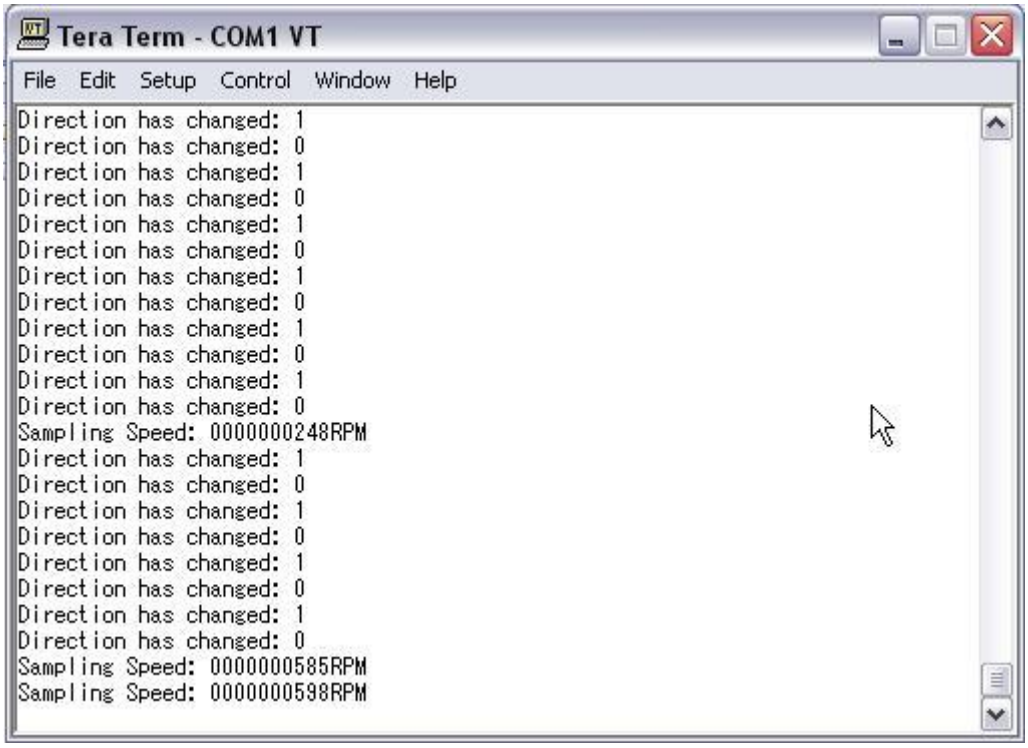

Fig 115. QEI process after connect pin

3.16 RIT

3.16.1 Interrupt

3.16.1.1 Example description

Purpose

This example describes how to use RIT as a timer to generate interrupt to drive LED

Process

RIT time interval configure is 1000 ms = 1s.

So each 1s, RIT will generate interrupt and invoke RIT_IRQHandler( ) to turn on/off LED.

- MCB board, LED2.2 is available
- IAR board, LED1 is available

3.16.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

rit_interrupt.c: Main program

3.16.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.16.1.4 Running mode

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

3.16.1.5 Step to run

Step 1: Choose correct working board by uncomment correct defined board in main.c file

- MCB1700 board, uncomment "#define MCB_LPC_1768"
- IAR-LPC1768-KS board, uncomment "#define MCB_LPC_1768"

  (Should not uncomment both symbols at the same time)

```
//#define MCB_LPC_1768
#define IAR_LPC_1768
```

```
#define MCB_LPC_1768
//#define IAR_LPC_1768
```

Fig 116. Choose working board

Step 2: Build example.

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect UART0 on this board to COM port on your computer

Step 5: Configure hardware and serial display as above instruction

Step 6: Run example.

Note:

- MCB1700 will blink LED2.2 for testing
- IAR KS will blink LED1 for testing

The screen will be displayed like this:

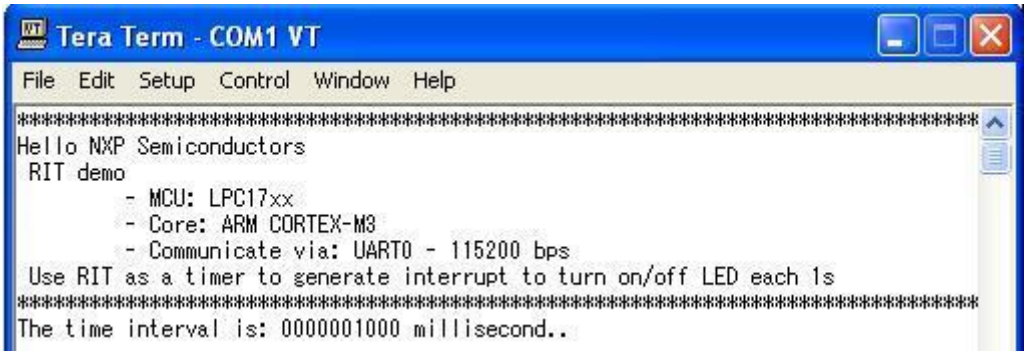

Fig 117. RIT example demo

3.17 RTC

3.17.1 AlarmCntIncrInterrupt

3.17.1.1 Example description

Purpose

This example describes how to generate interrupt in Second Counter Increment Interrupt (1s) and generate Alarm interrupt at 10s

Process

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

8:00:00PM, 2009-04-24

And set Alarm time at 10s

RTC is set generate interrupt each second, so each 1s it will call RTC_IRQHandler( ), it will get time from RTC register and display on serial screen. It all check if alarm match interrupt occurs or not. (This interrupt just only occurs at 10s). So after 10s, alarm interrupt occurs and a notice sentence will be wirte on serial display screen.

3.17.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

rtc_alarm_cntincr_int.c: Main program

3.17.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.17.1.4 Running mode

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

3.17.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe data from serial display

The screen will be displayed like this:

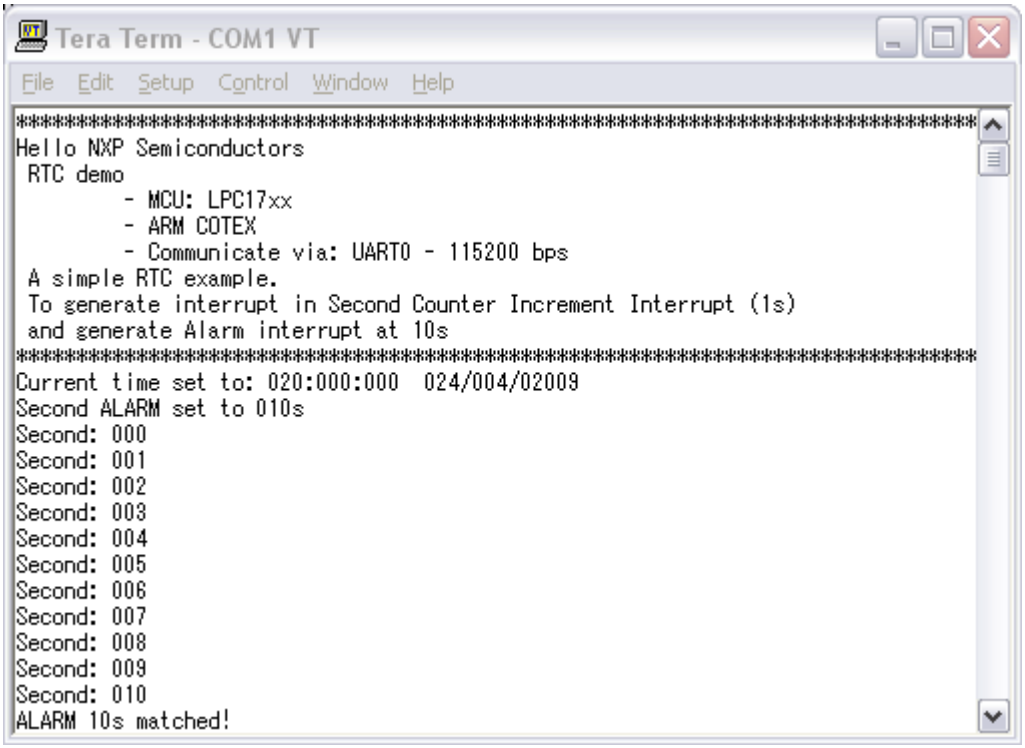

Fig 118. RTC demo

3.17.2 Calibration

3.17.2.1 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 ocunter, or by incrementing the counter by 2 instead 1. This allow calibrating 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

After each 5s, real-time clock will adjust automatically by incrementing the counter by 2 instead of 1. You can observe calibration process via serial display.

3.17.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

rtc_calib.c: Main program

3.17.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.17.2.4 Running mode

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

3.17.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe data from serial display

The calibration process like this:

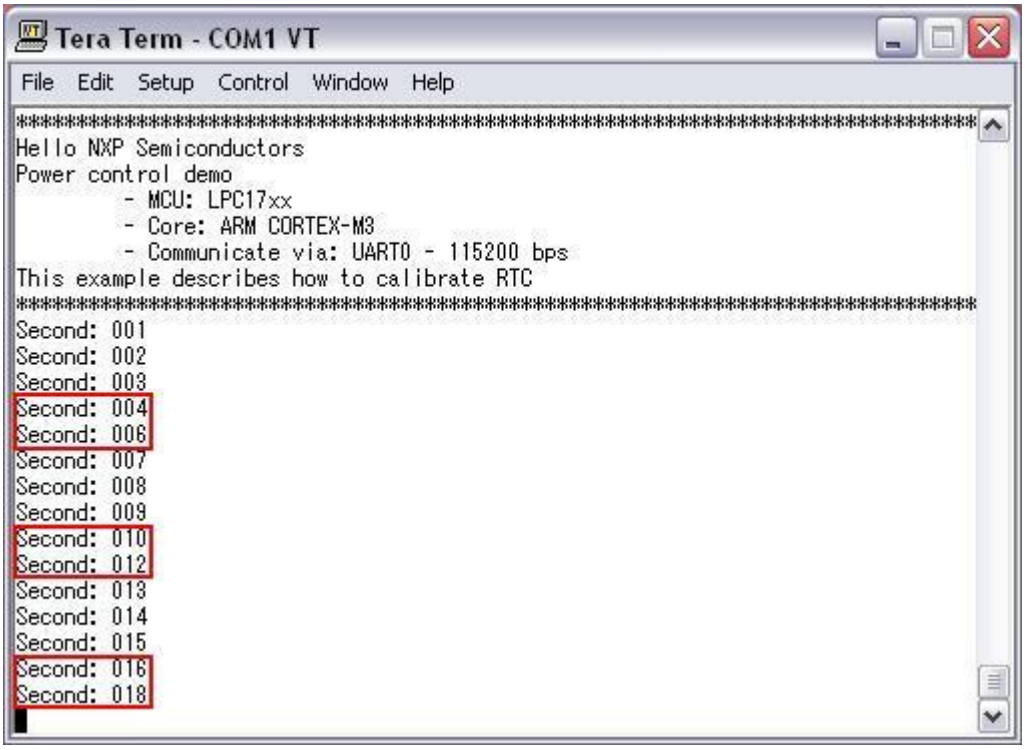

Fig 119. RTC calibration demo

3.18 SPI

3.18.1 LoopBack

3.18.1.1 Example description

Purpose

This example describes how to test SPI operation by using loop-back mode

Process

SPI configuration:

- CPHA = 1: data is sampled on the second clock edge of SCK.
- CPOL = 1: SCK is active low
- Clock rate = 2MHz
- LSBF = 0: SPI data is transferred MSB first
- BITS = 10: 10 bits per transfer
- MSTR = 1: SPI operates in Master mode

After initialize buffer, SPI will transfer data to itself (loop-back mode) by calling 'SPI_ReadWrite( )' function and use POLLING mode to send/receive data.

After transmittion completed, receive and transmit buffer will be compare, it they are not similare, the program will be enter infinite loop and a error notice will be displayed.

3.18.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

spi_loopback_test.c: Main program

3.18.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SPI connection

- P0.17 - MISO

- P0.18 - MOSI

MOSI pin must be cross-connected with MISO pin (loop-back mode)

3.18.1.4 Running mode

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

3.18.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe SPI transfer result from serial display

The screen will be displayed like this:

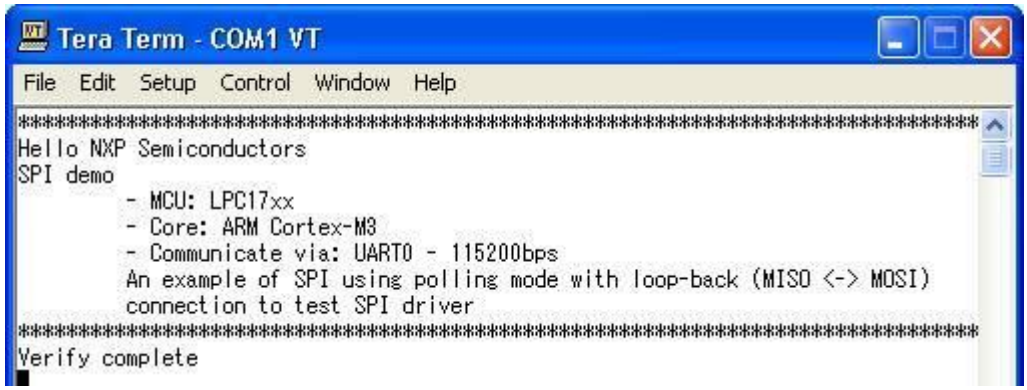

Fig 120. SPI Loopback demo

3.18.2 Master

3.18.2.1 Example description

Purpose

This example describes how to use SPI as master to connect with SPI slave

Note that: this example will run combine with slave example at folder \Slave.

(Pls see "Step to run" for more information)

Process

SPI configuration:

- CPHA = 0: data is sampled on the second clock edge of SCK.
- CPOL = 1: SCK is active low
- Clock rate = 2MHz
- LSBF = 0: SPI data is transferred MSB first

- BITS = 10: 10 bits per transfer
- MSTR = 1: SPI operates in Master mode

Using SSEL as GPIO pin to handler chip select signal via 'CS_Force( )' function.

After initialize buffer, SPI master will transfer/receive data to/from SPI slave in POLLING mode by calling 'SPI_ReadWrite( )' function.

After transmittion completed, receive and transmit buffer will be compare, if they are not similare, the program will be enter infinite loop and a error notice will be displayed.

3.18.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

spi_master.c: Main program

3.18.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SPI connection

- P0.15 - SCK on master connects to P0.15 - SCK on slave board;
- P0.16 - SSEL on master connects to P0.16 - SSEL on slave board (used as GPIO)
- P0.17 - MISO on master connects to P0.17 - MISO on slave board
- P0.18 - MOSI on master connects to P0.18 - MOSI on slave board

Common ground must be connected together between two board.

3.18.2.4 Running mode

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

3.18.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into master board (if run on ROM mode)

Step 3: Build "Slave\spi_slave" example

Step 4: Burn this hex file into slave board.

Step 5: Connect UART0 on master board to COM port on your computer

Step 6: Configure hardware, connect master board and slave board as above instruction

Step 7: Configure serial display as above instruction

Step 8: Run example and observe SPI master transfer result from serial display

- Press '1' to start SPI transfer

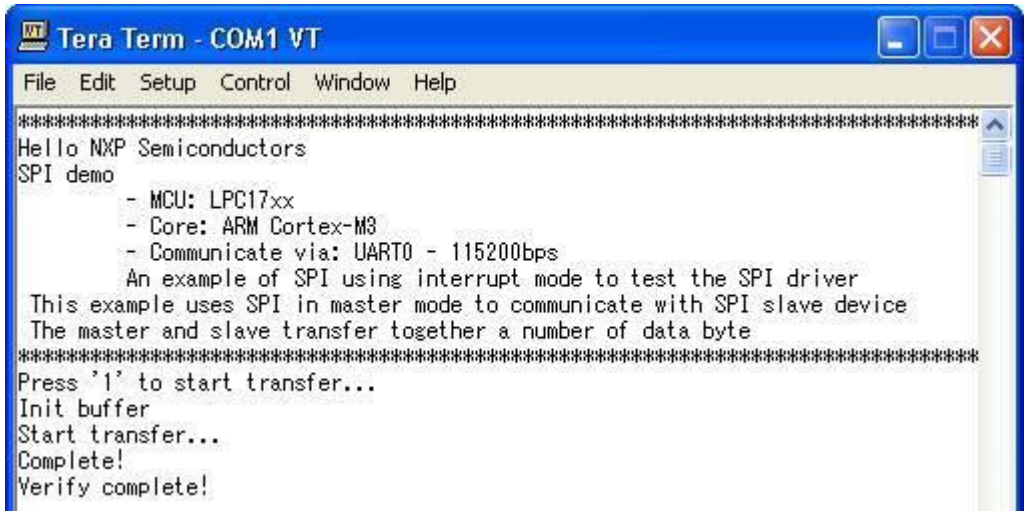

Fig 121. SPI master demo

3.18.3 sc16is750_int

3.18.3.1 Example description

Purpose

This example describes how to use SPI in interrupt mode to communicate with SC16IS750/760 Demo Board.

Process

SPI configuration:

- CPHA = 0: data is sampled on the first clock edge of SCK.
- CPOL = 0: SCK is active high
- Clock rate = 1MHz
- LSBF = 0: SPI data is transferred MSB first
- BITS = 8: 8 bits per transfer
- MSTR = 1: SPI operates in Master mode

First, SPI send commands to reset, config direction SC16IS740 chip in interrupt mode.

Start to use SPI polling mode to handle SC16IS740 board.

On serial display:

- Press 'r' to print menu
- Press '1': send 0x00 value to IOStat register to turn on 8 LEDs on SC16IS740 board.
- Press '2': send 0xFF value to IOStat register to turn off 8 LEDs on SC16IS740 board.

3.18.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

spi_interrupt_test.c: Main program

SC16IS740_750_760_6.pdf: SC16IS740_750_760_6's datasheet file

schematics.sc16is750.demo.board.pdf: SC16IS750 board's schematic file

3.18.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SC16IS750 board

- JP2: 2-3 (SPI)
- JP16: 2-3 (hard reset)
- Remain jumper: OFF

SPI connection

- P0.15 - SCK on eval board connects to SCLK on SC16IS750 board
- P0.16 - SSEL on eval board connects to /CS on SC16IS750 board (used as GPIO)
- P0.17 - MISO on eval board connects to MISO on SC16IS750 board
- P0.18 - MOSI on eval board connects to MOSI on SC16IS750 board

Common power source 3.3V and ground must be connected together between two board.

3.18.3.4 Running mode

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

3.18.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 5: Connect UART0 on board to COM port on your computer

Step 6: Configure hardware, connect this board with SC16IS750 board as above instruction

Step 7: Configure serial display as above instruction

Step 8: Run example and handle SPI transfer via serial display

- Press '1' to turn on 8 LEDs on SC16IS750 board
- Press '2' to turn off 8 LEDs on SC16IS750 board

The screen will be displayed like this:

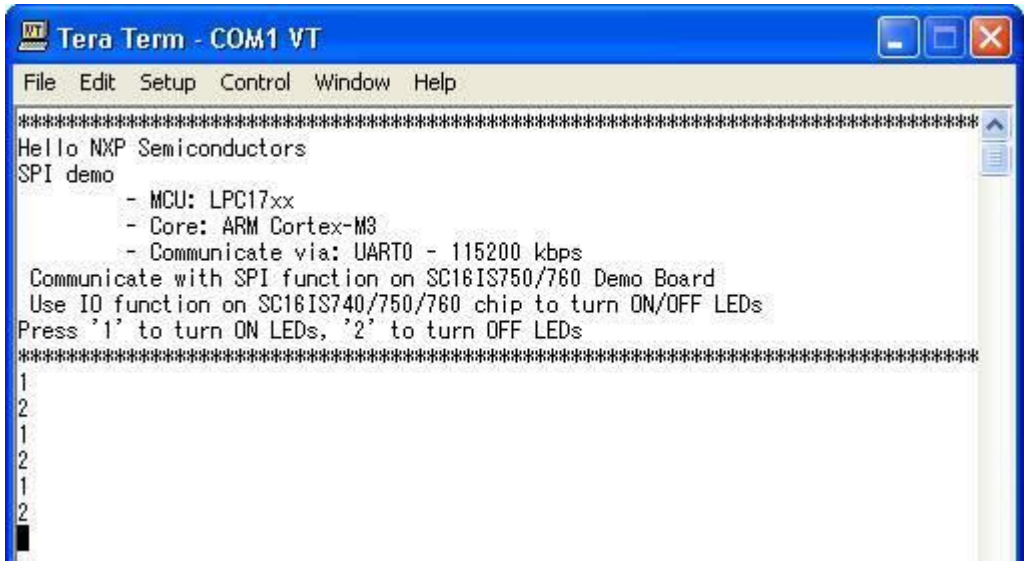

Fig 122. SPI – SC16IS750 interrupt demo

3.18.4 sc16is750_polling

3.18.4.1 Example description

Purpose

This example describes how to use SPI in polling mode to communicate with SC16IS750/760 Demo Board.

Process

SPI configuration:

- CPHA = 0: data is sampled on the first clock edge of SCK.
- CPOL = 0: SCK is active high
- Clock rate = 1MHz

- LSBF = 0: SPI data is transferred MSB first
- BITS = 8: 8 bits per transfer
- MSTR = 1: SPI operates in Master mode

SPI send commands to reset, config and handle communication with SC16IS740 board.

On serial display:

- Press 'r' to print menu
- Press '1': send 0x00 value to IOStat register to turn on 8 LEDs on SC16IS740 board.
- Press '2': send 0xFF value to IOStat register to turn off 8 LEDs on SC16IS740 board.

3.18.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

spi_polling _test.c: Main program

3.18.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SC16IS750 board

- JP2: 2-3 (SPI)
- JP16: 2-3 (hard reset)
- Remain jumper: OFF

SPI connection

- P0.15 - SCK on eval board connects to SCLK on SC16IS750 board
- P0.16 - SSEL on eval board connects to /CS on SC16IS750 board (used as GPIO)
- P0.17 - MISO on eval board connects to MISO on SC16IS750 board
- P0.18 - MOSI on eval board connects to MOSI on SC16IS750 board

Common power source 3.3V and ground must be connected together between two board.

3.18.4.4 Running mode

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

3.18.4.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on board to COM port on your computer

Step 4: Configure hardware, connect this board with SC16IS750 board as above instruction

Step 5: Configure serial display as above instruction

Step 6: Run example and handle SPI transfer via serial display

- Press '1' to turn on 8 LEDs on SC16IS750 board
- Press '2' to turn off 8 LEDs on SC16IS750 board

The screen will be displayed like this:

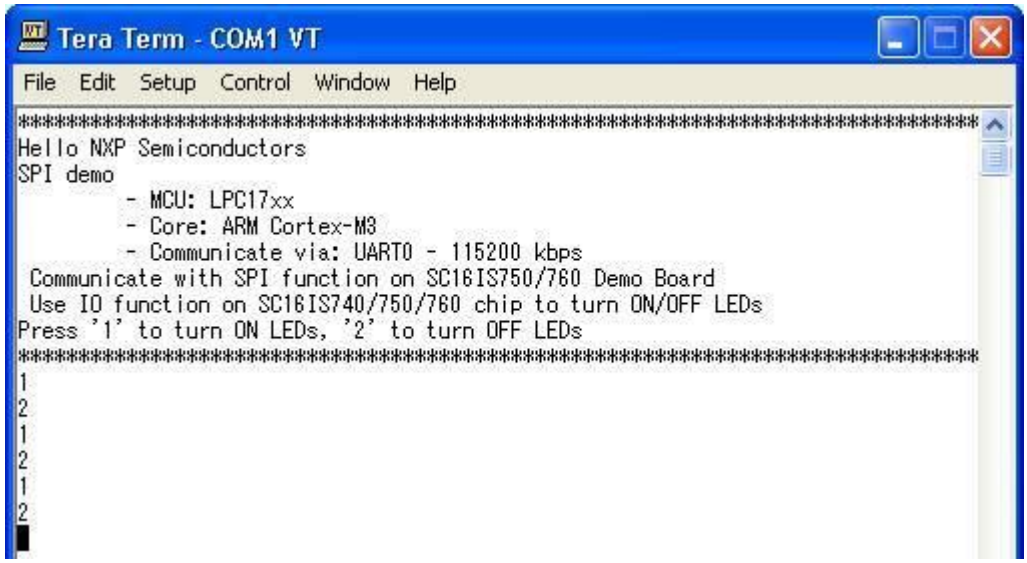

Fig 123. SPI – SC16IS750 polling demo

3.18.5 SDCard

3.18.5.1 Example description

Purpose

This example describes how to use SPI to read SD card's CID register

Process

SPI configuration:

- CPHA = 0: data is sampled on the first clock edge of SCK.
- CPOL = 0: SCK is active high
- Clock rate = 1MHz
- LSBF = 0: SPI data is transferred MSB first
- BITS = 8: 8 bits per transfer
- MSTR = 1: SPI operates in Master mode

1. Look for SD card connected or not, if yes then
2. Configure SD card in SPI mode, then start SD card's internal initialization process.
3. Read SD card's CID register then decode and display out via UART0

3.18.5.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

spi_sdcard.c: Main program

3.18.5.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

3.18.5.4 Running mode

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

3.18.5.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on board to COM port on your computer

Step 4: Run example

- Plug in the SD card.
- Look at the PC terminal screen to see the information of SD card.

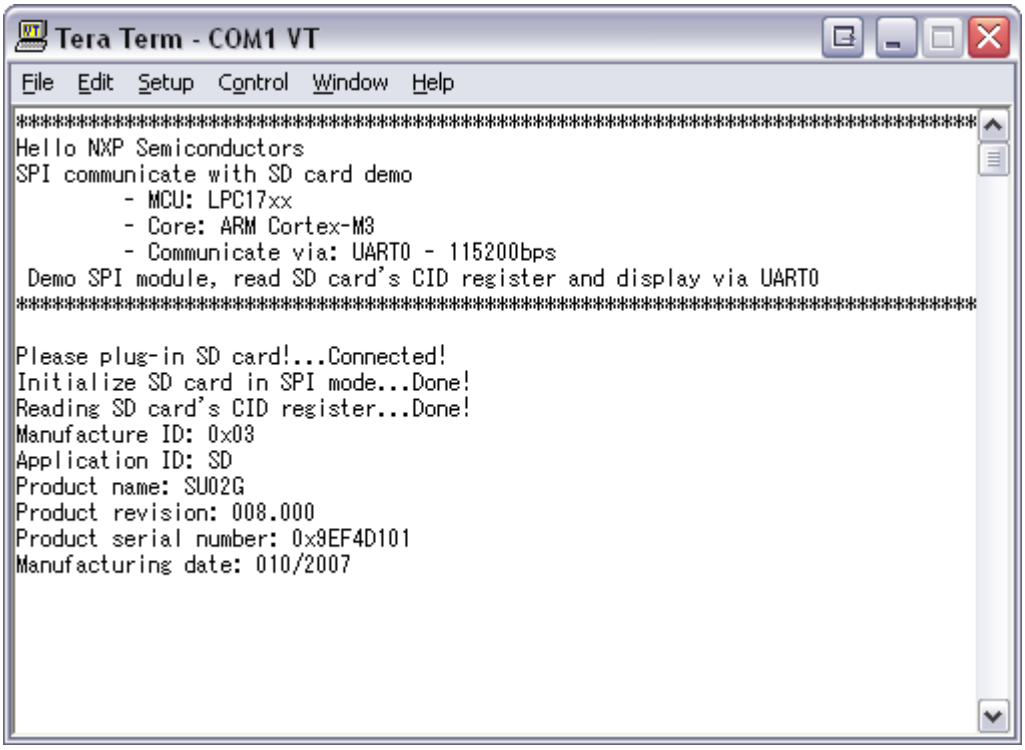

Fig 124. SDCard demo

3.18.6 Slave

3.18.6.1 Example description

Purpose

This example describes how to use SPI as slave to connect with SPI master

Note that: this example will run combinate with master example at folder \Master.

(Pls see "Step to run" part for more information)

Process

SPI configuration:

- CPHA = 1: data is sampled on the second clock edge of SCK.
- CPOL = 1: SCK is active low
- Clock rate = 2MHz
- LSBF = 0: SPI data is transferred MSB first
- BITS = 10: 10 bits per transfer
- MSTR = 1: SPI operates in Slave mode

After initialize buffer, SPI slave will wait to receive/send data from/to master in POLLING mode by calling 'SPI_ReadWrite( )' function.

After transmittion completed, receive and transmit buffer will be compare, if they are not similare, the program will be enter infinite loop and a error notice will be displayed.

3.18.6.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

spi_slave.c: Main program

3.18.6.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SPI connection

- P0.15 - SCK on slave connects to P0.15 - SCK on master board;
- P0.16 - SSEL on slave connects to P0.16 - SSEL on master board (used as GPIO)
- P0.17 - MISO on slave connects to P0.17 - MISO on master board
- P0.18 - MOSI on slave connects to P0.18 - MOSI on master board

Common ground must be connected together between two board.

3.18.6.4 Running mode

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

3.18.6.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into slave board (if run on ROM mode)

Step 3: Build "Slave\spi_slave" example

Step 4: Burn this hex file into master board.

Step 5: Connect UART0 on slave board to COM port on your computer

Step 6: Configure hardware, connect master board and slave board as above instruction

Step 7: Configure serial display as above instruction

Step 8: Run example and observe SPI slave transfer result from serial display

Note: You have to press '1' on master's serial display to start master transmition first

The screen will be displayed like this:

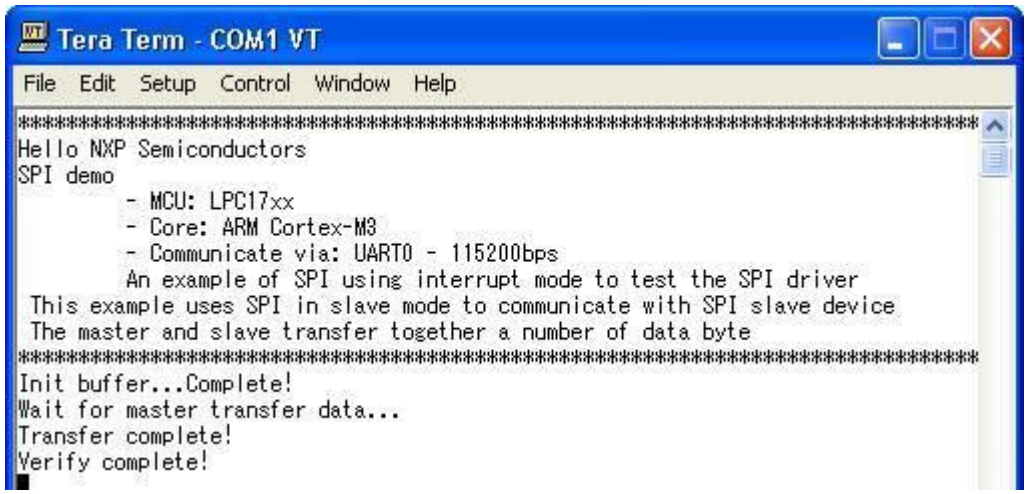

Fig 125. SPI slave demo

3.19 SSP

3.19.1 dma

3.19.1.1 Example description

Purpose

This example describes how to use SSP peripheral with DMA support.

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 uses SSP function in MASTER mode with Loop-back mode (MOSI <-> MISO).

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.

Channel 1 is used to transfer data from SSP peripheral to destination buffer.

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

3.19.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

ssp_dma.c: Main program

3.19.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source

- DBG_EN: ON
- Remain jumper: OFF

SSP connection

- P0.17 - MISO
- P0.18 - MOSI

MOSI must be connected with MISO pin.

3.19.1.4 Running mode

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

3.19.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe SSP transfer result on serial display

The screen will be displayed like this:

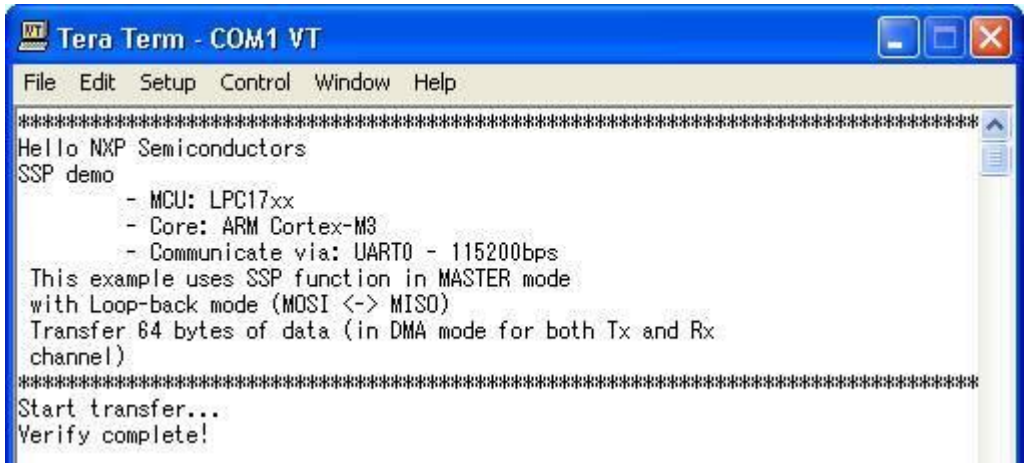

Fig 126. SSP – DMA demo

3.19.2 Master

3.19.2.1 Example description

Purpose

This example describes how to use SSP peripheral as master to connect with SSP slave

Note that: this example will run combinate with slave example at folder \Slave.

(Pls see "Step to run" for more information)

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

After initialize buffer, SPI master will transfer/receive data to/from SSP slave in POLLING mode by calling 'SPI_ReadWrite( )' function.

After transmittion completed, receive and transmit buffer will be compared, if they are not similar, the program will enter infinite loop and a error notice will be displayed.

3.19.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

ssp_master.c: Main program

3.19.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SSP connection

- P0.15 - SCK on master connects to P0.15 - SCK on slave board;
- P0.16 - SSEL on master connects to P0.16 - SSEL on slave board
- P0.17 - MISO on master connects to P0.17 - MISO on slave board
- P0.18 - MOSI on master connects to P0.18 - MOSI on slave board

Common ground must be connected together between two board.

3.19.2.4 Running mode

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

3.19.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into master board (if run on ROM mode)

Step 3: Build "Slave\spi_slave" example

Step 4: Burn this hex file into slave board.

Step 5: Connect UART0 on master board to COM port on your computer

Step 6: Configure hardware, connect master board and slave board as above instruction

Step 7: Configure serial display as above instruction

Step 8: Run example and observe SPI master transfer result from serial display

- Hit reset button on slave board first
- Hit reset button on master board
- At master side: Press '1' to start SSP transfer

The screen will be displayed like this:

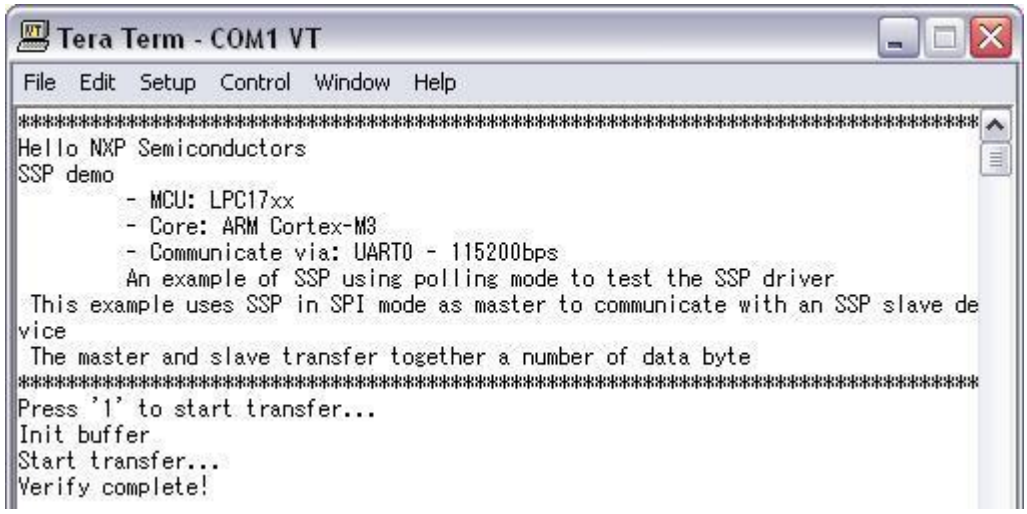

Fig 127. SSP – Master demo

3.19.3 MicroWire

3.19.3.1 Example description

Purpose

This example describes how to use SSP peripheral with MicroWire frame format.

Process

This example uses two SSP peripherals in MicroWire frame format, one is set as master mode and the other is set as slave mode.

SSP master/slave 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 = 0/1: SSP operates in Slave/Master mode
- FRF= 2: MicroWire Frame format

After initialize buffer, SPI master will transfer/receive data to/from SSP slave in POLLING mode.

After transmittion completed, receive and transmit buffer will be compared, if they are not similar, the program will enter infinite loop and a error notice will be displayed.

3.19.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

microwire.c: Main program

3.19.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SSP connection

- P0.15 <-> P0.7 : SCK
- P0.16 <-> P0.6 : SSEL
- P0.17 <-> P0.8 : MISO
- P0.18 <-> P0.9 : MOSI

3.19.3.4 Running mode

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

3.19.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe SSP transfer result on serial display

The screen will be displayed like this:

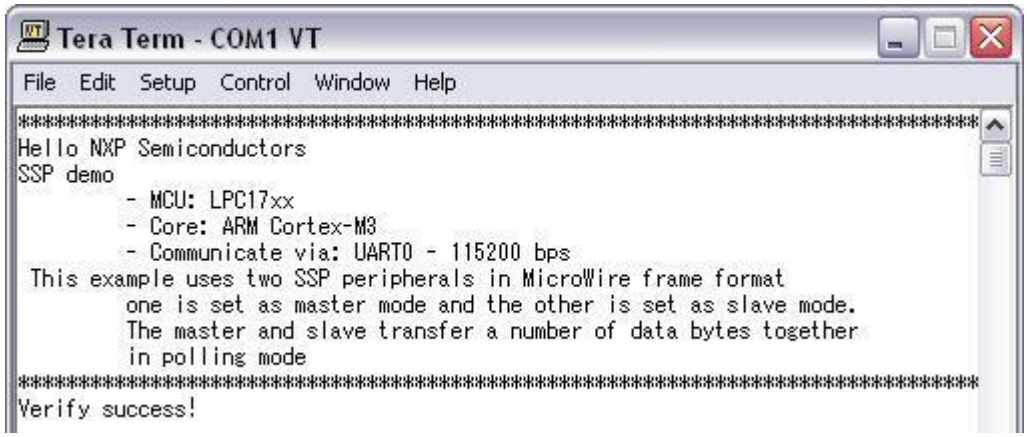

Fig 128. SSP – MicroWire demo

3.19.4 sc16is185_int

3.19.4.1 Example description

Purpose

This example describes how to use SSP in interrupt mode to communicate with SC16IS750/760 Demo Board.

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

First, SSP send commands to reset, config direction SC16IS740 chip in interrupt mode.

Start to use SSP polling mode to handle SC16IS740 board.

On serial display:

- Press 'r' to print menu

- Press '1': send 0x00 value to IOStat register to turn on 8 LEDs on SC16IS740 board.
- Press '2': send 0xFF value to IOStat register to turn off 8 LEDs on SC16IS740 board.

3.19.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

sc16is750_int.c: Main program

3.19.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SC16IS750 board

- JP2: 2-3 (SPI)
- JP16: 2-3 (hard reset)
- Remain jumper: OFF

SSP connection

- P0.15 - SCK on eval board connects to SCLK on SC16IS750 board
- P0.16 - SSEL on eval board connects to /CS on SC16IS750 board (used as GPIO)
- P0.17 - MISO on eval board connects to MISO on SC16IS750 board
- P0.18 - MOSI on eval board connects to MOSI on SC16IS750 board

Common power source 3.3V and ground must be connected together between two board.

3.19.4.4 Running mode

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

3.19.4.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on board to COM port on your computer

Step 4: Configure hardware, connect this board with SC16IS750 board as above instruction

Step 5: Configure serial display as above instruction

Step 6: Run example and handle SSP transfer via serial display

- Press '1' to turn on 8 LEDs on SC16IS750 board
- Press '2' to turn off 8 LEDs on SC16IS750 board

The screen will be displayed like this:

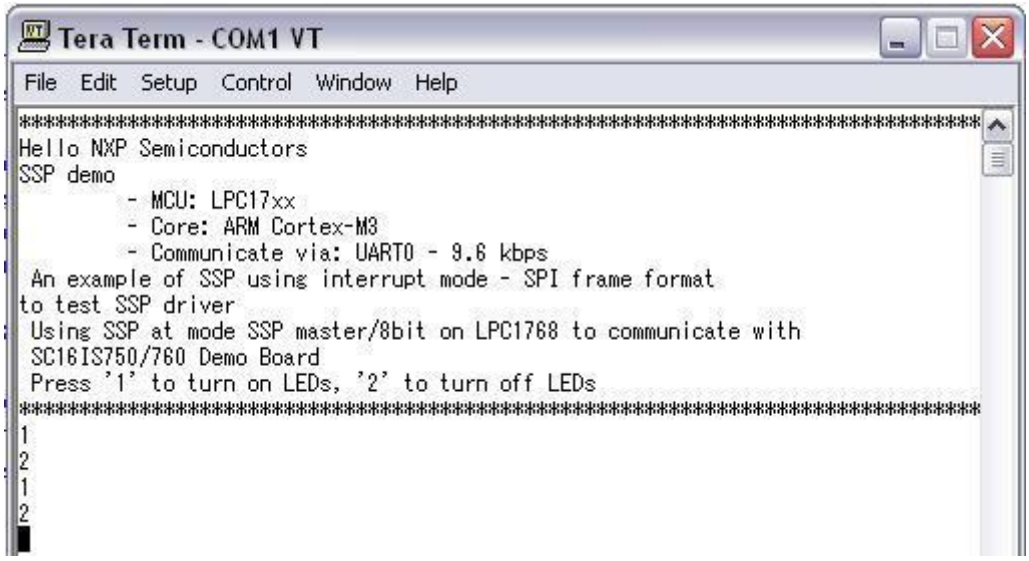

Fig 129. SSP – SC16IS750 Interrupt demo

3.19.5 sc16is750_polling

3.19.5.1 Example description

Purpose

This example describes how to use SSP in polling mode to communicate with SC16IS750/760 Demo Board.

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

SSP send commands to reset, config direction and handle SC16IS740 chip in polling mode.

On serial display:

- Press 'r' to print menu
- Press '1': send 0x00 value to IOStat register to turn on 8 LEDs on SC16IS740 board.
- Press '2': send 0xFF value to IOStat register to turn off 8 LEDs on SC16IS740 board.

3.19.5.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

sc16is750_polling.c: Main program

3.19.5.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SC16IS750 board

- JP2: 2-3 (SPI)
- JP16: 2-3 (hard reset)
- Remain jumper: OFF

SSP connection

- P0.15 - SCK on eval board connects to SCLK on SC16IS750 board
- P0.16 - SSEL on eval board connects to /CS on SC16IS750 board (used as GPIO)
- P0.17 - MISO on eval board connects to MISO on SC16IS750 board
- P0.18 - MOSI on eval board connects to MOSI on SC16IS750 board

Common power source 3.3V and ground must be connected together between two board.

3.19.5.4 Running mode

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

3.19.5.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on board to COM port on your computer

Step 4: Configure hardware, connect this board with SC16IS750 board as above instruction

Step 5: Configure serial display as above instruction

Step 6: Run example and handle SSP transfer via serial display

- Press '1' to turn on 8 LEDs on SC16IS750 board
- Press '2' to turn off 8 LEDs on SC16IS750 board

The screen will be displayed like this:

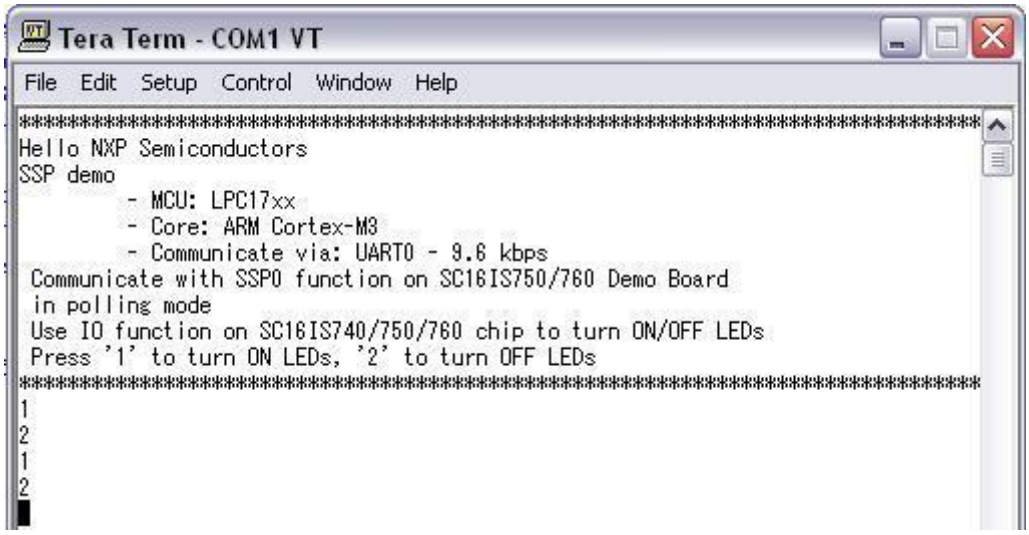

Fig 130. SSP –SC16IS750 Polling demo

3.19.6 Slave

3.19.6.1 Example description

Purpose

This example describes how to use SSP peripheral as slave to connect with SSP master

Note that: this example will run combinate with master example at folder \Master.

(Pls see "Step to run" part for more information)

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 Slave mode
- FRF= 0: SPI Frame format

After initialize buffer, SPI slave will receive/transfer data from/to SSP slave in POLLING mode by calling 'SPI_ReadWrite( )' function.

After transmittion completed, receive and transmit buffer will be compared, if they are not similar, the program will enter infinite loop and a error notice will be displayed.

3.19.6.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

ssp_slave.c: Main program

3.19.6.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SSP connection

- P0.15 - SCK on master connects to P0.15 - SCK on slave board;
- P0.16 - SSEL on master connects to P0.16 - SSEL on slave board
- P0.17 - MISO on master connects to P0.17 - MISO on slave board
- P0.18 - MOSI on master connects to P0.18 - MOSI on slave board

Common ground must be connected together between two board.

3.19.6.4 Running mode

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

3.19.6.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into master board (if run on ROM mode)

Step 3: Build "Master\spi_master" example

Step 4: Burn this hex file into master board.

Step 5: Connect UART0 on slave board to COM port on your computer

Step 6: Configure hardware, connect master board and slave board as above instruction

Step 7: Configure serial display as above instruction

Step 8: Run example and observe SPI slave transfer result from serial display

Note that: Press '1' to start SSP operation on master board first.

The screen will displayed like this:

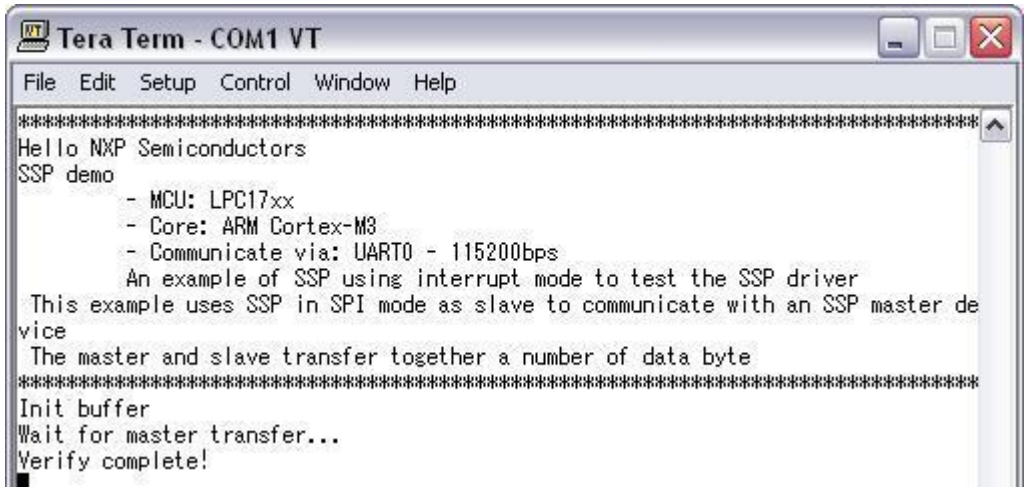

Fig 131. SSP – Slave demo

3.19.7 TI

3.19.7.1 Example description

Purpose

This example describes how to use SSP peripheral with TI frame format.

Process

This example uses two SSP peripherals in TI frame format, one is set as master mode and the other is set as slave mode.

SSP master/slave 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 = 0/1: SSP operates in Slave/Master mode
- FRF= 1: MicroWire Frame format

After initialize buffer, SPI master will transfer/receive data to/from SSP slave in INTERRUPT mode.

After transmittion completed, receive and transmit buffer will be compared, if they are not similar, the program will enter infinite loop and a error notice will be displayed.

3.19.7.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

ti_test.c: Main program

3.19.7.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

SSP connection

- P0.15 <-> P0.7 : SCK
- P0.16 <-> P0.6 : SSEL
- P0.17 <-> P0.8 : MISO
- P0.18 <-> P0.9 : MOSI

3.19.7.4 Running mode

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

3.19.7.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, observe SSP transfer result on serial display

The screen will be displayed like this:

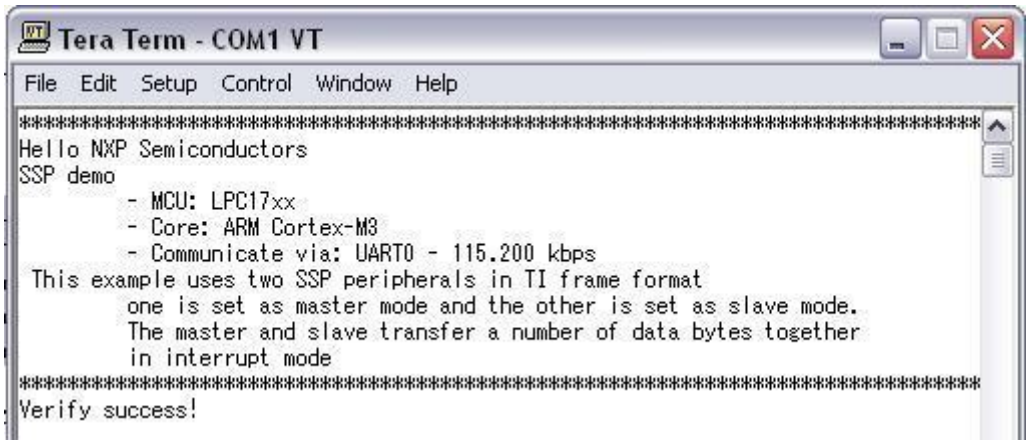

Fig 132. SSP – TI demo

3.20 SysTick

3.20.1 10ms_base

3.20.1.1 Example description

Purpose

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

Process

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

In this case, CPU clock = cclk = 100MHz

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 P0.0 pin. Use oscilloscope to observe signal on P0.0 and measure time between falling and rising edge, it would be: 10ms and signal frequency would be 100Hz.

3.20.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

10ms_base.c: Main program

3.20.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.20.1.4 Running mode

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

3.20.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware as above instruction

Step 4: Run example, observe System Tick operation via P0.0 signal

P0.0 signal displays on oscilloscope like this:

Measure time between two edge: 10ms

Signal frequency: 100Hz

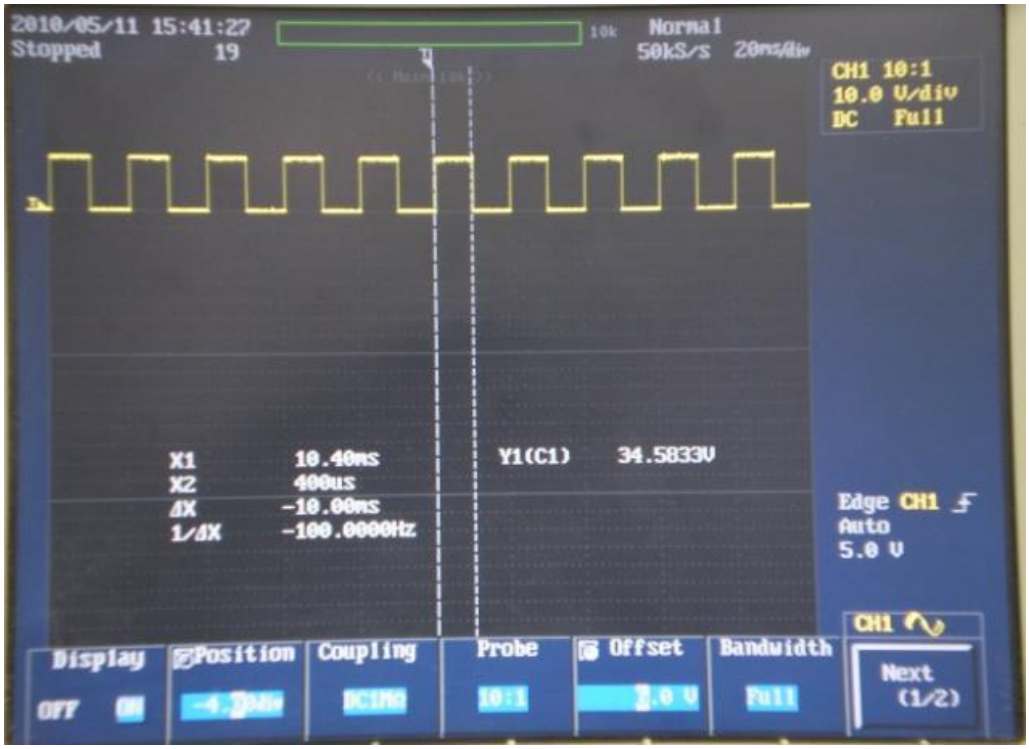

Fig 133. SysTick welcome screen

3.20.2 STCLK

3.20.2.1 Example description

Purpose

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

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 P0.0 pin.

Use oscilloscope to observe signal on P0.0 and measure time between falling and rising edge, it would be: 10ms.

3.20.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

systick_stclk.c: Main program

3.20.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

STCLK connection

- STCLK(P3.26) connects with MAT0.0(P1.28)

3.20.2.4 Running mode

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

3.20.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware as above instruction . Sure that P1.28 and P3.26 connected with each other

Step 4: Run example, observe System Tick operation via P0.0 signal

Fig 134. P0.0 on oscilloscope

3.21 TIMER

3.21.1 Capture

3.21.1.1 Example description

Purpose

This example describes how to use Capture Timer function.

Process

We use Timer 0 to take a snapshot of the timer value when an input signal on CAP0.0 (pin P1.26) transitions.

Timer configuration:

- Prescaler in microsecond value
- prescaler 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

Changing connect P1.26 with GND and VCC whenever want to capture time.

3.21.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

timer_capture.c: Main program

3.21.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.21.1.4 Running mode

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

3.21.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe timer capture value on serial display

Changing connect P1.26 with GND and VCC whenever want to capture time.

The screen will be displayed like this:

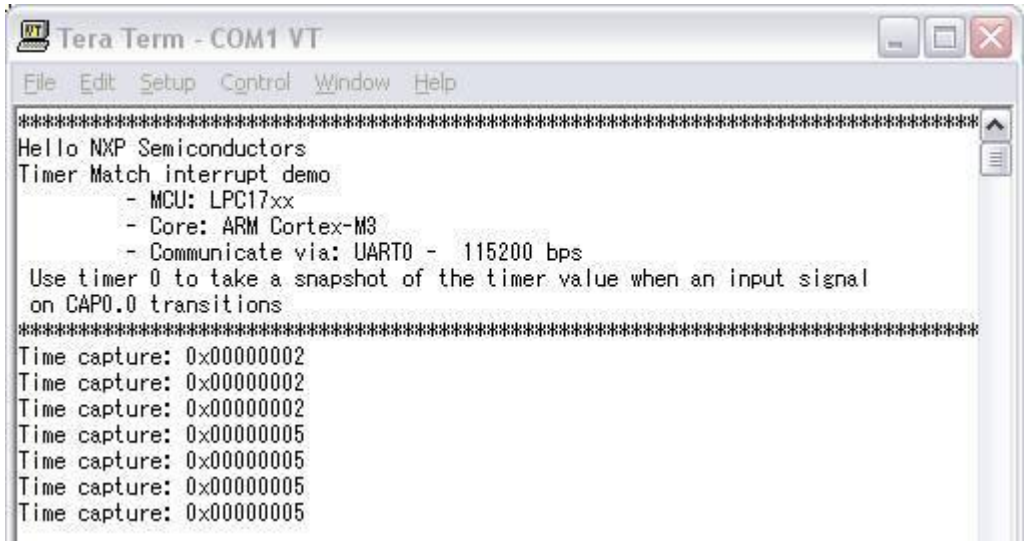

Fig 135. Time captured

3.21.2 FreqMeasure

3.21.2.1 Example description

Purpose

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

Process

1) Initialize UART0 and display information menu.

2) Ask user to input frequency for a test signal.

Configure P1.26 as CAP0.0, P0.6 as MAT2.0.

3) Configure TIMER2 as follow:

- Prescale register = 1 us.
- Match register = 1 / ((frequency)*1us*2)

= 500000 / freuqency

- No interrupt, no stop but reset timer counter on match.

  Configure TIMER0 as follow:

- Prescale register = 1 us.
- Capture register: channel 0, capture on rising edge, generate interrupt on capture.

4) Configure TIMER0 ISR:

- Prepare 5 interrupt times, in which timer counter and prescale are reset, for stable.
- After this 5 interrupt times, get the capture value, raise done flag.

5) Start TIMER0 and TIMER2.

Wait for done flag, then calculate and print out the measure frequency by:

one cycle = capture value * 1us (as TIMER0 configuration).

signal frequency = 1/one cycle = 1,000,000 / capture value.

Press ‘c’ if we intend to test with other frequencies. TIMER0 and TIMER2 are de-initialized.

3.21.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

freqmeasure.c: Main program

3.21.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.21.2.4 Running mode

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

3.21.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example

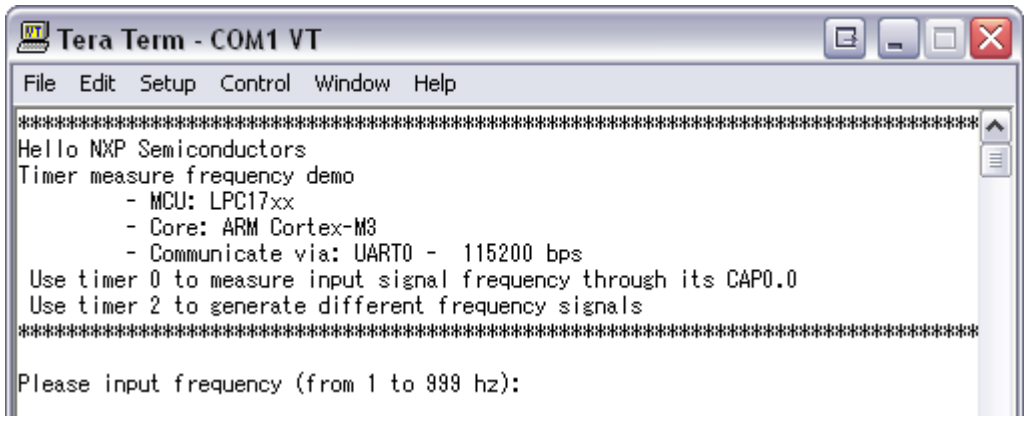

Fig 136. Timer measure frequency demo

Type input frequency, such as:300

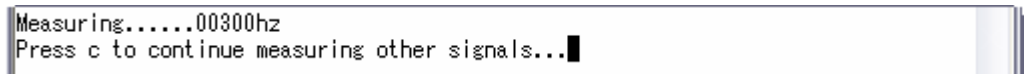

Fig 137. Measure input signal frequency

Program will be measure input signal and print into screen.

Press ‘c’ to continue measuring other signal

You can use oscilloscope to observe input signal frequency.

3.21.3 Gen_Diff_Delay

3.21.3.1 Example description

Purpose

This example describes how to use Timer Match to generate different delay signals.

Process

1) Initialize UART0 and display information menu.

2) Ask user to input delay time for T1 and T2 (unit in ms)

Configure P1.28 as MAT0.0

3) Configure TIMER0:

- Prescale register = 100 us.
- Match register = T1/100us = T1 ms/ 100us = T1*10
- Interrupt, no stop, no reset timer counter on match.

4) Timer0 ISR:

- Stop timer.
- Reset and synchronize timer and prescal counter.
- Update new timer match value.
- Start timer.

5) Start TIMER0.

Press ESC if we intend to test with other frequencies.

TIMER0 and TIMER0 interrupt service are de-initialized.

3.21.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

gen_diff_delay.c: Main program

3.21.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.21.3.4 Running mode

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

3.21.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe MAT0.0 waveform by oscilloscope.

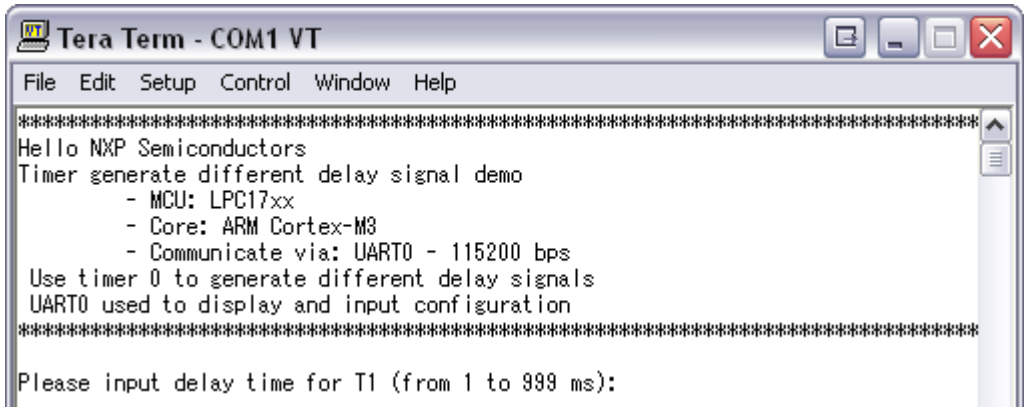

Fig 138. Delay signal demo

Type delay time for T1, such as: 050

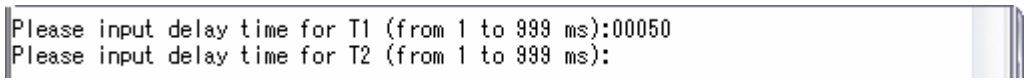

Fig 139. Delay time for T1

Type delay time for T2, such as: 025

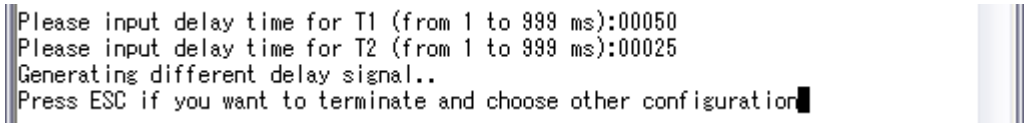

Fig 140. Delay time for T1

You can observe output signal via oscilloscope via MAT0.0 (P1.28) and see the LED P1.28 toogle according to output signal frequency.

3.21.4 Gen_Diff_Freqs

3.21.4.1 Example description

Purpose

This example describes how to use Timer Match to generate 2 different frequency signals.

Process

1) Initialize UART0 and display information menu.

2) Ask user to input frequency for channel1 (MAT0.0) and channel2 (MAT2.0)

Configure P1.28 as MAT0.0, P0.6 as MAT2.0.

3) Configure TIMER0 and TIMER2 as follow:

- Prescale register = 1 us.
- Match register = 1 / ((channel1 or channel2 frequency)*100us*2)

  = 500000 / channel1 or channel2 freuqency
- No interrupt, no stop but reset timer counter on match.

4) Start TIMER0 and TIMER2.

Press ESC if we intend to test with other frequencies.

TIMER0 and TIMER2 are de-initialized.

3.21.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

gen_diff_freqs.c: Main program

3.21.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.21.4.4 Running mode

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

3.21.4.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe MAT0.0 and MAT2.0 waveform by oscilloscope.

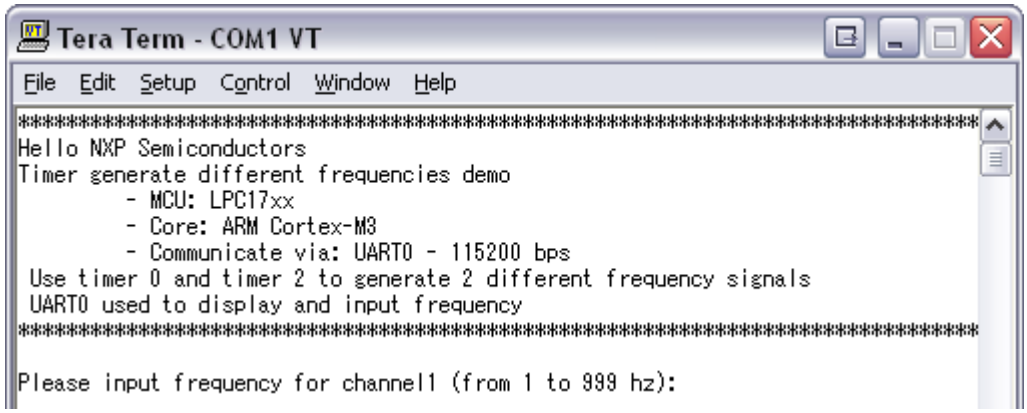

Fig 141. Generate signal frequencies demo

Type input frequency for channel1, and then for channel 2

```
Please input frequency for channel1 (from 1 to 999 hz):00060
Please input frequency for channel2 (from 1 to 999 hz):00090
Generating two different frequency signals..
Press ESC if you want to terminate and choose other frequencies
```

Fig 142. Type input frequency for to channels

Two signals will be generate via pin MAT0.0 (P1.28) and MAT2.0 (P0.6).

Using oscilloscope to observe two signal frequencies.

Press ‘ESC’ to terminate and choose other frequencies.

```
Press ESC if you want to terminate and choose other frequencies
Please input frequency for channel1 (from 1 to 999 hz):
```

Fig 143. Press ‘ESC’ for new signal generation

3.21.5 Interrupt_Match

3.21.5.1 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 configuration:

- Timer channel: 0
- Prescaler in microsecond value

- prescaler value = 100us

Match configuration:

- Use channel 0, MR0
- Match value = 10000

  Because timer tick = prescaler = 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 MAT0.0 will be toggled at frequency = 1Hz
- Generate match interrupt

Whenever MR0 matches the value in TC register, match interrupt is occurs, TIMER0 will invoke interrupt service routine to handle interrupt, in this case, it will print a notice sentence into serial display. MAT0.0(P1.28) toggle and timer will be reseted.

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

3.21.5.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

timer_int_match.c: Main program

3.21.5.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.21.5.4 Running mode

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

3.21.5.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe match interrupt notice on serial display

- On MCB1700 board: MAT0.0(P1.28) connect with LED P1.28, so you can observe MAT0.0 toggle via LED P1.28 blinky process.
- On IAR board: MAT0.0 signal can be observed by oscilloscope or wire P1.28 pin with LED1(P1.25)

The screen will be displayed like this:

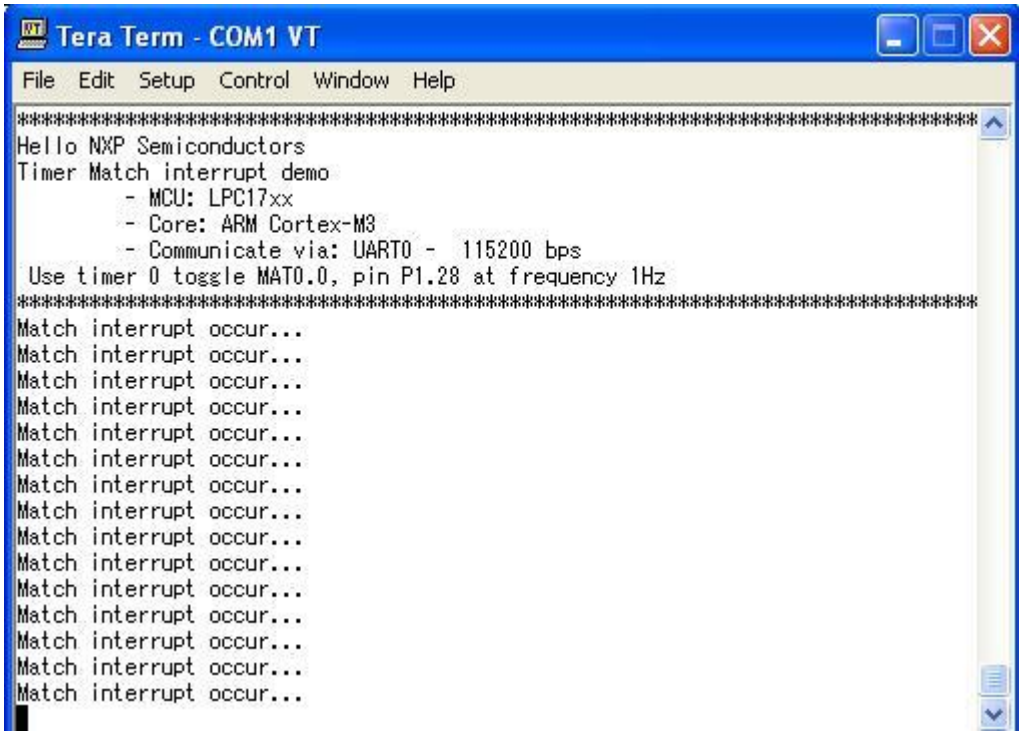

Fig 144. Time interrupt – Match demo

3.21.6 Polling_Match

3.21.6.1 Example description

Purpose

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

Process

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

Timer configuration:

- Timer channel: 0
- Prescaler in microsecond value
- prescaler value = 100us

Match configuration:

- Use channel 0, MR0
- Match value = 1000

  Because timer tick = prescaler = 100us

  So match time = 100 * 1000 = 100000us = 100ms
- Timer reset after match
- Not stop MR0 when match
- Toggle MR0.0 when match

  Because match time = 100ms

  So MAT0.0 will be toggled at frequency = 10Hz
- Generate match interrupt

Using 'TIM_GetIntStatus( )' to wait until interrupt flag for MR0 is set.

When MR0 matches the value in the TC, interrupt flag MR0I is set, the program run out of loop function and notice match interrupt occur, MAT0.0 pin (P1.28) is toggled.

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

3.21.6.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

timer_poll_match.c: Main program

3.21.6.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON

- Remain jumper: OFF

3.21.6.4 Running mode

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

3.21.6.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe match interrupt notice on serial display

- On MCB1700 board: MAT0.0(P1.28) connect with LED P1.28, so you can observe MAT0.0 toggle via LED P1.28 blinky process.
- On IAR board: MAT0.0 signal can be observed by oscilloscope or wire P1.28 pin with LED1(P1.25)

The screen will be displayed like this:

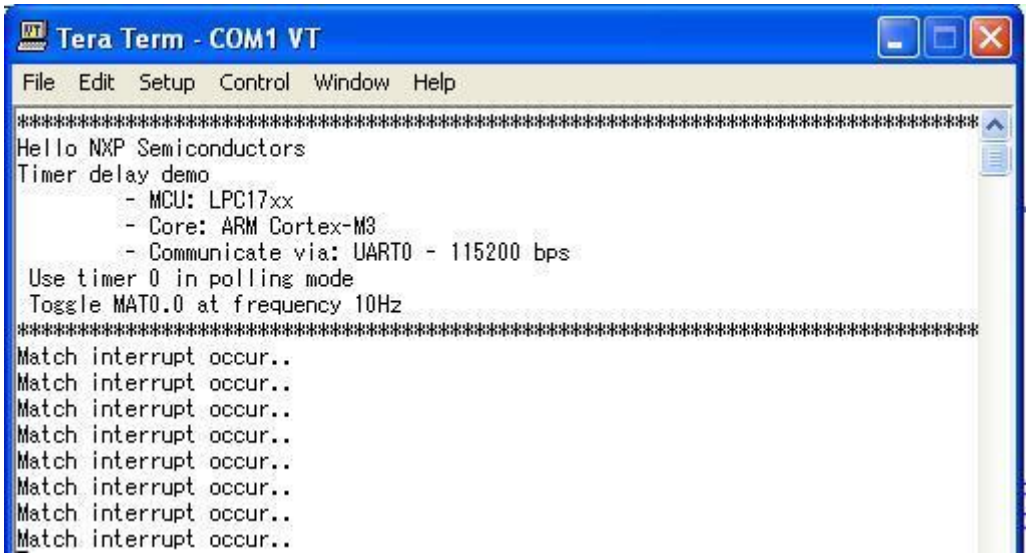

Fig 145. Time – Match polling demo

3.21.7 PWMSignal

3.21.7.1 Example description

Purpose

This example describes how to use TIMERs to generate PWM signals with different duty cycles.

Process

In this case, use 4 TIMERs with match channel 0 to generate 4 PWM signal with 4 duty cycles:

- TIME0: Match channel 0 (MAT0.0 - P1.28)
  - HIGH state: 100 (100*100 = 10000us = 10ms)
  - LOW state: 700 (70ms)

  So duty cycle = 100/800 = 12.5%
- TIME1: Match channel 0 (MAT1.0 - P1.22)
  - HIGH state: 200 (20ms)
  - LOW state: 600 (60ms)

  So duty cycle = 200/800 = 25%
- TIMER2: Match channel 0 (MAT2.0: P0.6)
  - HIGH state: 300 (30ms)
  - LOW state: 500 (50ms)

  So duty cycle = (300/800) = 37.5%
- TIMER3: Match channel 0 (MAT3.0 - P0.10)
  - HIGH state: 800 (80ms)
  - LOW state: 800 (80ms)

  So duty cycle = (800/1600) = 50%

Using oscilloscope to observe these signals

3.21.7.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

pwm_signal.c: Main program

3.21.7.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.21.7.4 Running mode

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

3.21.7.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, see the welcome screen on serial display and observe 4 signals on oscilloscope:

- P1.28: duty cycle = 12.5%
- P1.22: duty cycle = 25%
- P0.6: duty cycle = 37.5%
- P0.10: duty cycle = 50%

The welcome screen will be displayed like that:

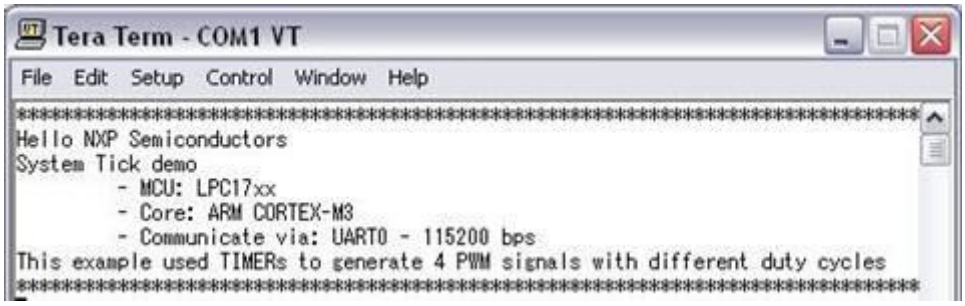

Fig 146. TIMER PWMSignal demo

4 signals will be like that:

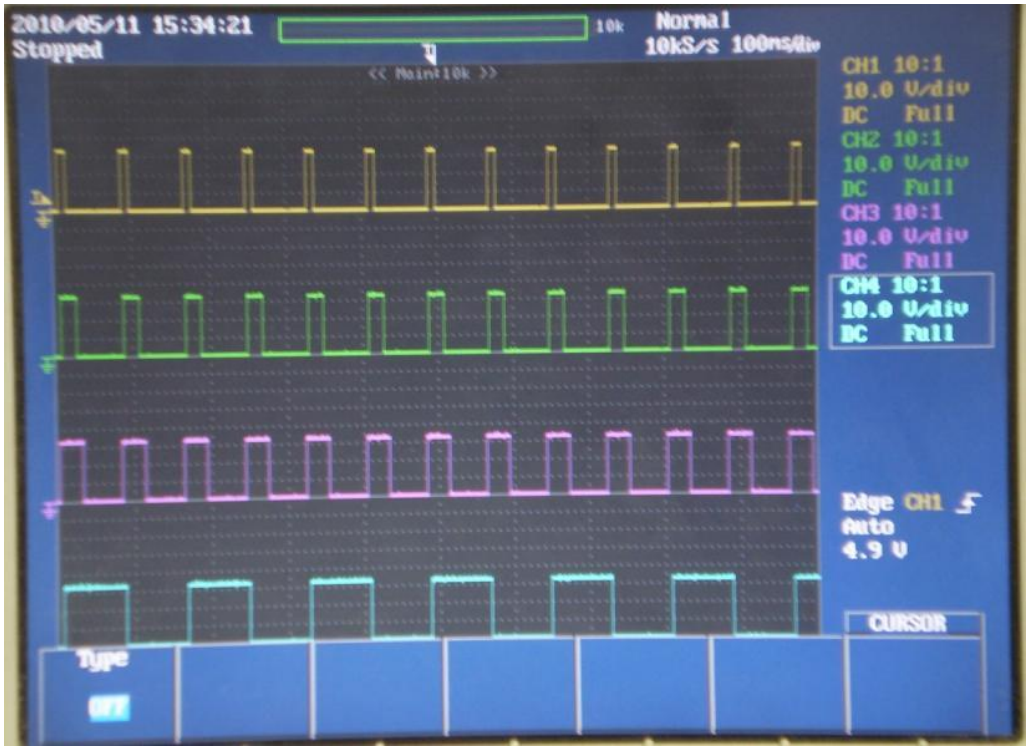

Fig 147. PWM signals

3.22 UART

3.22.1 AutoBaud

3.22.1.1 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, first, 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.

3.22.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

uart_autobaud_test.c: Main program

3.22.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.22.1.4 Running mode

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

3.22.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example:

- Chose any baudrate
- Press 'A' or 'a' to synchronize
- Press any key and see echo on serial display

The screen will be displayed like this:

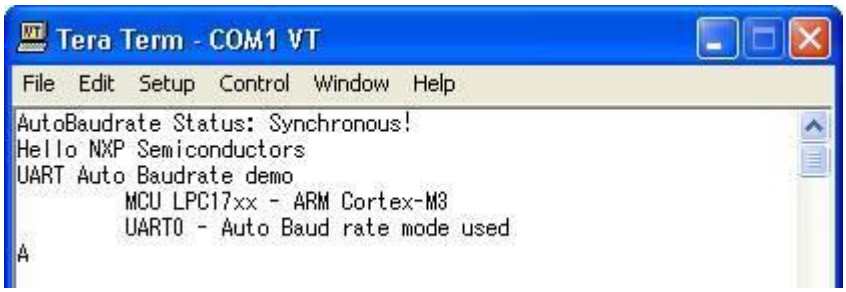

Fig 148. UART auto baudrate demo

3.22.2 DMA

3.22.2.1 Example description

Purpose

This example describes how to use UART in DMA mode

Process

UART0 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)

3.22.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

uart_dma_test.c: Main program

3.22.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.22.2.4 Running mode

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

3.22.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, using Serial display to monitor the message and to send a charater to UART

The screen will be displayed like this:

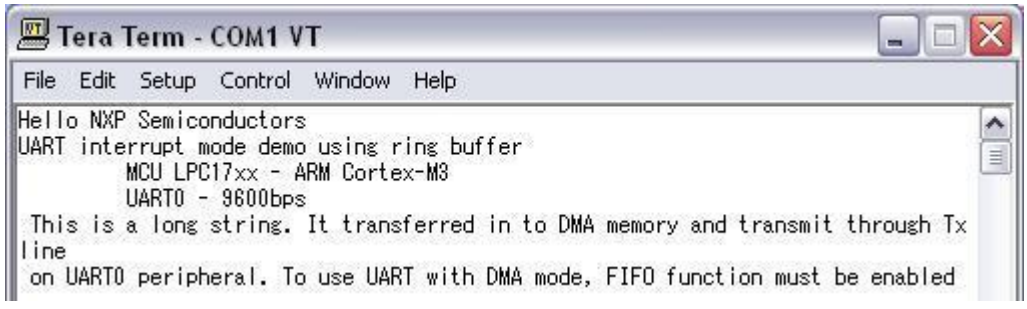

Fig 149. UART DMA demo

3.22.3 HWFlowControl

3.22.3.1 Example description

Purpose

This example describes how to use UART in hardware flow control mode

Process

UART1 configuration:

- 9600bps
- 8 data bit
- No parity
- 1 stop bit
- RTS/CTS flow control
- Receive and transmit enable

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.

3.22.3.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

uart_hw_flow_control.c: Main program

3.22.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- LED: OFF
- Remain jumper: OFF

UART connection

Because COM1 port just has TXD1, RXD1, GND, so we need to make external RS232 board to connect TXD1, RXD1, RTS1, CTS1, GND with PC's COM port.

This external board is not difficult, the schematic can be found in the 232 IC datasheet, below is just some notes:

- DB9 connector:
  - pin 2: connect to 232's T2OUT.
  - pin 3: connect to 232's R2IN
  - pin 7: connect to 232's R1IN //RTS

- pin 8: connect to 232's T1OUT. //CTS

- UART1 pin:
  - P2.0 TXD1: connect to 232's T2IN
  - P2.1 RXD1: connect to 232's R2OUT
  - P2.2 CTS1: connect to 232's R1OUT
  - P2.7 RTS1: connect to 232's T1IN

3.22.3.4 Running mode

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

3.22.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect external board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example

Open PC terminal application, configure

- 9600bps,
- 8 databit
- no parity
- 1 stop bit
- hardware control

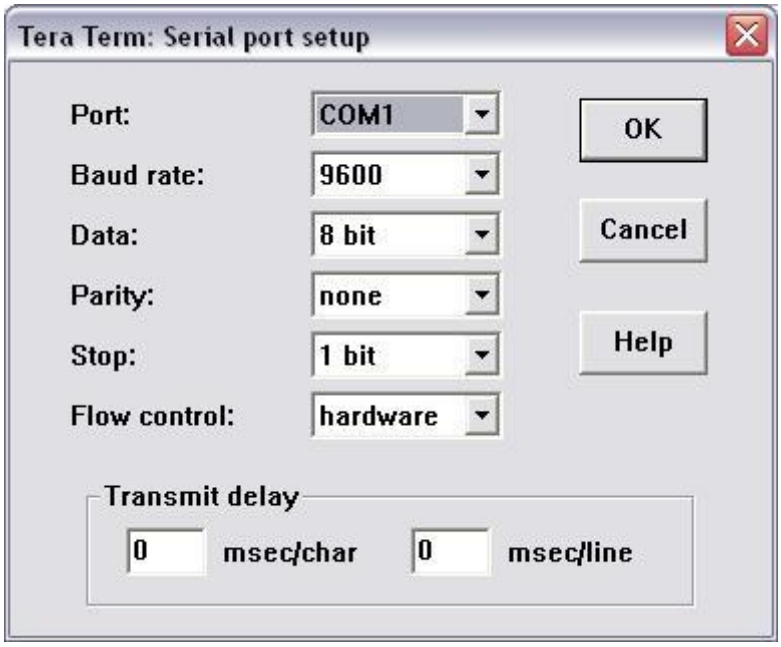

Fig 150. UART setting

Type in some characters to see they displayed back on terminal screen.

The screen will be displayed like this:

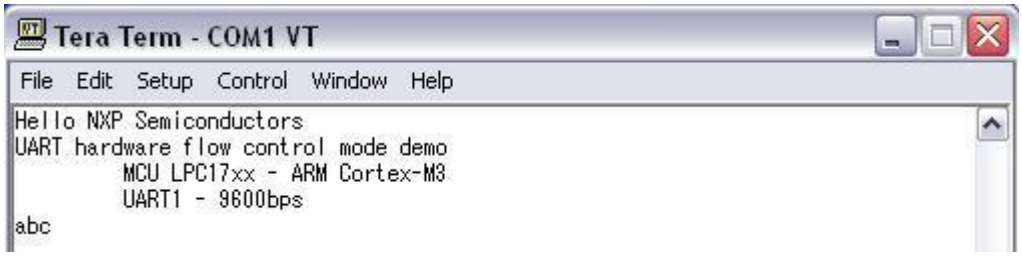

Fig 151. UART HWFlowControl demo

3.22.4 Interrupt

3.22.4.1 Example description

Purpose

This example describes how to use UART in interrupt mode

Process

UART0 configuration:

- 9600bps
- 8 data bit
- No parity
- 1 stop bit
- No flow control
- Enable UART interrupt

UART0 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.

3.22.4.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

uart_interrupt_test.c: Main program

3.22.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.22.4.4 Running mode

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

3.22.4.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, using Serial display to monitor the message and to send a charater to UART

The screen will be displayed like this:

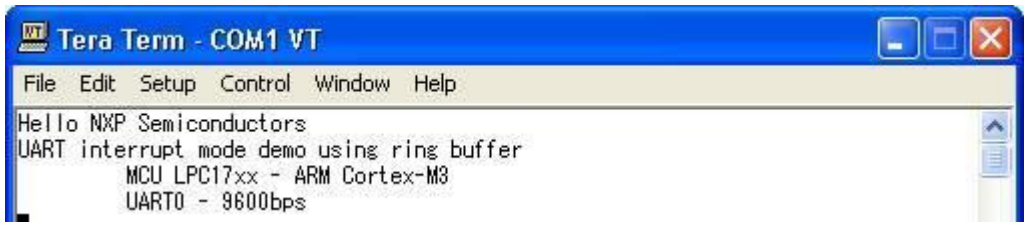

Fig 152. UART Interrupt demo

3.22.5 IrDA - Transmit

3.22.5.1 Example description

Purpose

This example, together with receive example, describes how to use UART in IrDA mode

Process

UART0/UART1 configuration:

- 9600bps
- 8 data bit
- No parity
- 1 stop bit
- No flow control
- Receive and transmit enable

UART3 configuration:

- 9600bps
- 8 data bit
- No parity
- 1 stop bit
- No flow control
- Enable IrDA mode
- Transmit enable

UART will print welcome screen first, then:

- Press ESC to exit.
- Press 'r' to print welcome screen menu again.
- Press two hex digits (each digit is from 0..F/f, ex: 1f means 0x1F) to form a byte value for UART3 TXD to transmit.
- Press other keys will take no effect.

Note: If using this example to print with UART1, pls add conversion type (LPC_UART_TypeDef *)LPC_UART1 because UART1 has different structure type

Ex: UART_Send((LPC_UART_TypeDef *)LPC_UART1, menu1, sizeof(menu1), BLOCKING);

3.22.5.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

uart_irda_transmit.c: Main program

3.22.5.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON

- AD0.2: OFF
- Remain jumper: OFF

Add in hardware

(1 infrared transmit led, 1 C1815 transistor, 1 33R, 1 150R)

- An infrared transmit led with its anode connected to 3.3V, cathode connected to 33R resistor. The other end of this resistor connects to C1815 collector. C1815's emmitor connects to GND.
- Use a wire to connect P0.25, AD0.2 jumper, with 150R resistor, the other end of this resistor connectd to C1815 base.

3.22.5.4 Running mode

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

3.22.5.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Also burn the receive example hex file to other board.

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run both examples.

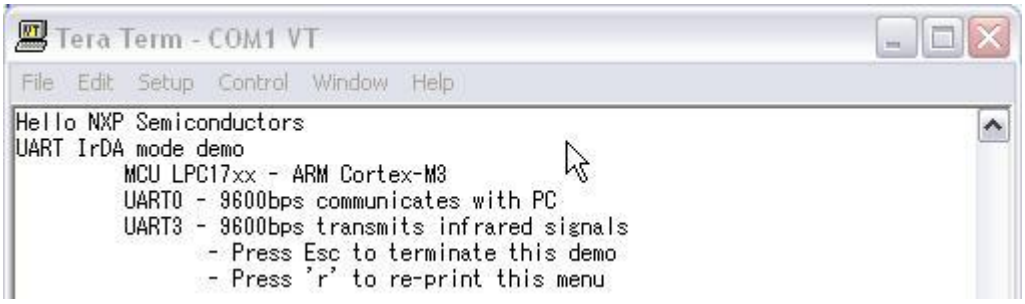

Fig 153. UART IrDA welcome screen

Point the transmit infrared led to the receive infrared led.

From PC's terminal application, type 2 hex digits to form a byte to transmit.

Observe the 8 leds bank of the receive board to compare with the transmit value.

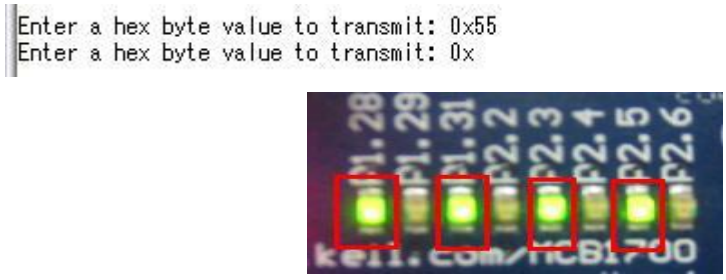

Fig 154. Transmit board send 0x55 value and LEDs display on received board

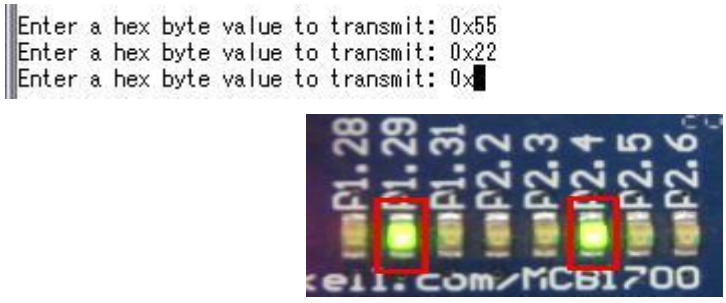

Fig 155. Continue send 0x22 value and LEDs display on received board

3.22.6 IrDA – Receive

3.22.6.1 Example description

Purpose

This example, together with the transmit example, describes how to use UART in IrDA mode

Process

- UART0/ UART1 configuration
  - 9600bps
  - 8 data bit
  - No parity
  - 1 stop bit
  - No flow control
  - Receive and transmit enable
- UART3 configuration
  - 9600bps

- 8 data bit
- No parity
- 1 stop bit
- No flow control
- Enable IrDA mode

GPIO P2.2-P2.6, P1.28, P1.29, P1.31 are configured as output for display the received byte.

UART will print welcome screen first, then UART3 keep reading the income irda signal and output to 8 leds bank the received value.

Note: If using this example to print with UART1, pls add conversion type (LPC_UART_TypeDef *)LPC_UART1 because UART1 has different structure type

Ex: UART_Send((LPC_UART_TypeDef *)LPC_UART1, menu1, sizeof(menu1), BLOCKING);

3.22.6.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

uart_irda_receive.c: Main program

3.22.6.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- AD0.2: OFF
- Remain jumper: OFF

Add in a simple hardware

(1 infrared receive led, 1 150R resistor)

- An infrared receiver led which has its anode connected to 3.3V, cathode connected to a 150R resistor. The other end of this resistor connects to GND.
- Connect a wire between P0.26, SPK jumper, and this infrared receiver led's cathode.

3.22.6.4 Running mode

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

3.22.6.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Also burn the transmit example hex file to the other board.

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run both examples

Point the infrared transmit led to the infrared received led66

Observe the 8 leds bank and compare with the value send out by the transmit example.

(Read “IrDA – Transmit” example for more information)

3.22.7 Polling

3.22.7.1 Example description

Purpose

This example describes how to use UART in polling mode

Process

UART0 configuration:

- 9600bps
- 8 data bit
- No parity
- 1 stop bit
- No flow control
- Enable UART interrupt

UART0 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.

3.22.7.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

uart_polling _test.c: Main program

3.22.7.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON

- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

3.22.7.4 Running mode

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

3.22.7.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example, using Serial display to monitor the message and to send a charater to UART

The screen will be displayed like this:

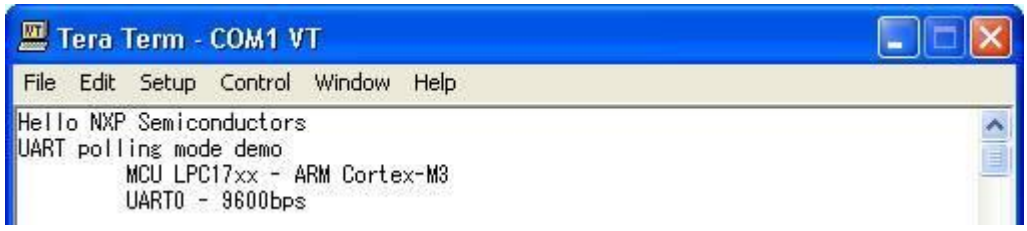

Fig 156. UART Polling demo

3.22.8 RS485_Master

3.22.8.1 Example description

Purpose

This example describes how to use RS485 functionality on UART1 of LPC1768 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).

3.22.8.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

rs485_master.c: Main program

3.22.8.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

RS485 connection

Pls see the 'Transceiver_Master.png' in this directory for wiring information.

3.22.8.4 Running mode

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

3.22.8.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Connect RS485 as above instruction (using ADM485)

Step 5: Run example and using Serial display to monitor the communication between master and slave.

The screen will be displayed like this:

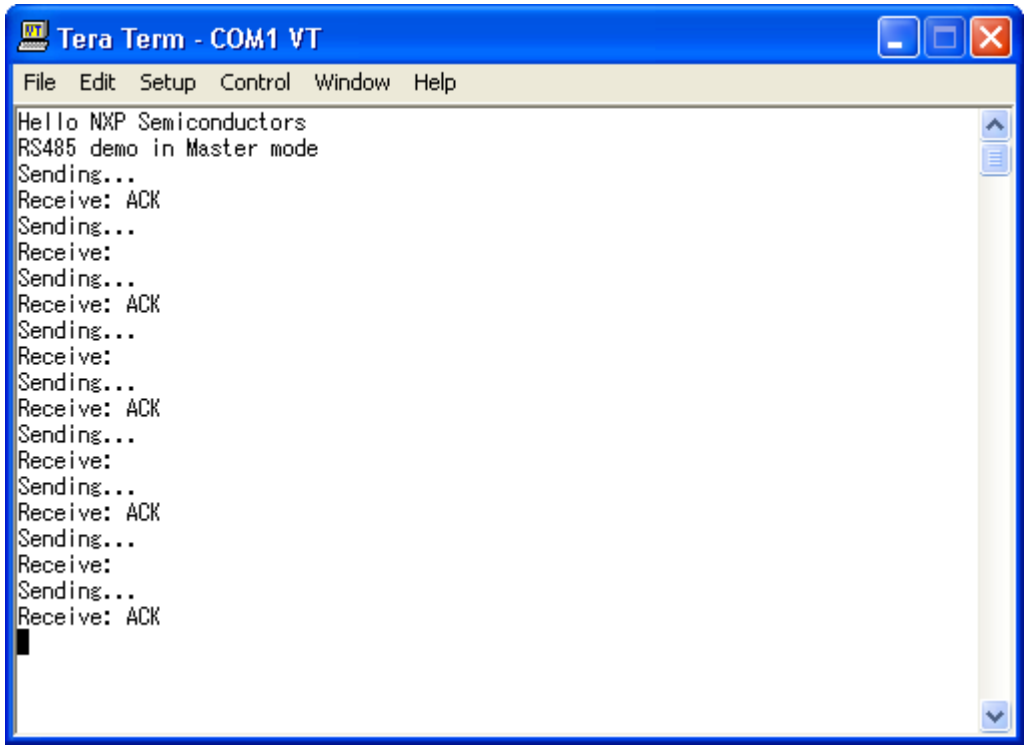

Fig 157. UART RS485 master

3.22.9 RS485_Slave

3.22.9.1 Example description

Purpose

This example describes how to use RS485 functionality on UART1 of LPC1768 in slave mode.

Process

RS485 function on UART1 acts as 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.

3.22.9.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

rs485_slave.c: Main program

3.22.9.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

RS485 connection

Pls see the 'Transceiver_Master.png' in this directory for wiring information.

3.22.9.4 Running mode

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

3.22.9.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Connect RS485 between master board and slave board (using ADM485)

Step 5: Run example and using Serial display to monitor the communication between master and slave.

Note: The RS485 slave device must initialize first.

The screen will be displayed like this:

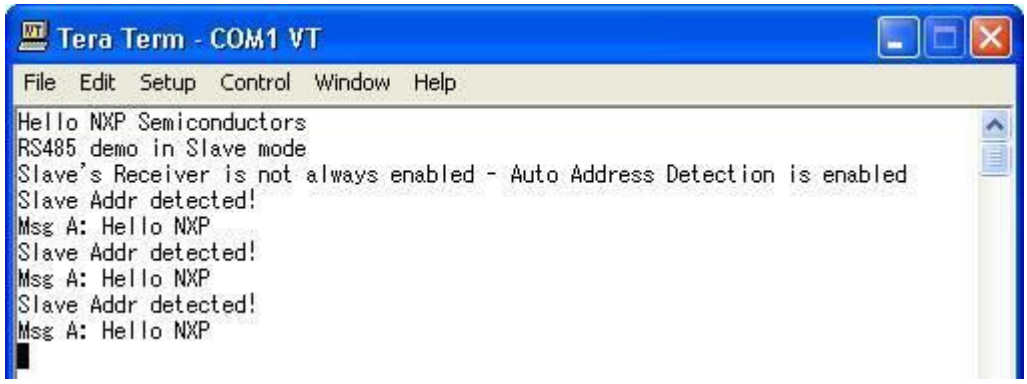

Fig 158. UART RS485 Slave demo

3.22.10 UART1_FullModem

3.22.10.1 Example description

Purpose

This is a simple UART example using UART1 will Full modem mode

Process

Uart pin configuration:

- If using MCB1700 eval board, assign pin P2.0 - P2.7 for UART1
- If using IAR 1768 KS board, assign pin P0.7 - P0.15 for UART2

Configure UART using the following settings:

- Baudrate = 9600bps
- 8 data bit
- 1 Stop bit
- None parity

After reset UART will send the welcome massage then start to receive the character from PC and send back that character to PC.

3.22.10.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

uart_fullmodem_test.c: Main program

3.22.10.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREG: ON
- VBUS: ON
- Remain jumper: OFF

IAR LPC1768 KickStart version A

- PWR_SEL: depend on power source
- DBG_EN: ON
- Remain jumper: OFF

Full Modem connection

Because eval 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 must be pulled-low

- MCB board: CTS pin is P2.2
- IAR board: CTS pin is P0.17

3.22.10.4 Running mode

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

3.22.10.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART1 on this board to COM port on your computer

Step 4: Run example and using Serial display to see the result.

The screen will be displayed like this:

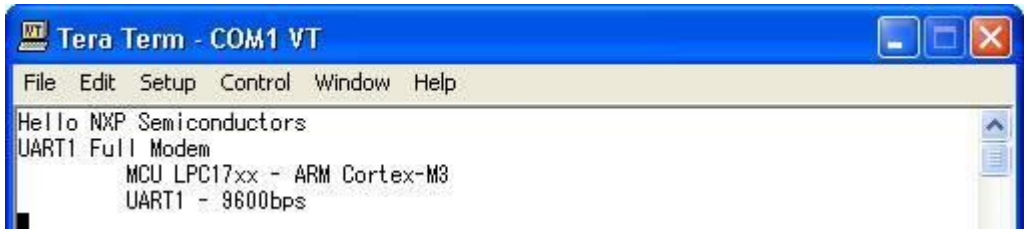

Fig 159. UART Full Modem demo

3.23 USBDEV

3.23.1 USBAudio

3.23.1.1 Example description

Purpose

This example describes how to use USBDEV on LPC1768 to demo the USB Audio Device - Speaker.

Process

Clock Settings:

- XTAL = 12 MHz
- PLL = 400 MHz
- processor clock = CCLK = 100 MHz
- USB clock = 48 MHz
- CCLK / 4 clock = 25 MHz

The USB Audio Device is recognized by the host PC running Windows which will load a generic Audio driver and add a speaker which can be used for sound playback on the PC. Potenciometer on the board is used for setting the Volume.

3.23.1.2 Directory contents

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

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

adcuser.h/.c: Audio Device Class Custom User Module

audio.h: USB Audio Device Class Definitions

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

usb.h: USB Definitions

usbaudio.h: USB Audio Demo Definitions

usbcfg.h: USB Custom Configuration

usbcore.h/.c: USB Core Module

usbdesc.h/.c: USB Descriptors

usbdmain.c: main program

usbhw.h/.c: SB Hardware Layer Module

usbreg.h: USB Hardware Layer Definitions for NXP Semiconductors LPC

usbuser.h/.c: USB Custom User Module

makefile: Example's makefile (to build with GNU toolchain)

3.23.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREGS: ON

- VBUS: ON
- D+: DEVICE
- D-: DEVICE
- UMODE: 1-2 (USB)
- E/U: 1-2 (USB)
- Remain jumpers: OFF

3.23.1.4 Running mode

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

3.23.1.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware as above instruction

Step 4: Hit reset button to run example.

Step 5: After see UGL(USB Good Link) led on board turn on, open "Device Manager > Sound,video and game controller" to see if "USB Audio Device" appears or not.

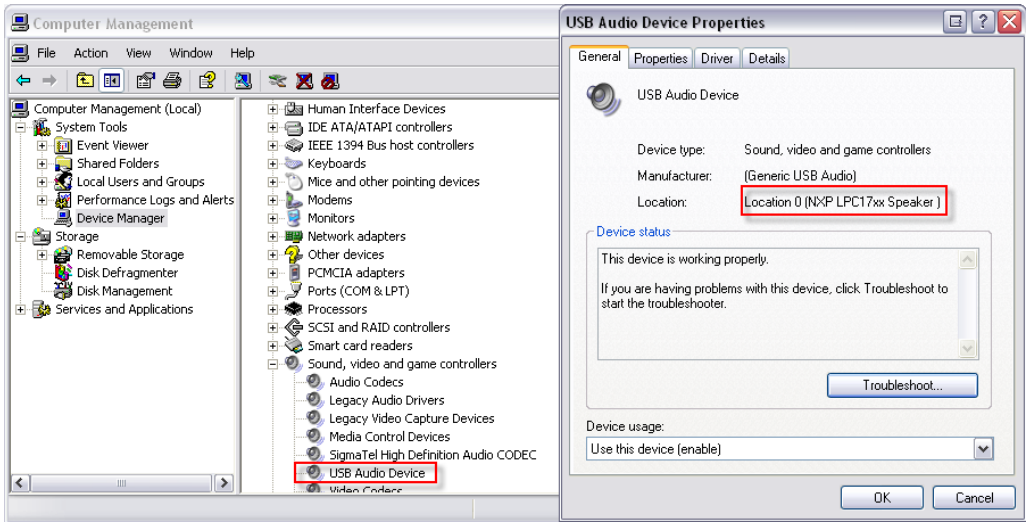

Fig 160. USB Audio Device

Step 6: Open "Control Panel > Sound > Audio" choose Default device is "NXP LPC17xx Speaker"

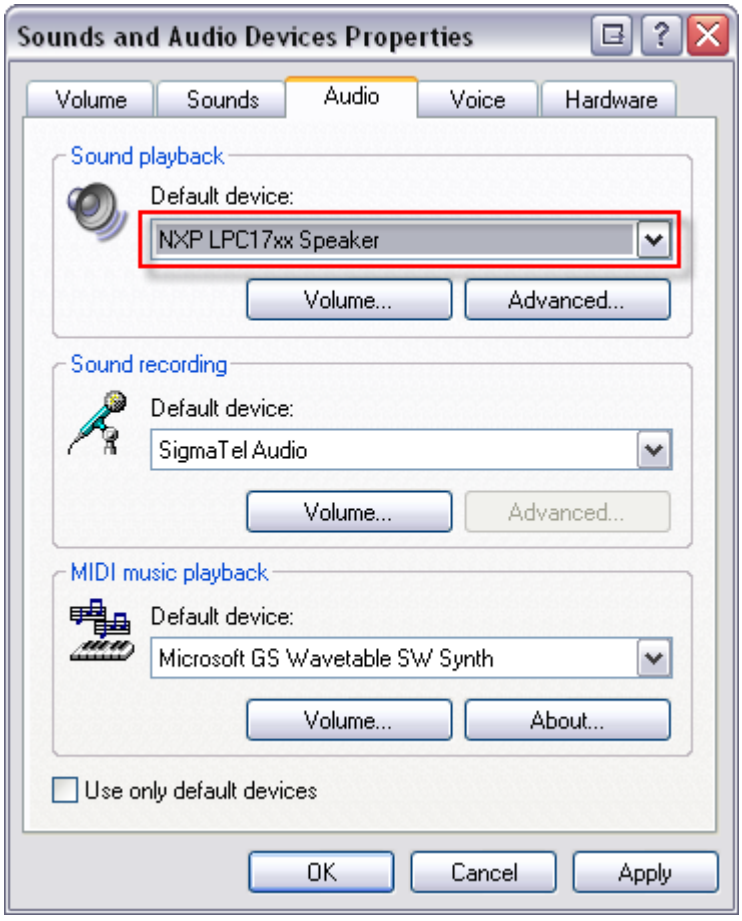

Fig 161. Choose “NXP LPC17xx Speaker” on Audio tab

Step 7: Choose Default device "NXP LPC17xx Speaker" for Voice tab

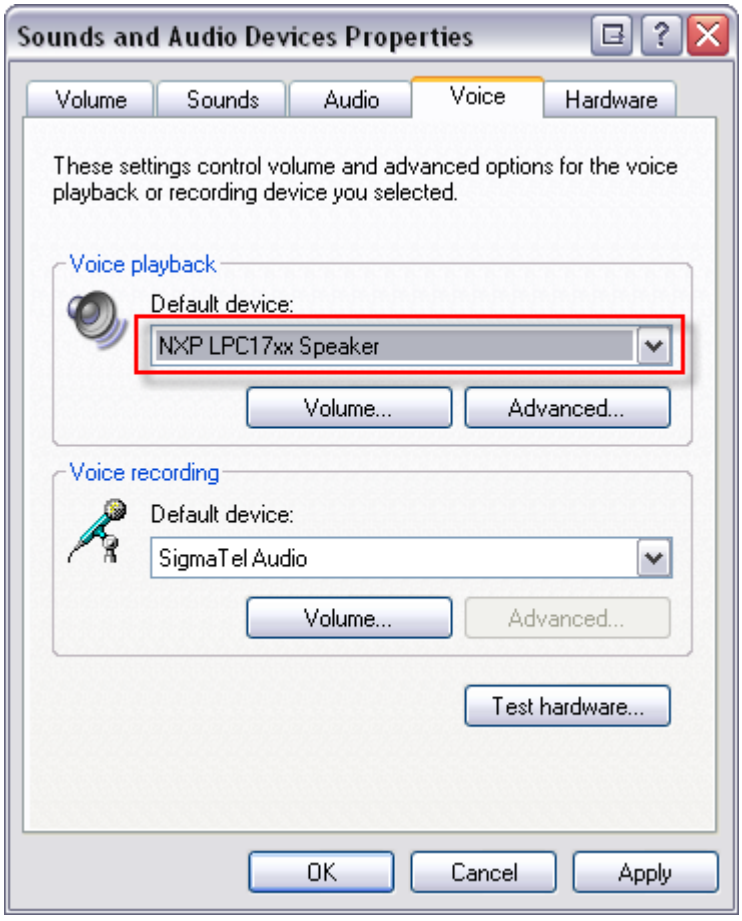

Fig 162. Choose "NXP LPC17xx Speaker" on Voice tab

Step 8: Try to play a sound file (such as .mp3 file) and hear the voice from speaker on board. Turn potiometer to change volume.

3.23.2 USBCDC

3.23.2.1 Example description

Purpose

This example describes how to use USBDEV on LPC1768 to demo the virtual COM port.

Process

Clock Settings:

- XTAL = 12 MHz
- PLL = 400 MHz
- processor clock = CCLK = 100 MHz
- USB clock = 48 MHz

- CCLK / 4 clock = 25 MHz

The PC will install a virtual COM port on the PC (see Driver Installation). After installation an additional port "LPC17xx 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.

Testing the USB Virtual COM port with serial cable:

Open two Hyperterminal windows.

One with "LPC17xx USB VCom Port(COMx)"

One with "Communications Port (COM1)".

Connect PC port COM1 to the comport on the board and open "COM1" and "COMx". Data from COM1 will be echoed on "COMx" and visa versa.

So, this is bi-directional communication between the physical COM port 0 or 1 on the board and the virtual COM port COMx on host PC.

By default, COM PORT1 on the board is used for VirtualCOM port test.

In order to use COM PORT0 on the board, modify the definition PORT_NUM from 1 to 0 in serial.h, recompile and reprogram the flash. RST jumper

needs to removed to start the Virtual COM port test.

3.23.2.2 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 serach'
- 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'

3.23.2.3 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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

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

makefile: Example's makefile (to build with GNU toolchain)

lpc17xx-vom.inf: driver info for VCOM LPC17xx (used when Windows requires install driver)

3.23.2.4 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREGS: ON
- VBUS: ON
- D+: DEVICE
- D-: DEVICE
- UMODE: 1-2 (USB)
- E/U: 1-2 (USB)
- Remain jumpers: OFF

3.23.2.5 Running mode

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

3.23.2.6 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware as above instruction

Step 4: Hit reset button to run example. (install driver if required)

Step 5: After see UGL(USB Good Link) led on board turn on, open Device Manager > Ports (COM & LPT) see if "LPC17xx USB Vcom Port (COMx)" appears or not.

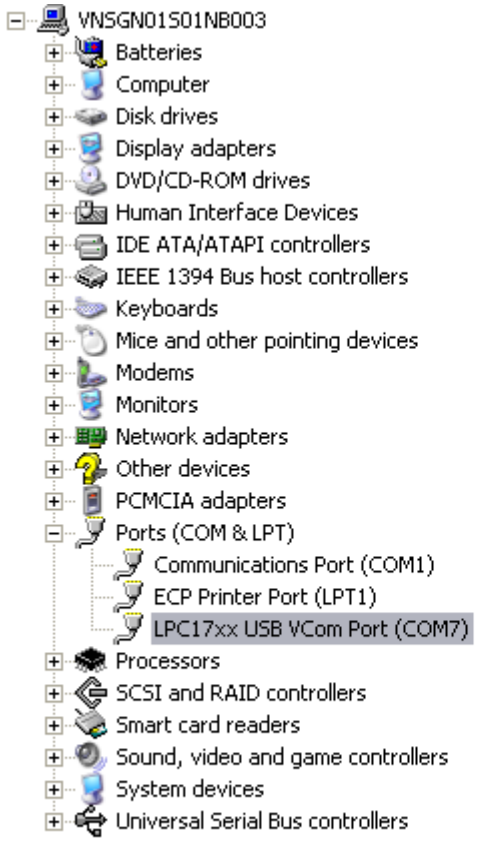

Fig 163. LPC17xx USB VCOM Port

Step 6: Open one HyperTerminal for COM1 and other for COMx (in this case it's COM7) with below configuration:

- 9600 bps
- 8 data bits
- none parity
- 1 stop bit
- None flow control

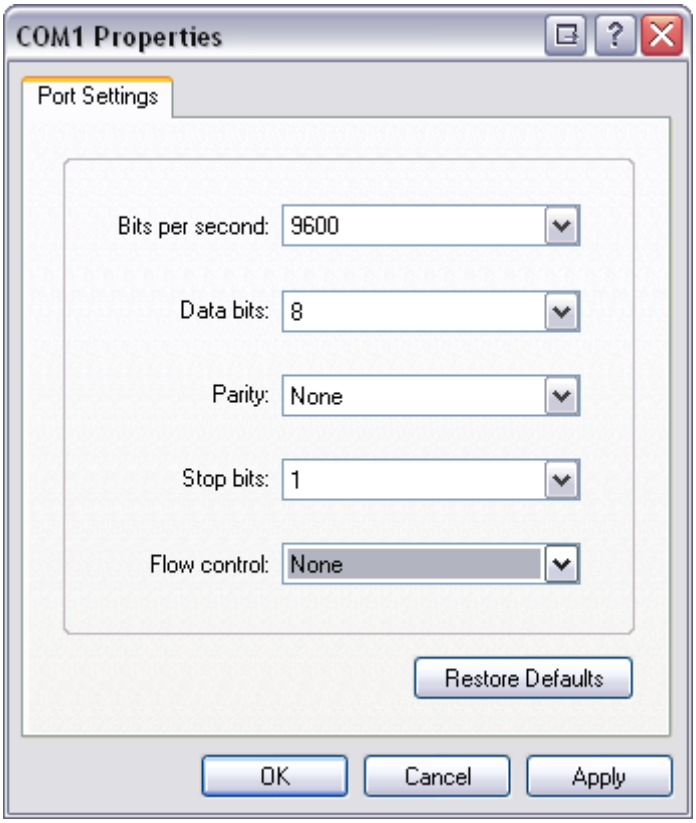

Fig 164. HyperTerminal for COM1 & COM7

Step 7: Sure that you already connected COM1 port on board with PC. Type any character on one HyperTerminal screen and see this character will be echoed in other screen and vice versa

3.23.3 USBHID

3.23.3.1 Example description

Purpose

This example describes how to use USBDEV on LPC1768 to run a simple HID application.

Process

Clock Settings:

- XTAL = 12 MHz
- PLL = 400 MHz
- processor clock = CCLK = 100 MHz
- USB clock = 48 MHz

- CCLK / 4 clock = 25 MHz

It demonstrates an USB HID (Human Interface Device):

- LEDs (LEDs controlled by P1.28, P1.29, P1.31, P2.2-6)
- Push Button (INT0)

The USB HID is recognized by the host PC running Windows which will load a generic HID driver. The board LEDs and Push Buttons can then be accessed from the PC through a custom HID Client Program.

3.23.3.2 Directory contents

\app: HID Client application, use to test HID class in this example.

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

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

hid.h: USB HID (Human Interface Device) Definitions

hiduser.h/.c: HID Custom User module

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

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

makefile: Example's makefile (to build with GNU toolchain)

demo.h/.c: Main program

3.23.3.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREGS: ON
- VBUS: ON
- LED: ON
- INT1: ON
- D+: DEVICE
- D-: DEVICE
- UMODE: 1-2 (USB)
- E/U: 1-2 (USB)
- Remain jumpers: OFF

3.23.3.4 Running mode

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

3.23.3.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware as above instruction

Step 4: Hit reset button to run example.

Step 5: After see UGL(USB Good Link) led on board turn on, run HIDClient application by clicking HIDClient.ext at USBHID\app folder

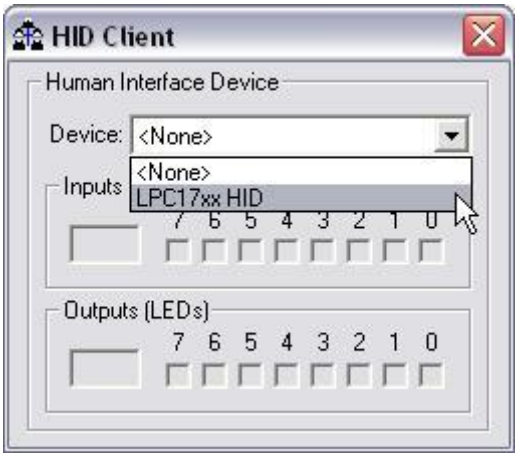

Fig 165. HID Client application

Step 6: Chose "LPC17xx HID" device.

Step 7: Click in any boxes on Outputs (LEDs) to turn on/off 8 LEDs on board

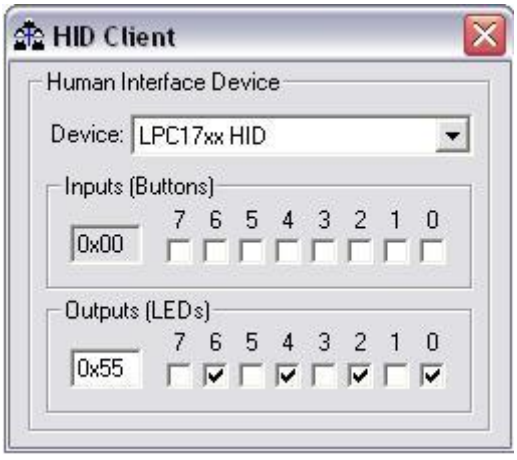

Fig 166. Tick on outputs(LEDs) to turn on/off LEDs on MCB1700 board

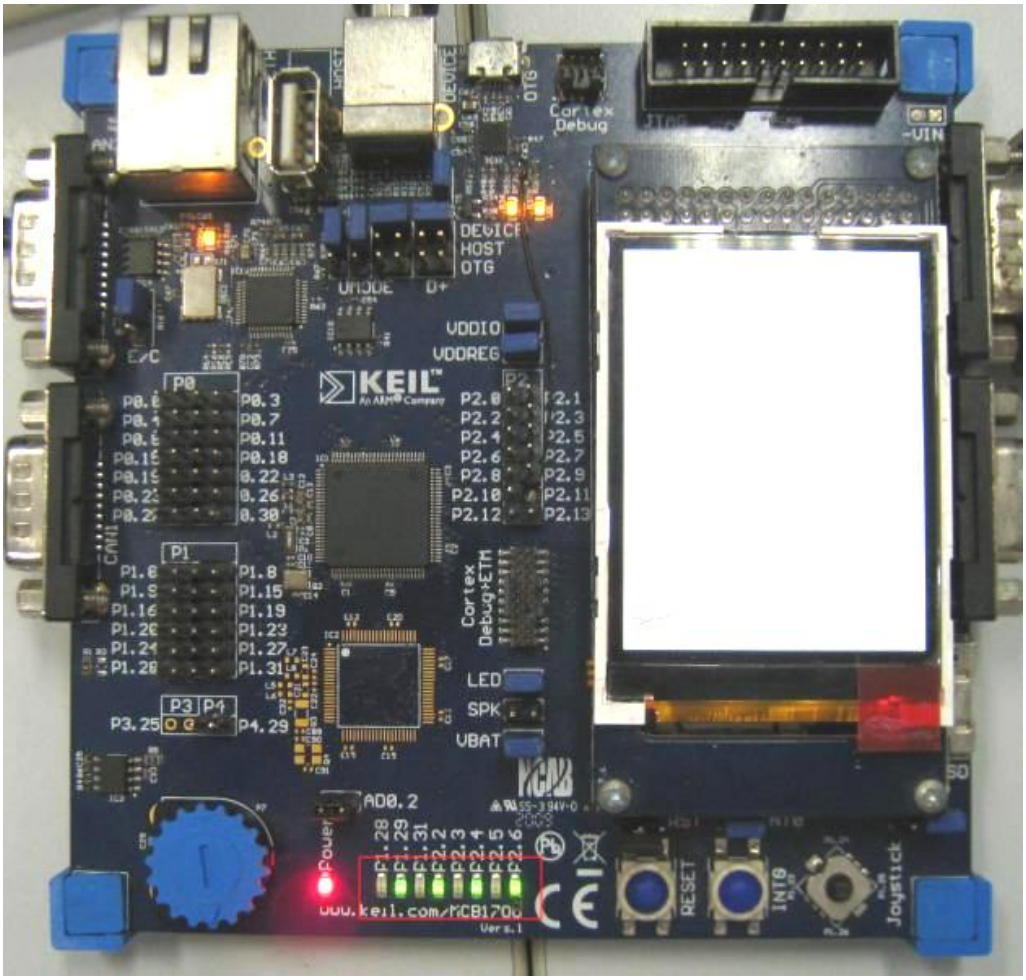

Fig 167. LEDs display on MCB1700 board

Step 8: Hit INT1 button and see if Inputs(BUTTON) box-0 turn on or not

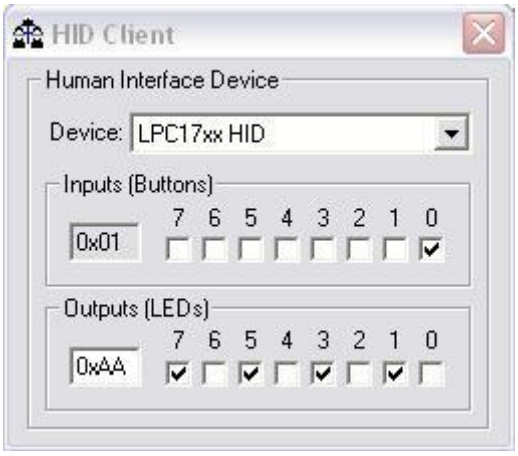

Fig 168. Inputs turn on 0x01 value when hit button INT1

3.23.4 USBMassStorage

3.23.4.1 Example description

Purpose

This example describes how to write a simple USB Mass Storage application on LPC1768.

Process

The MassStorage project is a Mass Storage simple demo run on LPC1768

Clock Settings:

- XTAL = 12 MHz
- PLL = 400 MHz
- processor clock = CCLK = 100 MHz
- USB clock = 48 MHz
- CCLK / 4 clock = 25 MHz

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.

3.23.4.2 Directory contents

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

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

DiskImg.c: Disk Image (LPC17xx) data

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

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 Configurate 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)

makefile: make file (to build with GNU toolchain)

3.23.4.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREGS: ON
- VBUS: ON
- LED: ON
- INT1: ON
- D+: DEVICE
- D-: DEVICE
- UMODE: 1-2 (USB)
- E/U: 1-2 (USB)
- Remain jumpers: OFF

IAR LPC1768 KichStart version A

- PWR_SEL: depend on power source
- DBG_EN : ON
- USB_D- : 1-2 (USB device is active)
- USB_D+ : 1-2 (USB device is active)
- Remain jumpers: OFF

3.23.4.4 Running mode

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

3.23.4.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Configure hardware as above instruction

Step 4: Hit reset button to run example.

Step 5: After see UGL(USB Good Link). Open My Computer, you will see 'LPC17xx' disk.

Fig 169. LPC17xx disk appear on My Computer

Step 6: Open this disk, you will see 'Readme.txt' file (it has read-only attribute)

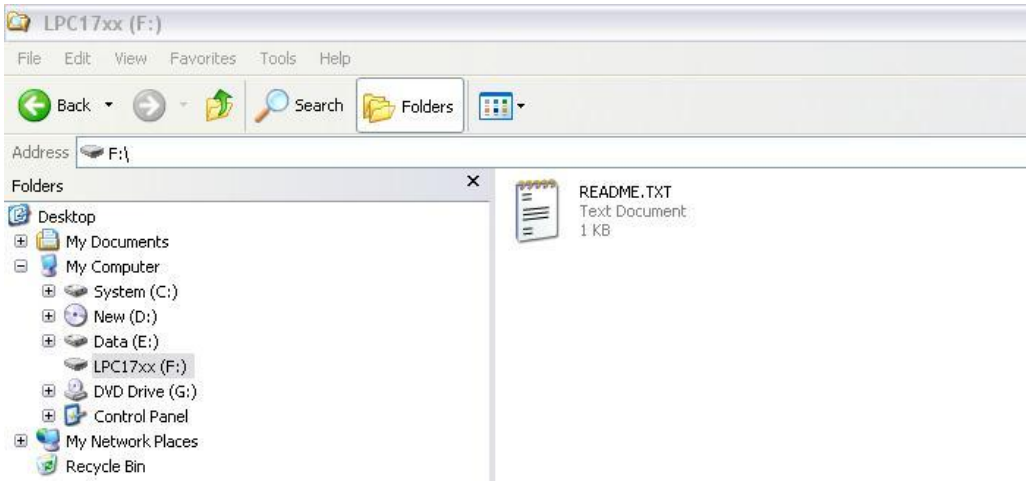

Fig 170. Readme.txt file in LPC17xx disk

Step 7: Try to copy/delete another file into disk to test if Mass Storage is work we

3.24 WDT

3.24.1 INTERRUPT

3.24.1.1 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 just generates interrupt when the Watchdog times out.

WDT setting:

- clock source: IRC (Internal RC oscillator)
- time-out = 5000000us = 5s

Before WDT interrupt, current value of Watchdog timer will be written continuously into serial display (these value will be decreased from 5000000 to 0). After 5s, the Watchdog timer times out, WDT interrupt sevice routine 'WDT_IRQHandler( )' will be invoked to change 'wdt_flag'. The notice sentence will be display and LED begin blinky.

Note that: dispaly data via UART will caused delay.

3.24.1.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

wdt_interrupt_test.c: Main program

3.24.1.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREGS: ON
- VBUS: ON
- LED: ON
- Remain jumpers: OFF

IAR LPC1768 KichStart version A

- PWR_SEL: depend on power source
- DBG_EN : ON
- Remain jumpers: OFF

3.24.1.4 Running mode

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

3.24.1.5 Step to run

Step 1: Choose correct working board by uncomment correct defined board in main.c file

- MCB1700 board, uncomment "#define MCB_LPC_1768"
- IAR-LPC1768-KS board, uncomment "#define MCB_LPC_1768"

  (Should not uncomment both symbols at the same time)

Step 2: Build example.

Step 3: Burn hex file into board (if run on ROM mode)

Step 4: Connect UART0 on this board to COM port on your computer

Step 5: Configure hardware and serial display as above instruction

Step 6: Run example and observe data on serial display

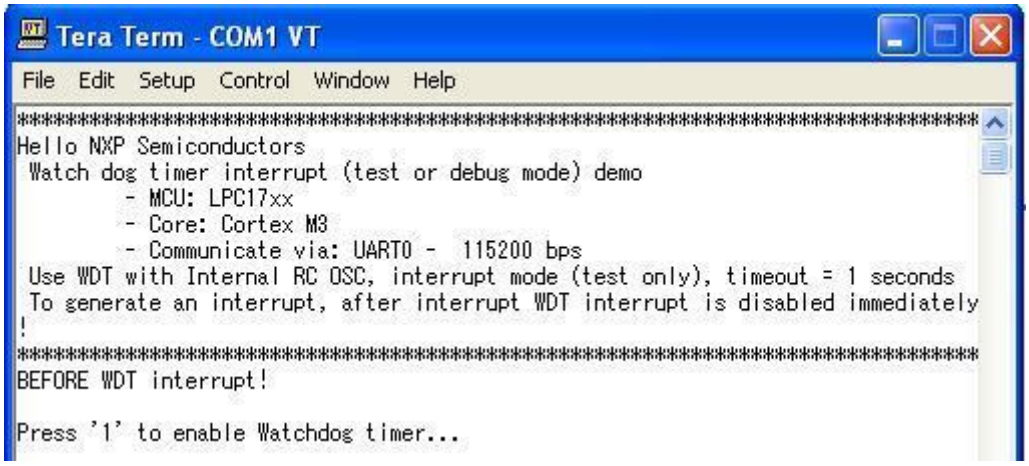

Fig 171. WDT interrupt welcome screen

- Press '1' to start WDT operation

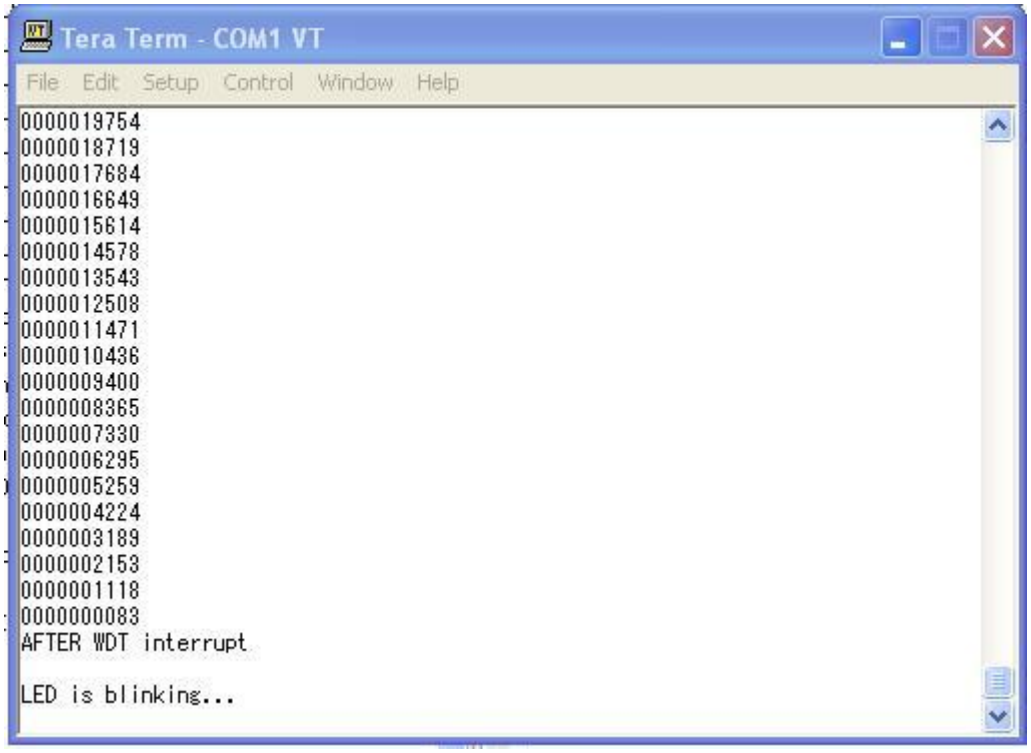

Fig 172. WDT demo

- See the current value of WDT displayed on screen
- After WDT interrupt, the notice sentence will be display and LED begin blink.

Note that:

- If using MCB1700 board, LED2.2 is available
- If using IAR1768 board, LED1 is available

3.24.2 RESET

3.24.2.1 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 = 5s
- 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)

3.24.2.2 Directory contents

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

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

lpc17xx_libcfg.h: Library configuration file - include needed driver library for this example

makefile: Example's makefile (to build with GNU toolchain)

wdt_reset_test.c: Main program

3.24.2.3 Hardware configuration

These jumper must be configured as follows:

Keil MCB1700 with LPC1768 version 1.0

- VDDIO: ON
- VDDREGS: ON
- VBUS: ON
- Remain jumpers: OFF

IAR LPC1768 KichStart version A

- PWR_SEL: depend on power source
- DBG_EN : ON
- Remain jumpers: OFF

3.24.2.4 Running mode

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

3.24.2.5 Step to run

Step 1: Build example.

Step 2: Burn hex file into board (if run on ROM mode)

Step 3: Connect UART0 on this board to COM port on your computer

Step 4: Configure hardware and serial display as above instruction

Step 5: Run example and observe data on serial display

- If in between 5s, hit RESET button, after reset, we will have the notice

  " Last MCU reset caused by External!"

- If not, WDT cause chip reset and after reset, we will have the notice

  " Last MCU reset caused by WDT TimeOut!"

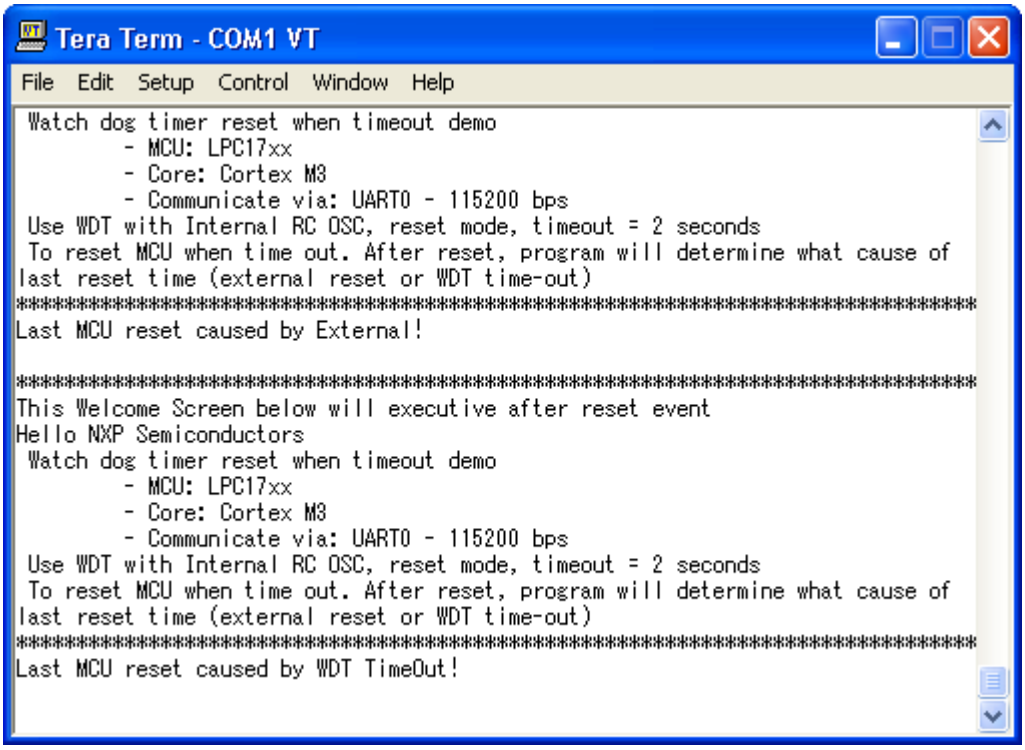

Fig 173. WDT reset demo

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

continued >>

continued >>

continued >>

continued >>

continued >>

continued >>

continued >>