User manual

DA14580 Peripheral Examples

UM-B-005

Abstract

This document describes how to configure a DA14850 development board and run the set of engineering application examples for the various hardware peripherals (SPI flash, EEPROM, Quadrature Encoder, Buzzer, etc.)

Contents

Figures

Tables

1 Terms and definitions

ADC	Analog to Digital Converter
BLE	Bluetooth Low Energy
DK	Development Kit
CS	Chip Select
EEPROM	Electrically Erasable Programmable Memory
GPIO	General Purpose Input Output
PWM	Pulse Width Modulation
SDK	Software Development Kit
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transceiver

2 References

1. UM-B-014, DA14580 Development Kit, Dialog Semiconductor.

2. UM-B-004, DA14580 Peripheral Drivers, Dialog Semiconductor.

3. UM-B-010, DA14580 Proximity Application, Dialog Semiconductor.

4. AN-B-001: Booting from Serial Interfaces, Dialog Semiconductor.

5. DA14580 Datasheet, Dialog Semiconductor.

3 Introduction

The Peripherals Examples application combines DA14580's peripheral connectivity capabilities to demonstrate examples of peripheral devices usage, such as SPI flash and I2C EEPROM memories, Quadrature, buzzer etc. The user interaction, when applicable, is done via a UART console.

The applications provide the following examples:

- UART Print String Example: How to configure, initiate, send to and receive from the UART interface.
- SPI Flash Memory Example: How to initiate, read, write and erase an SPI flash memory.
- I2C EEPROM Example: How to initiate, read, write and erase an EEPROM memory.
- Quadrature Encoder Example: How to configure and read from the quadrature decoder peripheral.The Wakeup Timer setup for responding to GPIO activity is also demonstrated in this example.
- TIMER0 (PWM0, PWM1) Example: How to configure TIMER0 to produce PWM signals. A melody is produced on an externally connected buzzer.
- TIMER2 (PWM2, PWM3, PWM4) Example: How to configure TIMER2 to produce PWM signals.
- Battery Example: How to read the battery indication level, using the ADC.

4 Software description

The DA14580 Software Development kit (SDK) includes a Keil project that implements a set of engineering examples. These demonstrate the use of the Hardware Adaptation Layer (HAL) for accessing the main peripheral devices of DA14580.

The project name is *peripheral_examples\DA14580_peripheral_setup.uvproj* and it uses the HAL drivers implemented under *dk_apps\src\plf\refip\src\driver* folder**.**

To use the DA14580 HAL drivers, one should:

- Add the driver's source code file (e.g. *spi\spi.c*) to the project.
- Include the driver's header file (e.g. *spi\spi.h*) whenever the driver's API is needed.
- Add the driver folder path to the Include Paths (C/C++ tab of Keil Target Options).

There is an example function for each of the main peripherals:

- UART: uart_test()
- SPI Flash: spi_test()
- I2C EEPROM: i2c_test(),
- Quadrature encoder: quad_test()
- Timer0: timer0_test()
- Timer2: timer2_test()
- ADC: batt_test()

Please note that some of the peripheral hardware (like the SPI flash and I2C EEPROM board) is provided separately and is not included in the default SDK configuration.

The project includes the following files:

DA14580_examples.c, .h: Includes both the main and menu functions**.** The battery example function, which is based on the battery and the ADC drivers is also included here.

peripherals.c, .h: Includes the system initialization and GPIO configuration functions

uart.c, .h: Includes the UART initialization, reception and transmission functions. It calls functions from the GPIO driver. Please note: These files contain legacy code and are not part of the Peripheral Drivers API, which is the recommended resource for new projects development. For information on the Peripheral Drivers API functions, please refer to [1].

spi_test.c, .h: Includes the SPI example functions and it is based on the SPI and SPI flash drivers

eeprom_test.c, .h: Includes the EEPROM example functions and it is based on the I2C EEPROM driver.

quad_decoder_test.c, .h: Includes the Quadrature and Wakeup timer example functions and it is based on the Quadrature and Wakeup timer driver.

pwm_test.c, .h: Includes the PWM example functions and is based on the PWM driver.

periph_setup.h: Defines the hardware configuration, such as GPIO assignment for each peripheral device. It is based on the following user configuration:

- **SPI Flash with UART** (defined as SPI_FLASH_WITH_UART_EXAMPLE): Configures UART on pins {04, 07} and SPI flash on pins {00, 03, 05, 06}. A hardware modification is needed due to conflict on pin 05 (see section 5.1).
- **I2C EEPROM with UART** (defined as I2C_EEPROM_WITH_UART_EXAMPLE): Configures UART on pins {04, 05} and I2C EEPROM on pins {02, 03}.
- **Quadrature Encoder, Timers & Buzzer with UART**

  (defined as QUADRATURE_ENCODER_WITH_BUZZER_AND_UART_EXAMPLE): Configures UART on pins {04, 05} and the Quadrature Decoder pins: CHX_A {00} and CHX_B{01}, Button K1{11}, Button K2{06} and the PWM pins: PWM0{02}, PWM1{03}, PWM2{10}, PWM3{12}, PWM4{13}.

5 Getting started

5.1 Hardware configuration

This section describes the GPIO assignment of the DA14580 development board for each example. It also illustrates the compilation procedure and usage of the engineering examples.

It is assumed that the user is familiar with the use of the DA14580 Development Kit [2].

Configure the DA14580 development board

Prior to running a desired peripheral example, the user has to configure the DA14580 development board accordingly, depending on the hardware configuration selected:

- **UART** example: The user has to connect PIN1 to PIN2 and PIN3 to PIN4 on both J25 and J26 connectors to enable UART TX/RX and CTS/RTS functionality.
- **SPI Flash Memory** example: Since UART RX default GPIO pin conflicts with the SPI MISO pin (P0_5), a separate pin must be used for UART RX. Then, J25 connector's *PIN3* must be connected to *P0_7* with a cable (see Figure 1 below). In addition, the UART RTS conflicts with the SPI /CS, so the jumpers on connector J26 have to be removed and no CTS/RTS functionality can be supported (or a separate GPIO pin must be connected to J26 PIN3 with a cable, similar to the previously described approach for the UART RX).
- **I2C EEPROM** example: The pins used for the I2C EEPROM (P0_2 and P0_3) conflict with UART CTS/RTS default GPIOs, so the user has to remove the jumper from connector J26 (UART CTS/RTS functionality will not be enabled). There is no conflict with UART default TX/RX pins, so the user has to connect PIN1 to PIN2 and PIN3 to PIN4 on connector J25 to enable UART functionality.
- **Quadrature Encoder** example: The quadrature encoder CHX_A and CHX_B pins have to be connected to P0_0 and P0_1 respectively. The common terminal of the quadrature decoder must be connected to ground. Please, refer to [1] for additional information on the DK hardware layout. To enable K1 and K2 buttons functionality, the user has to connect PIN5 to PIN6 on both J15 and J16 connectors.
- **Timer0 (PWM0, PWM1)** example:

  In order to have an audio indication of the produced PWM signals, the user can connect a buzzer in P0_2 and ground (PWM0) or in P0_3 and ground (PWM1).
- **Timer2 (PWM2, PWM3, PWM4)** example:

  To use the LED segments inside D1 and D2 as a visual indication for signals PWM2, PWM3 and PWM4, the user has to connect PIN3 to PIN4 on connector J16 (PWM2), PIN1 to PIN2 on connector J16 (PWM3) and PIN3 to PIN4 on connector J15 (PWM4). The brightness of the LED segments is then directly influenced by the duty cycle of the PWM signals.
- **Battery** example:

  A CR2032 battery has to be inserted in the battery case of the DK. The user has to connect PIN3 to PIN5 on jumper J13 to force the DK to use the battery as its power source. After the test the user should restore the original J13 jumper configuration (connect PIN4 to PIN5).

For convenience, the GPIO assignment options for the aforementioned examples have been grouped into three main hardware configurations. The user can select one of these using the configuration wizard that accompanies *periph_setup.h* (see Appendix A).

The GPIO pin assignment selected for each example and the related hardware modifications are summarized in Table 1.

Table 1 - GPIO pin assignment

Example	UART Print String	SPI Flash memory	I2C EEPROM	Quadrature Encoder Timer0 (PWM0, PWM1) Timer2 (PWM2, PWM3, PWM4)
Hardware Configuration (see Appendix A)	**I2C EEPROM with UART**	**SPI Flash with UART**	**I2C EEPROM with UART**	**Quadrature Encoder, Timers & Buzzer with UART**
P0_0		SPI CLK		Encoder CHXA
P0_1				Encoder CHXB
P0_2	UART CTS		I2C SCL	PWM0
P0_3	UART RTS	SPI CS	I2C SDA	PWM1
P0_4	UART TX	UART TX	UART TX	UART TX
P0_5	UART RX	SPI MISO	UART RX	UART RX
P0_6		SPI MOSI		
P0_7		UART RX		PWM2(LED D2)
P1_0				PWM3 (LED D2)
P1_2				PWM4 (LED-D1)
P1_3				
Jumper setup	J25: PIN1-PIN2, PIN3-PIN4 J26: PIN1-PIN2, PIN3-PIN4	J25: PIN1 – PIN2, PIN3 – J1.P0_7 J26: No connection	J25: PIN1-PIN2, PIN3-PIN4 J26: No connection	J15: PIN3-PIN4 J16: PIN1-PIN2, PIN3-PIN4 J25: PIN1-PIN2, PIN3-PIN4 J26: No connection

If a different GPIO assignment is required, this can be done by changing the GPIO configuration, which is explained in detail in [1].

A PCB with an SPI flash and an I2C EEPROM was used for the respective examples. The SPI flash was a 1M-bit Winbond W25X10CL and the I2C EEPROM was a 1M-bit STMicroelectronics M24M01-RMN6P. The schematics of the PCB are provided in Appendix B .

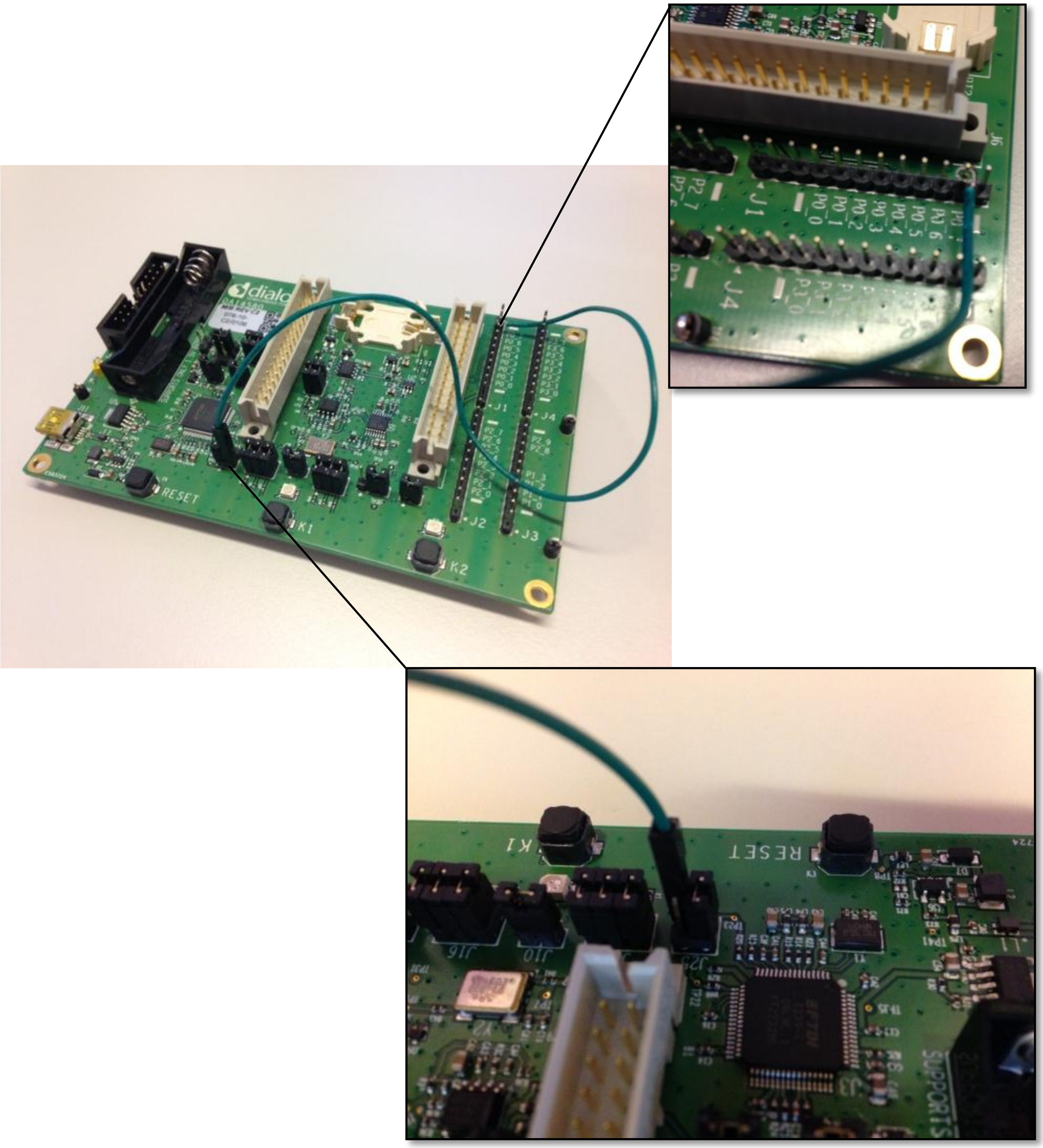

Figure 1: Connecting J25 PIN3 to pin P0_7 to resolve conflict on pin P0_5

5.2 Running the Peripheral Setup project

Open the Peripheral Setup project:

peripheral_examples\DA14580_peripheral_setup.uvproj

Build the project (F7) and follow the steps in the user guide [2] to load the executable to the DK and run the executable.

5.3 Using the console to access Peripheral Setup options menu

The DA14580 RS232 interface will be used for the UART interaction console. When the DA14580 is connected via a USB with a Windows machine (e.g. laptop) a Dual RS232-HS device should be discovered in the Windows Devices and Printers. In the Dual RS232-HS's properties window two USB Serial Ports are displayed. User must select the virtual COM port with the smaller number to provide it to a terminal console application.

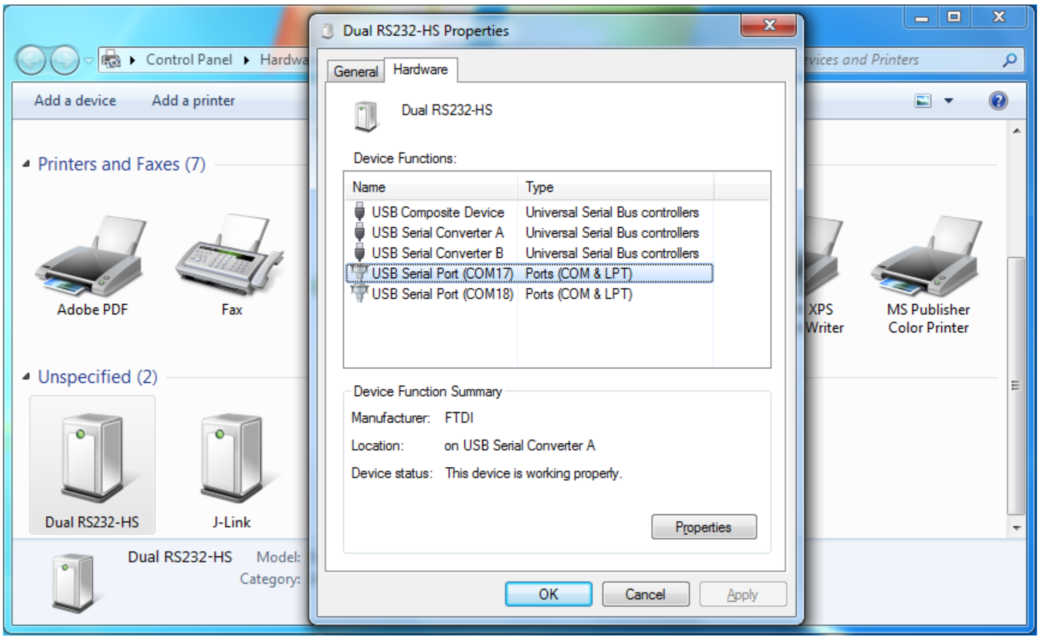

Figure 2: DA14580 virtual COM port

A terminal application (e.g. *Tera Term*) should be used next to access the console menu of the Peripheral Setup project. The COM port discovered with the process described before should be chosen here, along with the following settings:

- Baud rate: 115200
- Data: 8-bit
- Parity: none
- Stop: 1 bit
- Flow control: none

At this point, if the default hardware configuration has been selected the console's display should be identical to the one shown in Figure 3.

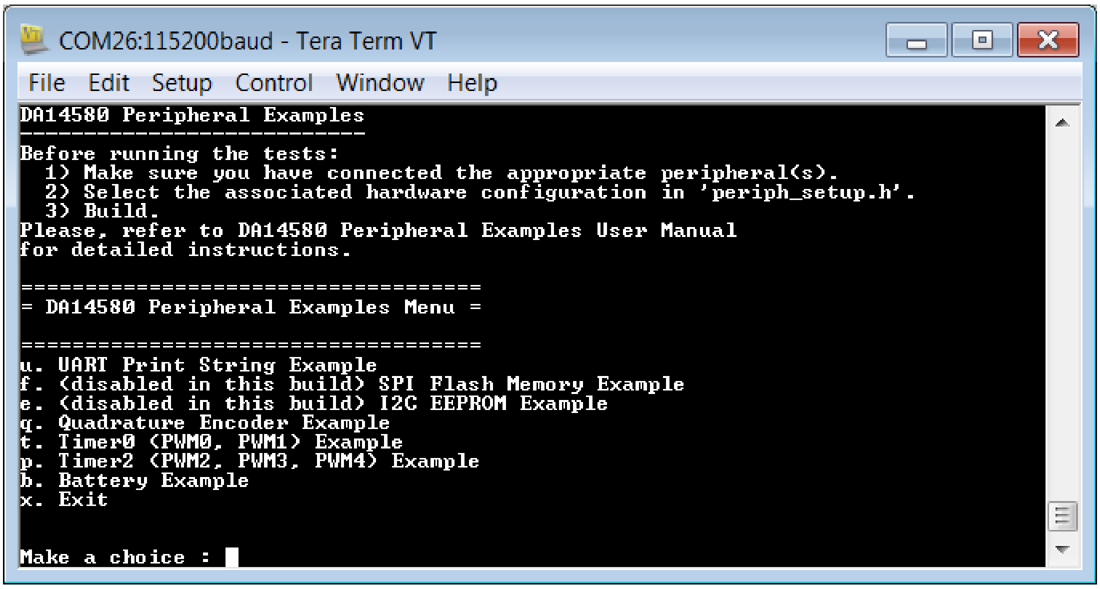

Figure 3: Peripheral Setup console menu

If another hardware configuration has been selected in the current build, a different combination of tests will be active.

In any case, The user should enter the letter that corresponds to the peripheral to be tested.

In the following section, each option is described in detail.

5.4 Running the examples

5.4.1 UART Print String example

After the user has selected the UART print string example, the console will display the message shown in Figure 4. The *uart_test* function uses *printf_byte* and *printf_string* functions to print the message on the console.

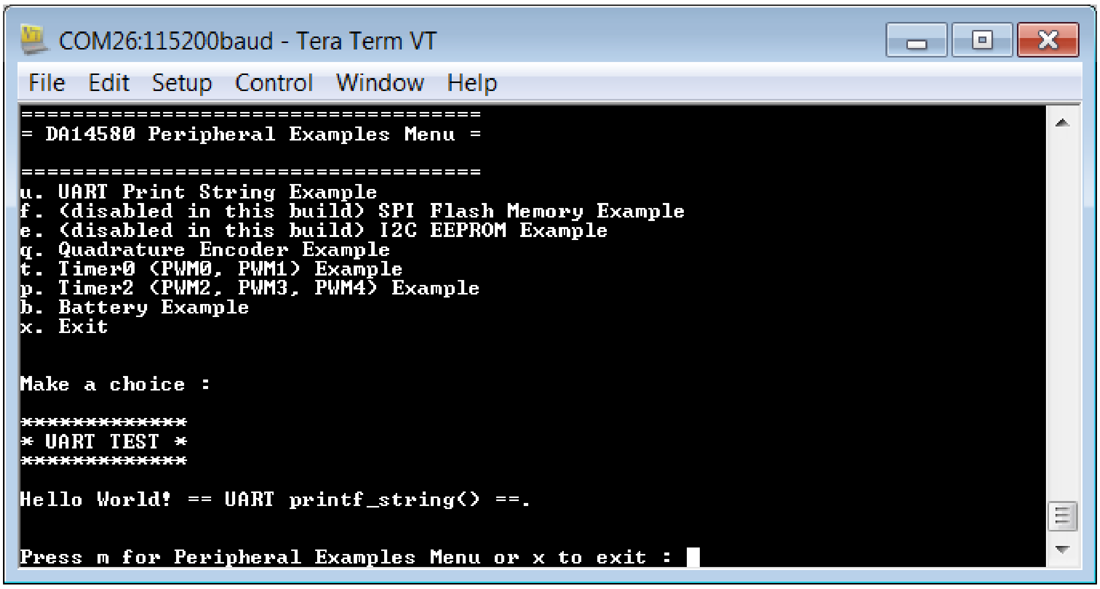

Figure 4: UART Print String example

The user can select the UART settings in header file:

peripheral_setup\include\periph_setup.h

The predefined settings are the following:

```
// Select UART settings
#define UART_BAUDRATE     UART_BAUDRATE_115K2       // Baudrate in bits/s:
                                                      { 9K6, 14K4, 19K2, 28K8,
                                                       38K4, 57K6, 115K2}
#define UART_DATALENGTH   UART_DATALENGTH_8         // Datalength in bits:
                                                       {5, 6, 7, 8}
#define UART_PARITY       UART_PARITY_NONE          // Parity: {UART_PARITY_NONE,
                                                                UART_PARITY_EVEN,
                                                                UART_PARITY_ODD}
#define UART_STOPBITS     UART_STOPBITS_1           // Stop bits: {UART_STOPBITS_1,
                                                                   UART_STOPBITS_2}
#define UART_FLOWCONTROL  UART_FLOWCONTROL_DISABLED // Flow control:
                                                       {UART_FLOWCONTROL_DISABLED,
                                                        UART_FLOWCONTROL_ENABLED}
```

The source code for this example can be found in function *uart_test* in *peripheral_setup\src\uart.c*.

Note that *peripheral_setup\src\uart.c* \uart.c file is not the Peripheral Drivers API source file for the UART block and should not be used in your applications. Please, consult [1] for detailed information on the current implementation of the UART driver and the location of the respective source code files.

5.4.2 SPI Flash memory example

Once the user has selected the SPI Flash memory example, a series of read and write operations will be performed on the SPI Flash memory (as shown in Figure 5). Prior to running the project, the user has to mount the SPI flash on DA14580's connector J1, using the pins shown in Table 1. That is, P0_0 for SCL, P0_3 for CS, P0_5 for MISO and P0_6 for MOSI. If a different selection of the GPIO pins is needed, the user should edit the SPI_CS_PIN, SPI_CLK_PIN, SPI_DO_PIN and SPI_DI_PIN defines in *periph_setup.h*.

In order for this test to be configured correctly, the hardware configuration *SPI Flash with UART* has to be selected (please refer to Appendix A). Also, the modifications described in Figure 1 should be made.

The user also has to enter the SPI flash's characteristics in header file:

peripheral_setup\include\periph_setup.h

The predefined characteristics are the following:

```
#define SPI_FLASH_SIZE   131072           // SPI Flash memory size in bytes
#define SPI_FLASH_PAGE   256              // SPI Flash memory page size in bytes
```

The user can also select the SPI module's parameters, like word mode, polarity, phase and frequency:

```
#define SPI_WORD_MODE SPI_8BIT_MODE     // Select SPI bit mode
#define SPI_SMN_MODE  SPI_MASTER_MODE   // {SPI_MASTER_MODE, SPI_SLAVE_MODE}
#define SPI_POL_MODE  SPI_CLK_INIT_HIGH // {SPI_CLK_INIT_LOW, SPI_CLK_INIT_HIGH}
#define SPI_PHA_MODE  SPI_PHASE_1       // {SPI_PHA_MODE_0, SPI_PHA_MODE_1}
#define SPI_MINT_EN   SPI_NO_MINT       // {SPI_MINT_DISABLE, SPI_MINT_ENABLE}
#define SPI_CLK_DIV   SPI_XTAL_DIV_2    // Select SPI clock divider between
                                        // 8, 4, 2 and 14
```

The user should also define the pin that is being used for the SPI CS signal:

```
#define SPI_CS_PIN    3           // Define Chip Select pin
```


The *spi_test* function performs the following tests:

1. Initializes the GPIO pins used for the SPI flash and the SPI module.
2. Reads the contents of the SPI flash and prints them to the console.
3. Reads the JEDEC ID.
4. Reads the Manufacturer/Device ID.
5. Reads the Unique ID.
6. Writes 256 bytes to the SPI flash using the Program Page instruction.
7. Reads the contents of the SPI flash and prints them on console.
8. Erases an SPI flash's sector.
9. Writes 512 bytes of data to SPI flash using the *spi_write_data* function, for writing amount of data larger than an SPI flash page.
10. Releases the configured GPIO pins and the SPI module.

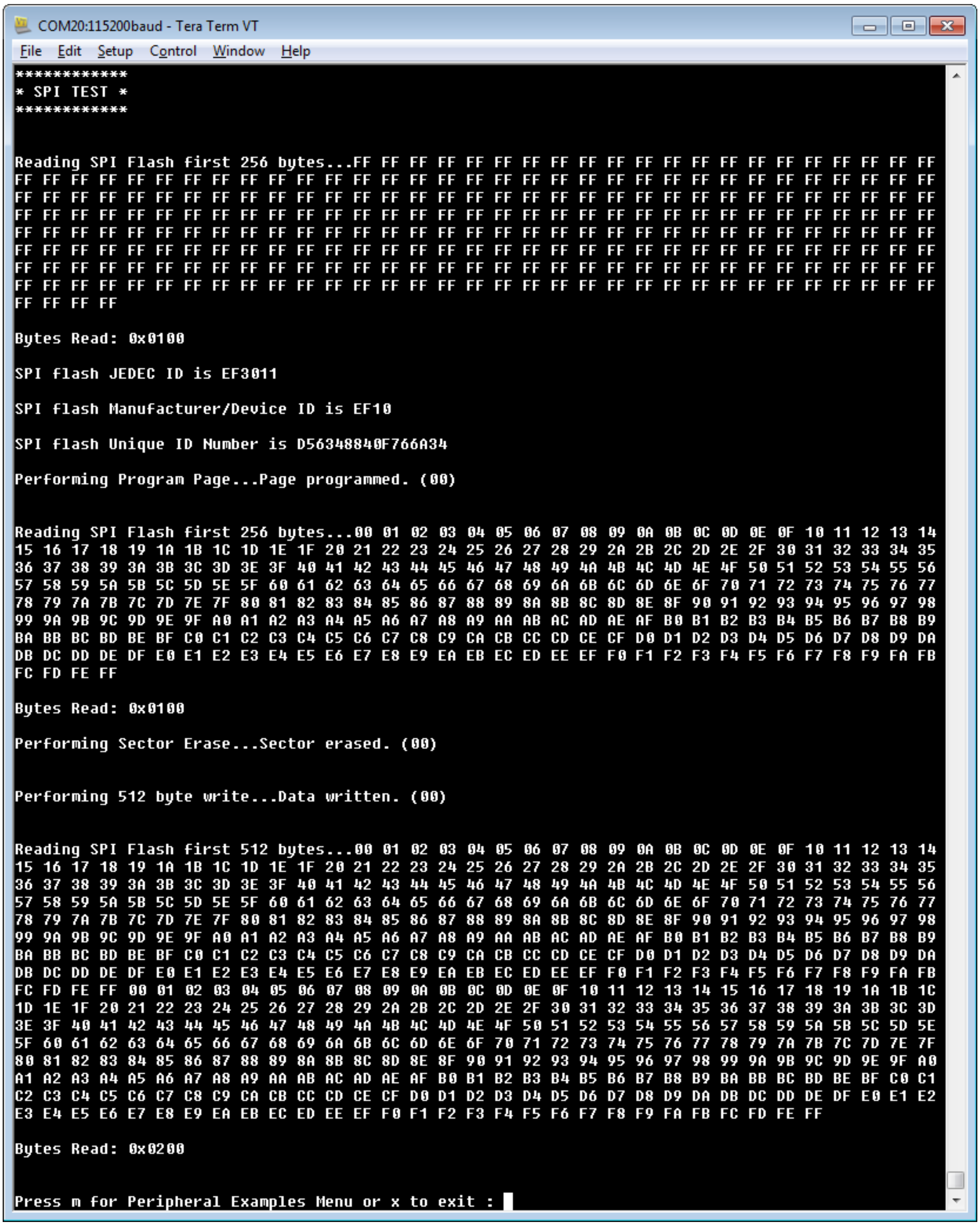

Figure 5: SPI Flash memory example

The user can change the test procedure by editing the function *spi_test*. The SPI flash driver API is described in detail in [1].

The source code for this example can be found in function *spi_test* inside *spi_test.c*.

5.4.3 I2C EEPROM example

After the user has selected the I2C EEPROM example, a series of read and write operations will be performed on the I2C EEPROM, as shown in Figure 6 below. An I2C EEPROM must be mounted on DA14580's connector J1, using the pins shown in Table 1. That is, P0_2 for SCL and P0_3 for SDA. If a different selection of GPIO pins is needed, the user should edit the I2C_GPIO_PORT, I2C_SCL_PIN and I2C_SDA_PIN defines in *periph_setup.h*.

In order for this test to be configured correctly, the hardware configuration *I2C EEPROM with UART* has to be selected (please refer to Appendix A).

The user also has to enter the I2C EEPROM's characteristics in header file:

peripheral_setup\include\periph_setup.h

The predefined characteristics are the following:

```
#define I2C_EEPROM_SIZE    0x20000     // EEPROM size in bytes
#define I2C_EEPROM_PAGE    256         // EEPROM's page size in bytes
```

The user can also select the I2C module's parameters, like slave address, speed mode, address mode and addressing scheme (1-byte/2-byte):

```
#define I2C_SLAVE_ADDRESS   0x50           // Set slave device address
#define I2C_SPEED_MODE      I2C_FAST       // Speed mode: I2C_STANDARD (100 kbits/s),
                                                          I2C_FAST:     (400 kbits/s)
#define I2C_ADDRESS_MODE    I2C_7BIT_ADDR  // Addressing mode: {I2C_7BIT_ADDR,
                                                                I2C_10BIT_ADDR}
#define I2C_ADDRESS_SIZE    I2C_2BYTES_ADD  // Address width: {I2C_1BYTE_ADDR,
                                                              I2C_2BYTES_ADDR,
                                                              I2C_2BYTES_ADDR}
```

The *i2c_test* function performs the following tests:

- Initializes the GPIO pins used for the I2C EEPROM and the I2C module.
- Writes 256 bytes of data to the I2C EEPROM.
- Reads the contents of the I2C EEPROM.
- Writes and reads bytes 0x5A, 0x6A, 0x7A and 0xFF at addresses 22, 0, 255 and 30 respectively.
- Reads the contents of the I2C EEPROM.
- Releases the configured GPIO pins and the I2C module.

This is shown in detail in Figure 6.

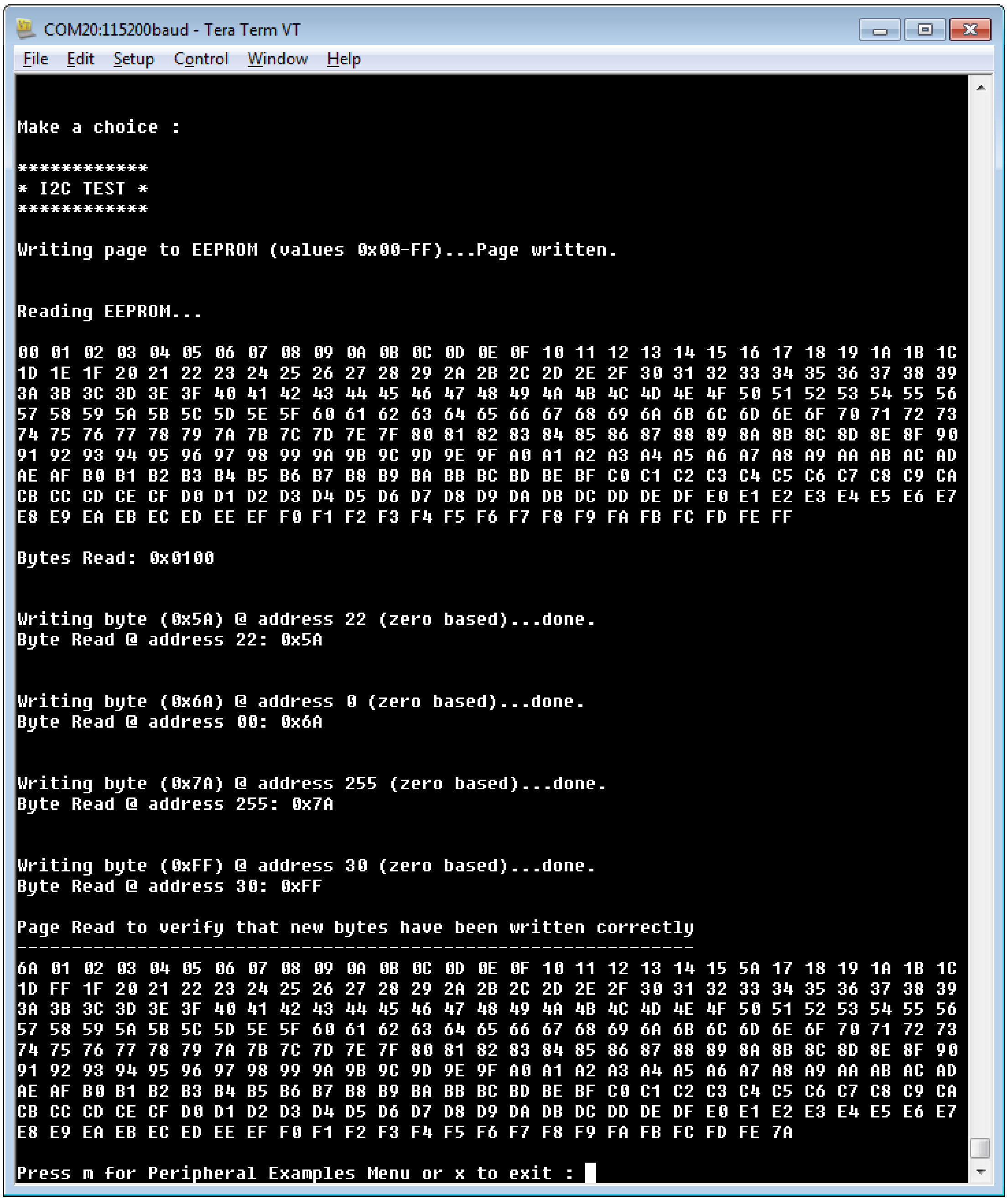

Figure 6: I2C EEPROM example

The user can change the test procedure by editing the function *i2c_test.* The I2C EEPROM driver API is described in detail in [1].

The source code for this example is located in function *i2c_test* inside *eeprom_test.c*.

5.4.4 Quadrature Encoder example

In order for this test to be configured correctly, the hardware configuration *Quadrature Encoder, Timers & Buzzer with UART* has to be selected (please refer to Appendix A). Also, the connections listed on 5.1 for “Quadrature encoder with Buzzer” have to be made.

After the Quadrature Test selection has been made, the console will display the screen shown in Figure 7.

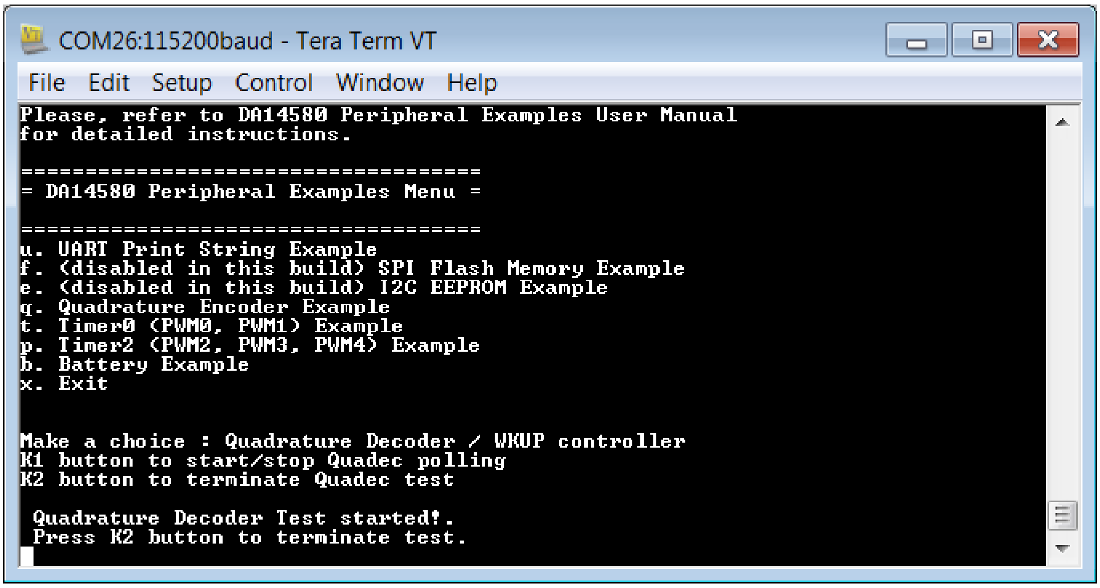

Figure 7: Quadrature Encoder example

If the rotary encoder is activated (e.g. the mouse wheel is scrolled and the rotary encoder is attached to a mouse) the quadrature decoder-wakeup timer interrupt will be triggered, and after each trigger the terminal screen will report the axes relative coordinates. In this configuration, only the X channel terminals are configured and connected (see Figure 8).

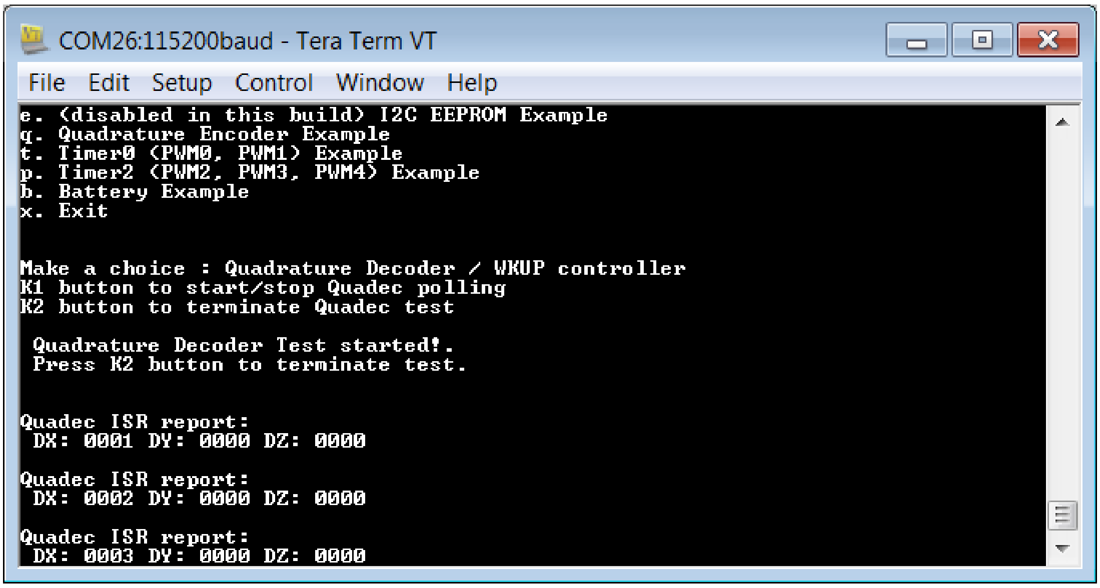

Figure 8: Quadrature Encoder ISR-only reports

If at any time the K1 button is pressed (the user should make sure that correct jumper configuration for buttons K1 and K2 is selected, as described in Table 1), polling of the relative coordinates will be enabled. Then, the terminal window will start polling the quadrature decoder driver (see Figure 9). If the quadrature encoder is activated, then a mixture of ISR and polling generated reports are displayed (see Figure 10).

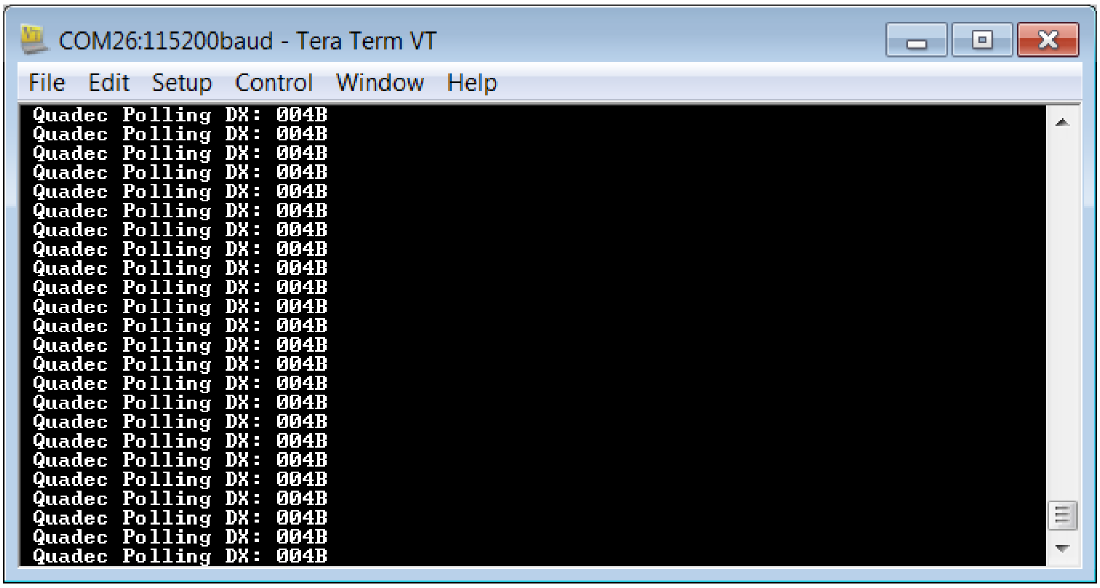

Figure 9: Quadrature Encoder Polling-only reports

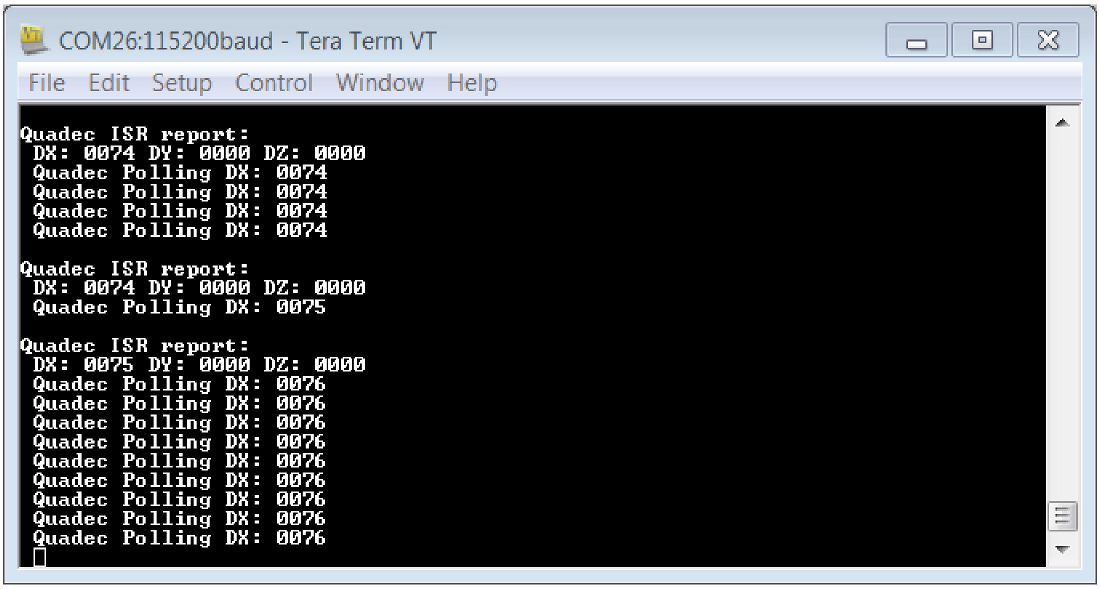

Figure 10: Quadrature Encoder Polling and ISR reports

The polling can be stopped by pressing K1 button again. To terminate the test, one can press K2 at any time. The message “Quadrature Decoder Test terminated!” will be printed.

If the count of events needs to be changed before an interrupt is triggered, the parameter QDEC_EVENTS_COUNT_TO_INT in *periph_setup.h* can be modified accordingly. In the same file, one can change the clock divisor of the quadrature decoder by altering the parameter QDEC_CLOCK_DIVIDER.

The source code for this example is located in function *quad_decoder_test* inside *quad_decoder_test.c*.

5.4.5 TIMER0 (PWM0, PWM1) example

In order for this test to be configured correctly, the hardware configuration *Quadrature Encoder, Timers & Buzzer with UART* has to be selected (please refer to Appendix A).

After the Timer0 (PWM0, PWM1) example has been selected, PWM0 and PWM1 signals will be started, producing an audible melody if a buzzer is connected in P02 or P03. While the melody is playing, stars are being drawn in the terminal window on each beat (interrupt handling - Figure 11).

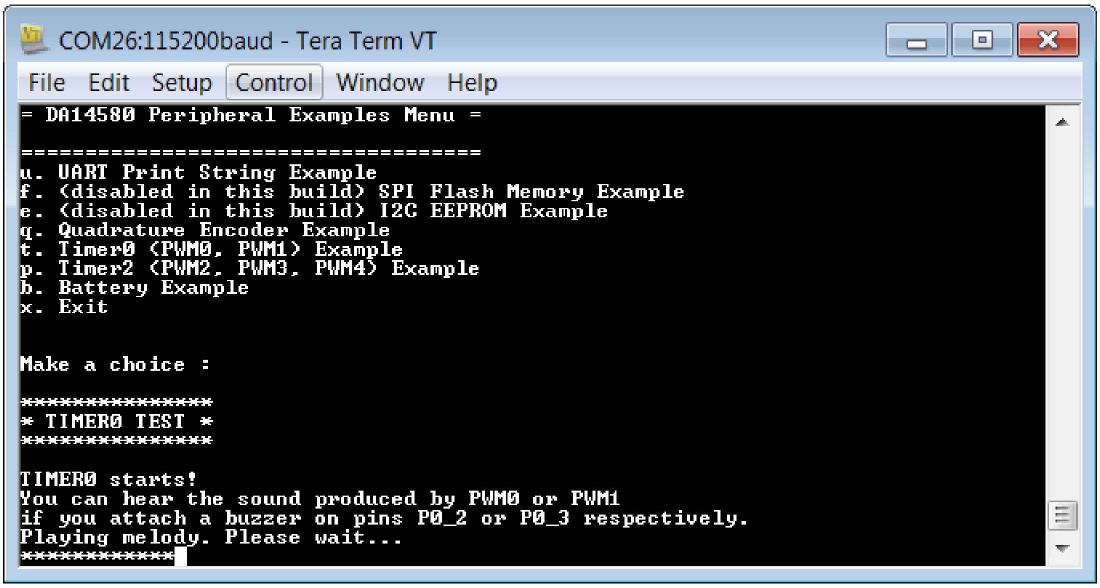

Figure 11: TIMER0 (PWM0, PWM1) test running

Upon completion, a prompt to go back to the main menu or to exit the examples will appear.

The source code for this example is located in function *timer0_test* inside *pwm_test.c*.

5.4.6 TIMER2 (PWM2, PWM3, PWM4) example

In order for this test to be configured correctly, the hardware configuration *Quadrature Encoder, Timers & Buzzer with UART* has to be selected (please refer to Appendix A).

After the Timer2 (PWM2, PWM3, PWM4) example has been selected, PWM2, PWM3 and PWM4 signals will be started. If the jumper configuration described in section 5.1 has been made, there will be a visible indication on segments of the LEDs D1 and D2, as their brightness will be directly influenced by the PWM signals' duty cycle. The screen shown in Figure 12 will appear.

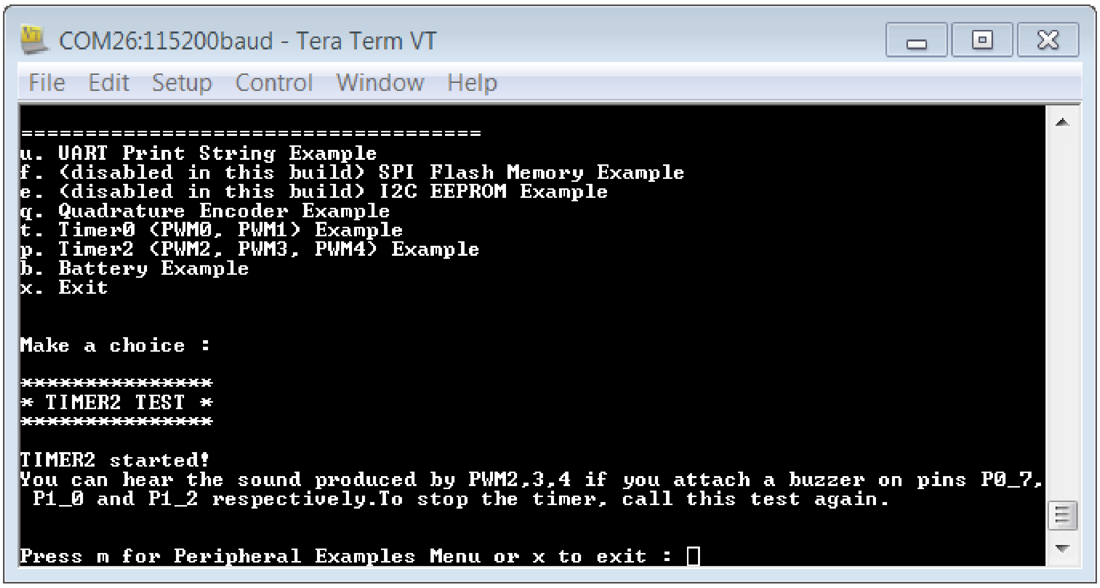

Figure 12: TIMER2 (PWM0, PWM1) test running

For the PWM signals to be stopped, one should go to the Examples Menu and select the test again. The screen shown in Figure 13 will then appear.

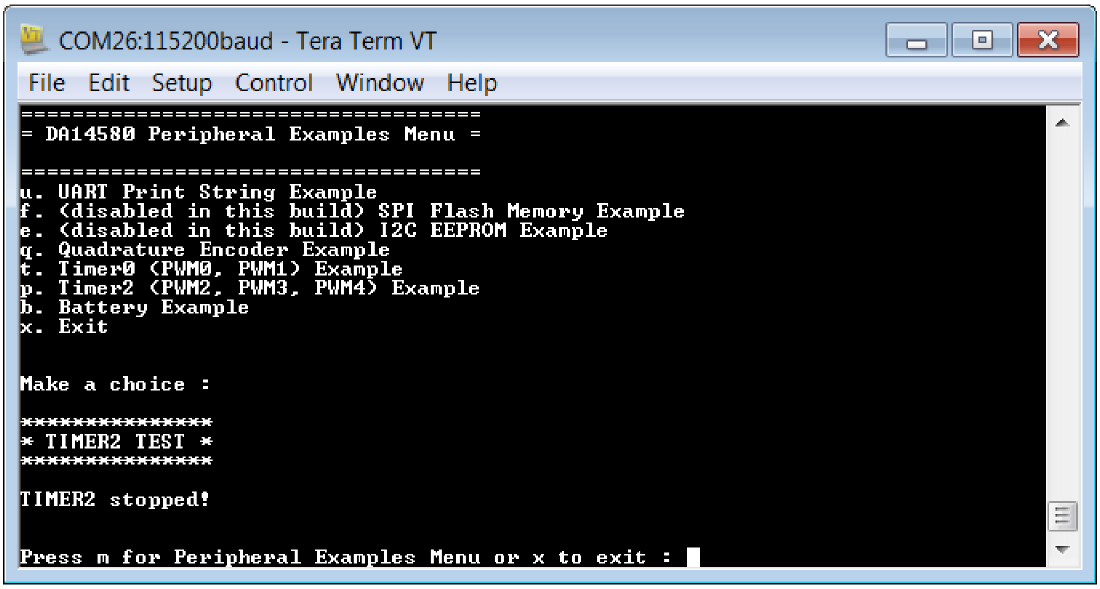

Figure 13: TIMER2 (PWM0, PWM1) test stopped

To change the duty cycle of the PWM signals, one should alter the values passed to the *timer2_set_pwm2_duty_cycle*, *timer2_set_pwm3_duty_cycle* and *timer2_set_pwm4_duty_cycle* functions in *timer2_test* which can be found in *pwm_test.c*. The triple PWM frequency has been previously set, using *timer2_init* function. For further details, one should consult [1] and [3].

The source code for this example is located in function *timer2_test* inside *pwm_test.c*.

5.4.7 Battery example

When the Battery example is selected, the ADC is configured to provide a measurement of the battery level. The percentage left indication calculated for the coin-cell battery CR2032 is displayed on the terminal screen (see Figure 14 below).

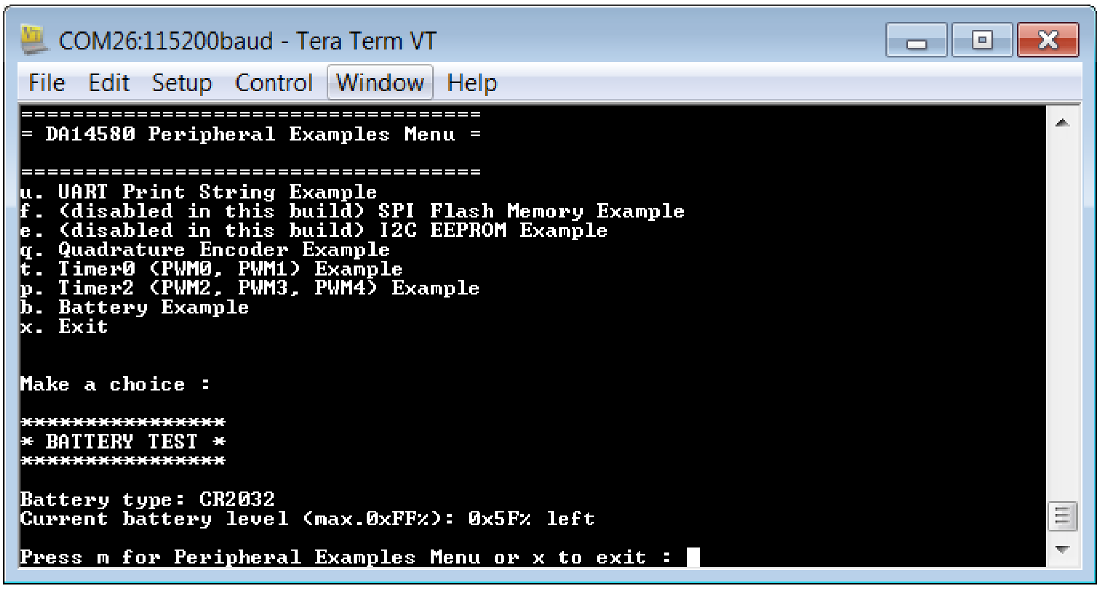

Figure 14: Battery Example

This example can be verified using an external voltage source (well-stabilized in the range 2.5 to 3V) instead of a 3V battery. The DK must have been configured for 3V operation [2]. Then, the jumper from connector J13 should be removed and the GND of the source should be connected to the DK's GND, e.g. to J1 PIN12. The positive terminal of the external voltage source must be connected to J13 PIN3.

CAUTION: the voltage of the external source must never exceed 3V.

For details about the configuration of the ADC, one should consult [1]

The source code for this example is located in function *batt_test* inside *DA14580_examples.c*.

Appendix A How to select Hardware Setup configuration

In order for some of the tests to run correctly, the associated hardware configuration has to be set. This can be accomplished in the file *periph_setup.h*, found in *Include Files* folder, either:

a. Using the uVision Configuration Wizard

Open the file *periph_setup.h* and in *Configuration Wizard* set the desired Hardware Configuration Option:

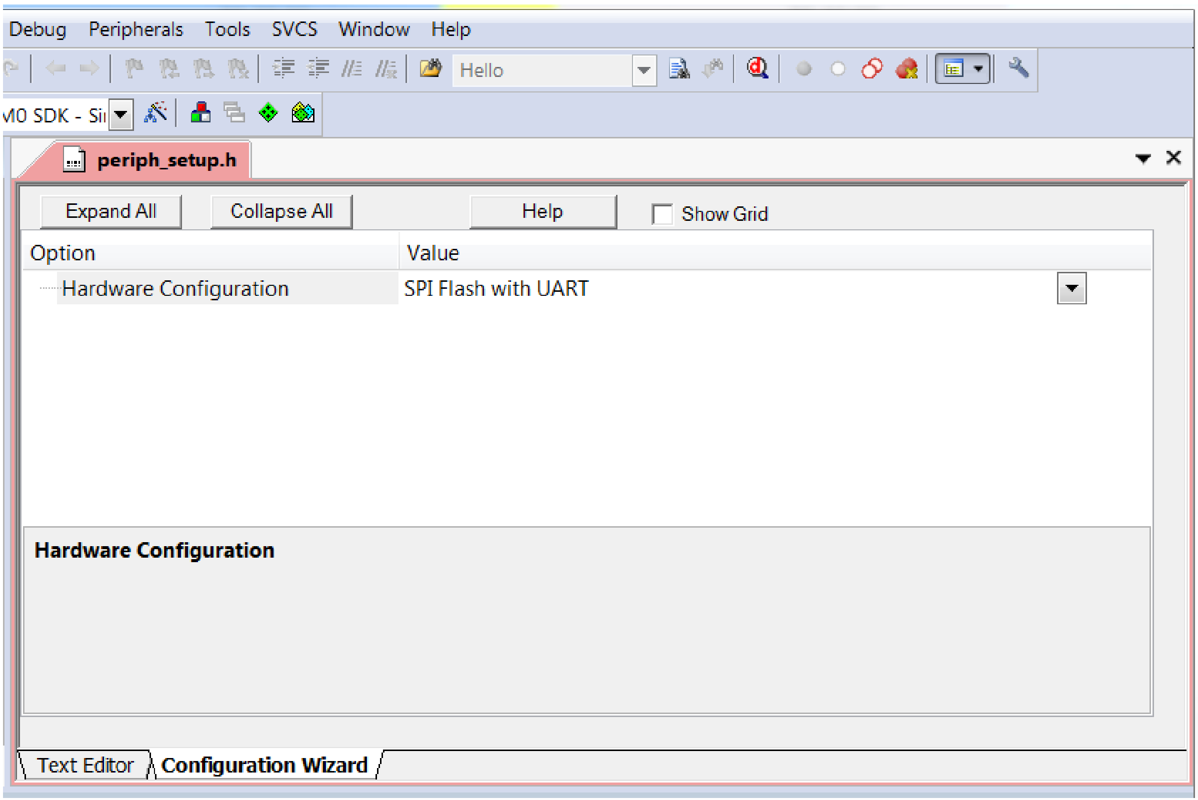

Figure 15: The Configuration Wizard

Or

b. Using the uVision Text Editor

In case Keil uVision version does not include the Configuration Wizard, edit the *periph_setup.h* file using the text editor. Setting the number in parentheses found in the following part of the code

```
// <o> Hardware Configuration <0=> SPI Flash with UART <1=> I2C EEPROM with
UART <2=> Quadrature Encoder, Timers & Buzzer with UART
#define HARDWARE_CONFIGURATION_INDEX (0)
```

to the desired value as follows:

0: SPI Flash with UART

1: I2C EEPROM with UART

2: Quadrature Encoder, Timers & Buzzer with UART

Finally, build the project (F7).

Please note: The examples that are not configured to run in the selected build, are included in the menu preceded by the label: (disabled in this build).

Appendix B SPI Flash/I2C EEPROM Dual PCB schematic

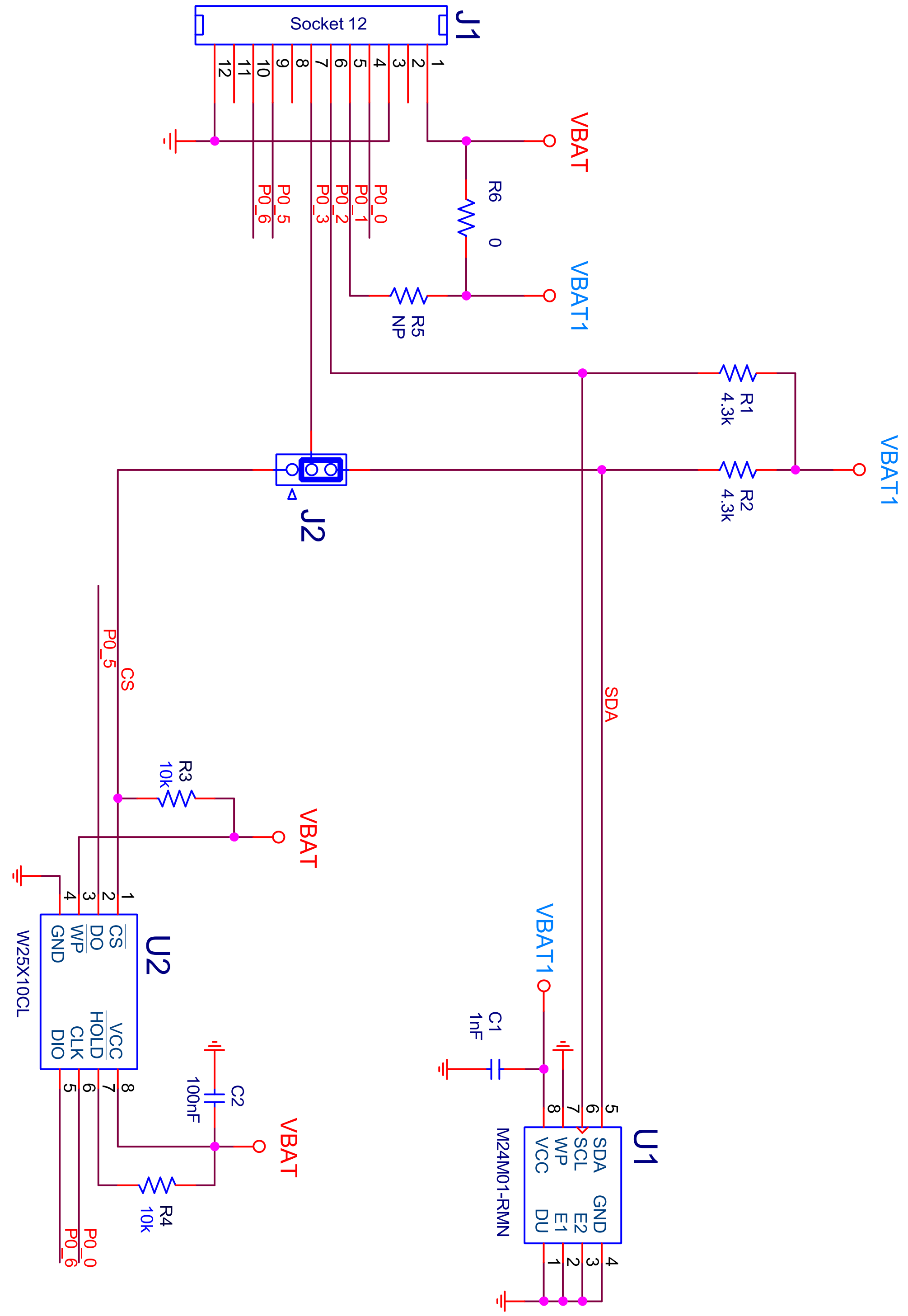

Figure 16: SPI Flash/I2C EEPROM Dual PCB schematic

6 Revision history

Revision	Date	Description
1.0	21-Mar-2014	Initial version.

Status definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

Disclaimer

RoHS Compliance

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2).
Dialog Semiconductor's statement on RoHS can be found on the customer portal https://support.diasemi.com/. RoHS certificates from our suppliers are available on request.

Contacting Dialog Semiconductor

Germany Headquarters
Dialog Semiconductor GmbH
Phone: +49 7021 805-0

United Kingdom
Dialog Semiconductor (UK) Ltd
Phone: +44 1793 757700

The Netherlands
Dialog Semiconductor B.V.
Phone: +31 73 640 8822

Email:
enquiry@diasemi.com

North America
Dialog Semiconductor Inc.
Phone: +1 408 845 8500

Japan
Dialog Semiconductor K. K.
Phone: +81 3 5425 4567

Taiwan
Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

Singapore
Dialog Semiconductor Singapore
Phone: +65 64 849929

China
Dialog Semiconductor China
Phone: +86 21 5178 2561

Korea
Dialog Semiconductor Korea
Phone: +82 2 3469 8291