TivaWare™ Examples

USER'S GUIDE

SW-TM4C-EXAMPLES-UG-2.1.4.178

Copyright

Texas Instruments
108 Wild Basin, Suite 350
Austin, TX 78746
www.ti.com/tiva-c

Revision Information

This is version 2.1.4.178 of this document, last updated on February 22, 2017.

Table of Contents

1 Introduction

Texas Instruments® TivaWare™ software provides code examples in two different locations. The first type of code example is specific to a particular board and is found in the **examples/boards** directory. The examples in this directory can be recomplied, downloaded and run on the specified board without modification. For more information on these examples, refer to the specific Board Firmware Development Package User's Guide.

The second type of example applies to all Tiva™ microcontrollers with a particular peripheral and can be found in the **examples/peripherals** directory. These examples are small, single-purpose code segments that are meant to clearly and simply demonstrate a specific feature and must be customized to run on a particular board.

This document describes the examples available in the **examples/peripherals** directory. Not every example can run on every Tiva device; consult the device data sheet to determine if a particular feature is present. Furthermore please note: THESE EXAMPLES ARE NOT READY TO RUN PROJECTS. For ready-to-run projects please see the **examples/boards** directory.

2 Peripheral Examples

Examples are organized by peripheral in the following sections. Each peripheral section contains a brief description of each example. They are located in your TivaWare installation under the **examples/peripherals** directory, where there is a separate sub-directory for each peripheral.

Note that these examples are different and separate from the board specific examples that you will find under the **examples/boards** directory and which are documented elsewhere.

2.1 ADC Examples

2.1.1 Differential ADC (differential)

This example shows how to setup ADC0 as a differential input and take a single sample between AIN0 and AIN1. The value of the ADC is read and printed to the serial port.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- ADC0 peripheral
- GPIO Port E peripheral (for ADC0 pins)
- AIN0 - PE3
- AIN1 - PE2

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of the ADC.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- None.

2.1.2 Single Ended ADC (single_ended)

This example shows how to setup ADC0 as a single ended input and take a single sample on AIN0/PE3.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- ADC0 peripheral
- GPIO Port E peripheral (for AIN0 pin)

- AIN0 - PE3

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of the ADC.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- None.

2.1.3 ADC Temperature Sensor (temperature_sensor)

This example shows how to setup ADC0 to read the internal temperature sensor.

NOTE: The internal temperature sensor is not calibrated. This example just takes the raw temperature sensor sample and converts it using the equation found in the LM3S9B96 datasheet.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- ADC0 peripheral

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of the ADC.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- None.

2.2 CAN Examples

2.2.1 Multiple CAN RX (multi_rx)

This example shows how to set up the CAN to receive multiple CAN messages using separate message objects for different messages, and using CAN ID filtering to control which messages are received. Three message objects are set up to receive 3 of the 4 CAN message IDs that are used by the multi_tx example. Filtering is used to demonstrate how to receive only specific messages, and therefore not receiving all 4 messages from the multi_tx example. As messages are received the content of each are printed to the serial console.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- CAN0 peripheral
- GPIO port B peripheral (for CAN0 pins)
- CAN0RX - PB4
- CAN0TX - PB5

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of CAN.

- GPIO port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- INT_CAN0 - CANIntHandler

2.2.2 Multiple CAN TX (multi_tx)

This example shows how to set up the CAN to send multiple messages. The CAN peripheral is configured to send messages with 4 different CAN IDs. Two of the messages (with different CAN IDs) are sent using a shared message object. This shows how to reuse a message object for multiple messages. The other two messages are sent using their own message objects. All four messages are transmitted once per second. The content of each message is a test pattern. A CAN interrupt handler is used to confirm message transmission and count the number of messages that have been sent.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- CAN0 peripheral
- GPIO Port B peripheral (for CAN0 pins)
- CAN0RX - PB4
- CAN0TX - PB5

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of CAN.

- GPIO port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- INT_CAN0 - CANIntHandler

2.2.3 Simple CAN RX (simple_rx)

This example shows the basic setup of CAN in order to receive messages from the CAN bus. The CAN peripheral is configured to receive messages with any CAN ID and then print the message contents to the console.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- CAN0 peripheral
- GPIO port B peripheral (for CAN0 pins)
- CAN0RX - PB4
- CAN0TX - PB5

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of CAN.

- GPIO port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- INT_CAN0 - CANIntHandler

2.2.4 Simple CAN TX (simple_tx)

This example shows the basic setup of CAN in order to transmit messages on the CAN bus. The CAN peripheral is configured to transmit messages with a specific CAN ID. A message is then transmitted once per second, using a simple delay loop for timing. The message that is sent is a 4 byte message that contains an incrementing pattern. A CAN interrupt handler is used to confirm message transmission and count the number of messages that have been sent.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- CAN0 peripheral
- GPIO Port B peripheral (for CAN0 pins)
- CAN0RX - PB4
- CAN0TX - PB5

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of CAN.

- GPIO port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- INT_CAN0 - CANIntHandler

2.3 EPI Examples

2.3.1 EPI SDRAM Mode (sdram)

This example shows how to configure the TM4C129 EPI bus in SDRAM mode. It assumes that a 64Mbit SDRAM is attached to EPI0.

For the EPI SDRAM mode, the pinout is as follows: Address11:0 - EPI0S11:0 Bank1:0 - EPI0S14:13 Data15:0 - EPI0S15:0 DQML - EPI0S16 DQMH - EPI0S17 /CAS - EPI0S18 /RAS - EPI0S19 /WE - EPI0S28 /CS - EPI0S29 SDCKE - EPI0S30 SDCLK - EPI0S31

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- EPI0 peripheral
- GPIO Port A peripheral (for EPI0 pins)
- GPIO Port B peripheral (for EPI0 pins)
- GPIO Port C peripheral (for EPI0 pins)
- GPIO Port G peripheral (for EPI0 pins)
- GPIO Port K peripheral (for EPI0 pins)
- GPIO Port L peripheral (for EPI0 pins)
- GPIO Port M peripheral (for EPI0 pins)
- GPIO Port N peripheral (for EPI0 pins)
- EPI0S0 - PK0
- EPI0S1 - PK1
- EPI0S2 - PK2
- EPI0S3 - PK3
- EPI0S4 - PC7
- EPI0S5 - PC6
- EPI0S6 - PC5
- EPI0S7 - PC4
- EPI0S8 - PA6
- EPI0S9 - PA7
- EPI0S10 - PG1
- EPI0S11 - PG0
- EPI0S12 - PM3
- EPI0S13 - PM2
- EPI0S14 - PM1
- EPI0S15 - PM0
- EPI0S16 - PL0
- EPI0S17 - PL1

- EPI0S18 - PL2
- EPI0S19 - PL3
- EPI0S28 - PB3
- EPI0S29 - PN2
- EPI0S30 - PN3
- EPI0S31 - PK5

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of EPI0.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- None.

2.4 I2C Examples

2.4.1 I2C Master Loopback (i2c_master_slave_loopback)

This example shows how to configure the I2C0 module for loopback mode. This includes setting up the master and slave module. Loopback mode internally connects the master and slave data and clock lines together. The address of the slave module is set in order to read data from the master. Then the data is checked to make sure the received data matches the data that was transmitted. This example uses a polling method for sending and receiving data.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- I2C0 peripheral
- GPIO Port B peripheral (for I2C0 pins)
- I2C0SCL - PB2
- I2C0SDA - PB3

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of I2C.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- None.

2.4.2 Slave Receive Interrupt (slave_receive_int)

This example shows how to configure a receive interrupt on the slave module. This includes setting up the I2C0 module for loopback mode as well as configuring the master and slave modules. Loopback mode internally connects the master and slave data and clock lines together. The address of the slave module is set to a value so it can receive data from the master.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- I2C0 peripheral
- GPIO Port B peripheral (for I2C0 pins)
- I2C0SCL - PB2
- I2C0SDA - PB3

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of I2C.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- INT_I2C0 - I2C0SlaveIntHandler

2.4.3 SoftI2C AT24C08A EEPROM (soft_i2c_atmel)

This example shows how to configure the SoftI2C module to read and write an Atmel AT24C08A EEPROM. A pattern is written into the first 16 bytes of the EEPROM and then read back.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- Timer0 peripheral (for the SoftI2C timer)
- GPIO Port B peripheral (for SoftI2C pins)
- PB2 (for SCL)
- PB3 (for SDA)

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of I2C.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application, you must add these interrupt handlers to your vector table.

- INT_TIMER0A - Timer0AIntHandler

2.5 LCD Controller Examples

2.5.1 LCD Controller Raster Mode Example (raster_example)

This application illustrates the use of the Tivaware Graphics Library and Tiva TM4C129x LCD controller driving an 800x480 display using raster (HSYNC/VSYNC/ACTIVE/DATA) mode. The display is initialized and enabled then a simple pattern including lines, a small image, some text and a circle is displayed.

By default, the application is set up to support an Innolux EJ090NA-03A display with 800x480 resolution, refreshed at 60Hz from a 16bpp frame buffer stored in SDRAM. The SDRAM is attached to the MCU via the External Peripheral Interface (EPI) module. The file drivers/raster_displays.c contains timings and initialization functions for several other displays and the application can be easily rebuilt to support any of these by replacing the preprocessor define "**INNOLUX_DISPLAY**" with one of the other display labels:

- **OPTREX_DISPLAY** supports an Optrex T-55226D043J-LW-A-AAN in 800x480 with 75Hz refresh rate.
- **LXD_DISPLAY** supports an LXD M7170A in 640x480 with 60Hz refresh rate.
- **FORMIKE_DISPLAY** supports at Formike KWH070KQ13 in 800x480 with 60Hz refresh rate.

Display interface timing information and any required initialization code is included in the file lcd/drivers/raster_displays.c. New raster-mode displays can be added to this file and raster_displays.h very easily and used by the application merely by adding another display label and appropriate code to set the tRasterDisplayInfo timing structure for that display at the top of raster_example.c.

Once appropriate display timings have been determined, the display can be used by the TivaWare Graphics Library via one of the supplied raster mode display drivers. Four distinct drivers are supplied in the lcd/drivers directory, each supporting a different color depth for the frame buffer:

- **grlib_raster_driver_1bpp.c** supports a monochrome (2 color) display buffer.
- **grlib_raster_driver_4bpp.c** supports a 4 bit per pixel (16 color) frame buffer.
- **grlib_raster_driver_8bpp** supports an 8 bit per pixel (256 color) frame buffer.
- **grlib_raster_driver_16bpp** supports a 16 bit per pixel (65536 color) frame buffer.

The size of frame buffer required varies with the resolution of the LCD display in use and the desired frame buffer color depth. Note that the frame buffer color depth may be lower than the native color resolution of the LCD panel - the LCD controller makes use of a color lookup table or palette to convert the pixels in the frame buffer to the correct color format for the LCD's hardware interface.

The size of frame buffer, in bytes, can be determined using the following formula:

Buffer Size = X $*$ Y $*$ (BPP / 8) + (Header Size)

where:

- X is the horizontal pixel resolution of the LCD panel
- Y is the vertical pixel resolution of the LCD panel
- BPP is the desired number of bits per pixel for the frame buffer
- Header Size is 512 for 8bpp frame buffers or 32 for all other color resolutions.

The frame buffer header contains information informing the LCD controller of the pixel format in the frame buffer and also the color lookup table used for 1, 4 and 8bpp cases. Note that a 32 byte header is still required even when using 16bpp frame buffers which do not require a color lookup table.

For large panels such as those described in raster_displays.h, a frame buffer supporting more than two colors is likely to be too large to fit in the internal memory of a TM4C129x device and would, therefore, require the use of external, EPI-connected SDRAM. The 16bpp 800x480 frame buffer used in this application requires almost 940KB of RAM for example. For lower resolution displays or lower color depths, internal SRAM may be suitable for use as the frame buffer. For example, a 16bpp QVGA (320x240) frame buffer occupies about 150KB of storage and a monochrome (1bpp) 800x480 frame buffer needs only 48KB.

2.6 PWM Examples

2.6.1 PWM dead-band (dead_band)

This example shows how to setup the PWM0 block with a dead-band generation.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- GPIO Port B peripheral (for PWM pins)
- M0PWM0 - PB6
- M0PWM1 - PB7

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of the PWM.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- None.

2.6.2 PWM Invert (invert)

This example shows how to setup PWM0 using the inverted output function. This feature allows you to invert the polarity of the PWM output. This example is setup to invert a 25% duty cycle to get a 75% duty cycle every 5 seconds.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- GPIO Port B peripheral (for PWM0 pin)
- PWM0 - PB6

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of the PWM.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- None.

2.6.3 PWM Reload Interrupt (reload_interrupt)

This example shows how to setup an interrupt on PWM0. This example demonstrates how to setup an interrupt on the PWM when the PWM timer is equal to the configurable PWM0LOAD register.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- GPIO Port B peripheral (for PWM0 pin)
- PWM0 - PB6

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of the PWM.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- INT_PWM0_0 - PWM0IntHandler

2.7 ROM Examples

2.7.1 Direct ROM Function Calls (rom_direct)

This example shows how to directly call a ROM based driver library function using the **ROM_** prefix on the driver library function name. When you call a ROM function in this way, it will only work on a part with ROM, and you will have to change it to work with a non-ROM part.

2.7.2 Mapped ROM Function Calls (rom_mapped)

This example shows how to map ROM function calls at compile time to use a ROM function if available on the part, or a library call if the function is not available in ROM. This allows you to write code that can be used on either a part with ROM or without ROM without needing to change the code. The mapping is performed at compile time and there is no performance penalty for using the mapped method instead of the direct method. Mapped ROM functions are called with a **MAP_** prefix on the driver library function name.

2.8 SSI/SPI Examples

2.8.1 SoftSSI Master (soft_spi_master)

This example shows how to configure the SoftSSI module. The code will send three characters on the master Tx then polls the receive FIFO until 3 characters are received on the master Rx.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- GPIO Port A peripheral (for SoftSSI pins)
- SoftSSIClk - PA2
- SoftSSIFss - PA3
- SoftSSIRx - PA4
- SoftSSITx - PA5

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of SoftSSI.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- SysTickIntHandler

Note:

This example provide the same functionality using the same pins as the spi_master example. As such, it can be used as a guide for how to convert code which uses hardware SSI to the SoftSSI module.

2.8.2 SPI Master (spi_master)

This example shows how to configure the SSI0 as SPI Master. The code will send three characters on the master Tx then polls the receive FIFO until 3 characters are received on the master Rx.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- SSI0 peripheral
- GPIO Port A peripheral (for SSI0 pins)
- SSI0Clk - PA2
- SSI0Fss - PA3
- SSI0Rx - PA4
- SSI0Tx - PA5

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of SSI0.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- None.

2.8.3 TI Master (ti_master)

This example shows how to configure the SSI0 as TI Master. The code will send three characters on the master Tx then poll the receive FIFO until 3 characters are received on the master Rx.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- SSI0 peripheral
- GPIO Port A peripheral (for SSI0 pins)
- SSI0Clk - PA2
- SSI0Fss - PA3
- SSI0Rx - PA4
- SSI0Tx - PA5

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of I2C0.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- None.

2.9 System Control Examples

2.9.1 System Clock Configuration with PLL (system_clock_pll)

This example shows how to set up the system clock to use the PLL.

2.10 System Tick Timer (SysTick) Examples

2.10.1 Systick Interrupt (systick_int)

This example shows how to configure the SysTick and the SysTick interrupt.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- NONE

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of Systick.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- SysTickIntHandler

2.11 General Purpose Timer Examples

2.11.1 Timer Edge Count (edge_count)

This example application demonstrates the use of a general purpose timer in down edge count mode. Timer 4 is configured to decrement each time a rising edge is seen on PM0/CCP0. The count is initialized to 9 and the match is set to 0, causing an interrupt to fire after 10 positive edges detected on the CCP pin.

2.11.2 16-Bit One-Shot Timer (oneshot_16bit)

This example shows how to configure Timer0B as a one-shot timer with a single interrupt triggering after 1ms.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- TIMER0 peripheral

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of Timer0.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- INT_TIMER0B - Timer0BIntHandler

2.11.3 16-Bit Periodic Timer (periodic_16bit)

This example shows how to configure Timer0B as a periodic timer with an interrupt triggering every 1ms. After a certain number of interrupts, the Timer0B interrupt will be disabled.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- TIMER0 peripheral

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of Timer0.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- INT_TIMER0B - Timer0BIntHandler

2.11.4 PWM using Timer (pwm)

This example shows how to configure Timer1B to generate a PWM signal on the timer's CCP pin.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- TIMER1 peripheral
- GPIO Port B peripheral (for T1CCP1 pin)

- T1CCP1 - PB5

The following UART signals are configured only for displaying console messages for this example. These are not required for operation of Timer0.

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

This example uses the following interrupt handlers. To use this example in your own application you must add these interrupt handlers to your vector table.

- None.

2.12 UART Examples

2.12.1 UART Loopback (uart_loopback)

This example demonstrates the use of a UART port in loopback mode. On being enabled in loopback mode, the transmit line of the UART is internally connected to its own receive line. Hence, the UART port receives back the entire data it transmitted.

This example echoes data sent to the UART's transmit FIFO back to the same UART's receive FIFO. To achieve this, the UART is configured in loopback mode. In the loopback mode, the Tx line of the UART is directly connected to its Rx line internally and all the data placed in the transmit buffer is internally transmitted to the Receive buffer.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board.

- UART7 peripheral - For internal Loopback
- UART0 peripheral - As console to display debug messages.
  - UART0RX - PA0
  - UART0TX - PA1

UART parameters for the UART0 and UART7 port:

- Baud rate - 115,200
- 8-N-1 operation

2.12.2 UART Polled I/O (uart_polled)

This example shows how to set up the UART and use polled I/O methods for transmitting and receiving UART data. The example receives characters from UART0 and retransmits the same character using UART0. It can be tested by using a serial terminal program on a host computer. This example will echo every character that is type until the return/enter key is pressed.

This example uses the following peripherals and I/O signals. You must review these and change as needed for your own board:

- UART0 peripheral
- GPIO Port A peripheral (for UART0 pins)
- UART0RX - PA0
- UART0TX - PA1

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265