QuickStartOMAPL1x rCSL

This article applies to the following devices:

- OMAP-L137 / OMAP-L138
- TMS320C6748 / TMS320C6747 / TMS320C6746 / TMS320C6745 / TMS320C6743 / TMS320C6742
- AM1808 / AM1806 / AM1707 / AM1705

Contents

Overview

quickStartOMAPL1x contains CCSv4 based system examples developed on the OMAP-L137/OMAP-L138 EVM showcasing the register Chip Support Library (rCSL) macros supported in the BIOS Platform Support Package (PSP) 1.30 release. These examples are intended to showcase the rCSL macros and their various use with different OMAP-L1x based applications which do not require the support of an operating system (DSP/BIOS, SYSBIOS, Linux, Windows CE etc...).

Additionally, these examples serve as basic foundational level programming to OMAP-L1x Software developers looking to better understand the inter chip communication between the OMAP-L138 CPUs and various peripherals. Specifically, the examples include basic functionality such as using the System Configuration module to control pin multiplexing, ARM & DSP integration, demonstrating simple steps to enable peripheral clocking with the Power and Sleep Controller, illustrating how to use the DSP and ARM Interrupt Controllers to service interrupts, and highlighting the key features of the supported peripherals. These examples will be beneficial for users looking for:

- Low level DSP or ARM examples
- Examples that do not use an Operating System (OS)
- DSP or ARM interrupt setup and handling
- Peripheral Setup/Functionality

Suggested Minimum Requirements

The following list contains the **suggested** minimum requirements for rebuilding the OMAP-L1x rCSL examples. The examples provided in quickStartOMAPL1x were not tested under these conditions. For the exact conditions in which these examples were tested, please review the section titled ***Test Conditions***.

- **Hardware**
  - PC with Microsoft Windows XP
  - OMAP-L137 Starter Kit Board (Silicon Rev 2.x)
  - OMAP-L138 Baseboard + SOM (Silicon Rev 1.x)
  - XDS Emulation Hardware
    - XDS100v1 USB Emulator
    - XDS100v2 USB Emulator (Blackhawk USB100v2 Model D with firmware v2010.07.20)
    - XDS510 USB Emulator
  - USB 2.0 Compliant Cable
  - RS-232 Serial Cable
  - 5 Volt Power Supply
- **Software**
  - Code Composer Studio Version 4.x
  - Code Generation Tools (TMS470/C6000)
  - Terminal Emulator
    - HyperTerminal
    - Tera Term

Downloads

Version 2.0: quickStartOMAPL1x_rCSL_2.0 (http://software-dl.ti.com/dsps/dsps_public_sw/omap_l1x/quickStartOMAPL1x_rCSL-2.0-Setup.zip)

Version 1.0 (legacy): quickStartOMAPL1x_rCSL_1.0 (http://software-dl.ti.com/dsps/dsps_public_sw/omap_l1x/quickStartOMAPL1x_rCSL_1.0.zip)

Installation Note: Choosing a installation directory other than the default will require a change in the respective CCSv4 Project Build Properties. It is recommended that you choose the default installation path.

---

rCSL Examples

This section describes the available examples for both the OMAPL137 and OMAPL138 EVMs, the suggested minimum hardware/software requirements, the conditions in which the examples were tested on, and an overview of each individual example.

Platforms Supported

Examples	OMAPL137 EVM	OMAPL138 EVM
GPIO_multi_led_interrupt*	Yes	No
I2C_eeprom	Yes	No
I2C_io_expander	No	Yes
TIMER_comparator*	Yes	No
TIMER_interrupt*	No	Yes
TIMER_led_blink_frequency*	Yes	No
TIMER_watchdog*	Yes	No
UART_hyperterminal*	Yes	Yes
UART_interrupt_loopback*	Yes	Yes
McASPEcho*	No	Yes
ARM2DSP_integration*	Yes	Yes
CACHE_dspLib_fft	Yes	Yes
EDMA_event_trig*	Yes	Yes
EDMA_manual_trig*	Yes	Yes
EDMA_ping_pong*	Yes	Yes
EDMA_self_chain*	Yes	Yes

* Example uses interrupts
Note: All examples are available for both the DSP and ARM

Example Synopsis

The following text provides an overview description of the available examples.

- **McASPEcho:** - The McASP Echo register Chip Support Library (rCSL) example provides a non-operating system based example of how to use the C674x Digital Signal Processor (DSP) in conjunction with the Multi-Channel Audio Serial Port (McASP) to receive and transmit 24 bit audio data using the industry standard Inter-Integrated Sound (I2S) Protocol to/from the TLV320AIC3106 Audio Coder / Decoder (CODEC).While no audio processing takes place in the OMAP-L138, this example provides a general understanding of the necessary CPU/peripheral configurations that need to be implemented on the OMAP-L138 Low Power Applications Processor.

- **ARM2DSP_integration:** - Using the rCSL macros, this example demonstrates interprocessor communication by following a typical ARM-DSP sequence. The ARM begins the communication by modifying shared variables in the shared memory. Next, the ARM interrupts the DSP using the chip signal register (CHIPSIG) and its corresponding system interrupt. The interrupt thus notifies the DSP that there are pending requests. The DSP then responds to the interrupt by processing the requests stored in shared memory and executing any tasks. Upon completing all requests, the DSP interrupts the ARM.

- **CACHE_dspLib_fft:** - This example initializes the cache using the rCSL macros to configure the L1, L2, and MAR bits to enhance the performance of source code located in external memory. To illustrate the performance enhancement, the example uses the C674x DSP Library assembly function DSPF_sp_fftSPxSP as a benchmark test case. The DSP function is tested with the cache disabled, with the MAR bits set as well as L1 enabled, and with the MAR bits, L1, and L2 enabled. The number of clock cycles to compute the function under each condition is printed to the console.

- **EDMA_event_trig:** - This example initializes the EDMA3 channel controller using the rCSL macros to perform a simple 1kB internal memory to memory transfer upon a timer time-out event. Once initialized, the timer is enabled once to trigger the EDMA3 event. The EDMA3 channel controller consequently sends a transfer request to the transfer controller and the data in the source buffer is transferred to the destination buffer. Upon completion of the transfer, the EDMA3 channel controller sends an interrupt to the CPU and the source and destination buffers are compared to verify the EDMA3 transfer. The results are printed to the console.

- **EDMA_manual_trig:** - This example initializes the EDMA3 channel controller using the rCSL macros to perform a simple 1kB internal memory to memory transfer. Once initialized, the channel controller is manually triggered to send a transfer request to the transfer controller. Upon verification that the transfer request is non-null, the data in the source buffer is transferred to the destination buffer. Once the transfer finishes, the EDMA3 channel controller sends an interrupt to the CPU and the source and destination buffers are compared to verify the EDMA3 transfer. The results are printed to the console.

- **EDMA_ping_pong:** - This example initializes the EDMA3 channel controller using the rCSL macros to perform two simple 1kB internal memory to memory transfers. These transfers are broken up into smaller packets (128 byte transfers), allowing the EDMA3 peripheral to alternate between the two larger transfers. Although not implemented, the CPU could service the previously transferred 128 byte packet while the next packet is transferred by the EDMA3 peripheral. After the peripherals are initialized, the timer is enabled once to trigger the EDMA3 event. The EDMA3 channel controller consequently sends a transfer request to the transfer controller and the first data packet in the ping source buffer is transferred to the ping destination buffer. The EDMA3 next sends a transfer request to the transfer controller requesting a pong data transfer. This sequence is repeated until all data packets for both the ping and pong source buffers are transferred. Upon completion of the transfer, the EDMA3 channel controller sends an interrupt to the CPU and the source and destination buffers are compared to verify the EDMA3 transfer. The results are printed to the console.

- **EDMA_self_chain:** - This example initializes the EDMA3 channel controller using the rCSL macros to perform a simple 1kB internal memory to memory transfer broken up into 8 separate 128 byte transfers. The initial transfer request is generated by a timer time-out event and is chained to the subsequent requests. After initialization, the timer is enabled once to trigger the EDMA3 event. The EDMA3 channel controller consequently sends a transfer request to the transfer controller and the data in the source buffer is transferred to the destination buffer. Upon completion of the transfer, the EDMA3 channel controller sends an interrupt to the CPU and the source and destination buffers are compared to verify the EDMA3 transfer. The results are printed to the console.

- **GPIO_multi_led_interrupt:** - This example initializes the GPIO peripheral using the rCSL macros to configure six pins as GPIO and bring the GPIO clock out of the sleep mode. Once initialized, the user turns on the LED display by toggling SW3-1 and turns off the LED display by toggling SW3-2. Once the LED display has been disabled, the program finishes execution and exits. The LED display consists of a sequential pattern of enabling each LED (DS1-DS4) in an ascending order where a delay is implemented in between each successive call to enable the next LED. This delay has been implemented to ensure visibility of the LED display to the user.

- **I2C_eeprom:** - This example initializes the I2C peripheral using the rCSL macros. Once initialized, the I2C peripheral acts as a master transmitter and performs a series of page writes to the on-board EEPROM. Upon completion, the I2C is then re-configured as a master receiver and reads back the previously transfered data, comparing the received data against the transfered data. The use of polling unique bits in the I2C status register has been implemented to identify the status of the transmit and receive buffers.

- **I2C_io_expander:** - This example initializes the I2C peripheral using the rCSL macros. Once initialized, the I2C peripheral acts as a master transmitter and writes to the IO Expander to configure the User LED1 as output. The I2C is then re-configured as a master receiver and waits until detecting user input on the DIP switch SW2-1. Once user input has been supplied, the User LED1 will blink five times before the example exits. The use of polling unique bits in the I2C status register has been implemented to identify the status of the transmit and receive buffers.

- **TIMER_comparator:** - This example demonstrates the use of both the GPIO and Timer peripherals and initializes them by using the rCSL macros. The first 64 Bit Timer is configured into two separate 32 Bit timers by declaring the Timer in the 32 Bit Unchain mode; the second 64 Bit Timer is not used. The primary focus of this example is to illustrate the use of the Timer Compare Registers. Once both peripherals have been initialized, the timers are enabled. When the timer counter contains the same value as a compare register, an interrupt occurs and a pre-defined LED sequence is displayed. The total example execution time is 20 seconds with three pre-defined LED displays occurring at 5s, 10s, and 15s.

- **TIMER_interrupt:** - This example initializes the Timer peripheral in the 32 Bit Unchain mode by using the rCSL macros and demonstrates the Timer module as a general purpose Timer. After initialization, the Timer runs continuously counting the number of interrupts that occur. After five interrupts, the example exits.

- **TIMER_led_blink_frequency:** - This example demonstrates the use of both the GPIO and Timer peripherals and initializes them by using the rCSL macros. The first 64 Bit Timer is configured into two separate 32 Bit timers by declaring the Timer in the 32 Bit Unchain mode; the second 64 Bit Timer is not used. The primary focus of this example is to illustrate the Timer as a general purpose timer. In this case, the timer has been utilized to control the rate at which an LED (DS1) blinks and also to implement a delay. Once both peripherals have been initialized, the user can toggle any user switch (SW3-1 through SW3-4) and the LED (DS1) will toggle with a period of 100, 200, 400, or 800 ms (the full blink cycle will be double this time). The user LED (DS1) will continue to blink until the user toggles the same switch subsequently.

- **TIMER_watchdog:** - This example demonstrates the use of both the GPIO and Timer peripherals and initializes them by using the rCSL macros. The first 64 Bit Timer is configured as a watchdog timer while the second 64 Bit Timer is configured into two separate 32 Bit timers by declaring the Timer in the 32 Bit Unchain mode. The primary focus of this example however is to illustrate the Timer as a watchdog. Once both peripherals have been initialized, the user LED (DS1) toggles with a period of 500 ms (or 1 full blink per second). The user LED (DS1) will continue to blink as long as the watchdog is serviced every 10 seconds. If the watchdog has not been serviced within the past 10 seconds, the user LED (DS1) will turn off and the example will exit. To service the watchdog, the user toggles the user switch (SW3-1).

- **UART_hyperterminal:** - This example initializes the UART peripheral using the rCSL macros. Once initialized, the UART peripheral performs a series of data transfers to the HypterTerminal through an RS-232 cable. The example provides the user with an option to read the contents of a register in memory or terminate the debug session. Depending on the user's choices, the example will perform the necessary operations and update the HypterTerminal screen. The use of interrupts has been implemented to update the status of the transmit and receive buffers.

- **UART_interrupt_loopback:** - This example initializes the UART peripheral in the loopback mode using the rCSL macros. Once initialized, the UART peripheral performs a series of data transfers storing the received data in an array. After completing the data transfers, the array containing the received data is compared to the transfered data to ensure module functionality. The use of interrupts has been implemented to update the status of the transmit and receive buffers.

Obtaining Support

In addition to the source file embedded comments, each example contains a project readme located in the project folder. The project readme includes high level project details as well as steps to run the example using CCSv4, detailed descriptions of the main project functions, and references to helpful documentation. A pdf file called ***Useful Documents*** (located in the OMAPL1x/documents folder) contains links (as well as an abstract) to the referenced documents.

Test Conditions

The rCSL examples included in quickStartOMAPL1x were tested under the following conditions:

- **CCS File Version:** 4.1.3.00038

- **Code Generation Tools**

  ARM Examples: TMS470 v4.6.4
  DSP Examples: C6000 v6.1.17

- **EVM Boards**

  OMAPL137

  Part Name: Spectrum Digital OMAP-L137/TMS320C6747 Floating Point Starter Kit
  Model Number: 702210
  Revision: Rev G
  NOTE: Although not tested, the OMAPL137 DSP side examples should be compatible with the c6747 EVM and the OMAPL137 ARM side examples should be compatible with the AM1707 EVM.

  OMAPL138

  Part Name: Logic PD Zoom OMAP-L138 eXperimenter Kit
  Model Number: TMDXL138LOGICEXP
  Baseboard Part Number: 1013220 Rev 2
  SOM Part Number: 1014650 Rev A
  NOTE: Although not tested, the OMAPL138 DSP side examples should be compatible with the c6748 SOM and the OMAPL138 ARM side examples should be compatible with the AM1808 SOM.

- **OMAPL1x Silicon Revision**

  OMAPL137: 2.0
  OMAPL138: 1.1

Troubleshooting/Feedback

For initial debug, try performing a system reset or power cycle to reset the device. If this does not correct any anomalies seen, please post your questions at http://e2e.ti.com. Any comments about the article and the quickStartOMAPL1x application should be posted here.

{{ 1. switchcategory:MultiCore= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **QuickStartOMAPL1x rCSL** here.	Keystone= ▪ For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum ▪ For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **QuickStartOMAPL1x rCSL** here.	C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article* ***QuickStartOMAPL1x rCSL*** *here.*	DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article* ***QuickStartOMAPL1x rCSL*** *here.*	MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article* ***QuickStartOMAPL1x rCSL*** *here.*	OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article* ***QuickStartOMAPL1x rCSL*** *here.*	OMAPL1 *technical OMAP pl your ques The OMA Please p comment article* ***QuickSta rCSL*** *her*

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters

DLP & MEMS
High-Reliability
Interface
Logic
Power Management

Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity

Retrieved from "https://processors.wiki.ti.com/index.php?title=QuickStartOMAPL1x_rCSL&oldid=68521"

This page was last edited on 8 July 2011, at 16:41.