Upgrading to Luminary Micro's Stellaris® Microcontrollers from Microchip's PIC Microcontrollers

Application Note

AN01249-03

Copyright

Texas Instruments
108 Wild Basin, Suite 350
Austin, TX 78746
Main: +1-512-279-8800
Fax: +1-512-279-8879
http://www.luminarymicro.com

Table of Contents

Introduction

It's time to move from the land of the PIC microcontroller into the exciting world of ARM®. As an experienced embedded designer, much of what you will encounter in working with Stellaris ARM® Cortex™-M3 microcontrollers will be familiar—but there are a few things that may require a second look and a trip up the learning curve. This application note covers a broad range of topics that intend to ease the transition.

For many readers some of the information here states the obvious—if that is the case, at least you will know that you are on the right path. This application note targets readers who have designed with PIC12x, 16x, or 18x devices, normally use MPLAB, and code in C with some ASM code as needed. Even if you are moving from another 8-bit platform, many of these pointers will be helpful.

About ARM

Tip 1: Some of the ARM information on the web does not apply to ARM Cortex-M3.

Stellaris microcontrollers contain a standard ARM Cortex-M3 core, licensed from ARM holdings PLC of Cambridge England, memory blocks, and an array of custom peripherals. It is worth noting that the ARM title covers a wide spectrum of licensed cores ("from $1 to 1GHz" as the catch-phrase goes), and, although they share many common attributes, there are differences in implementation. Some older documentation (often found on the web) may pre-date the expansion of the ARM line-up and may not be applicable to all ARM devices. Keep this in mind to avoid confusion, especially when browsing old newsgroups.

Integrated Development Environment (IDE)

Tip 2: Chose an IDE with a familiar look and feel.

These days there are only a few viable options for PIC IDEs; Microchip's own MPLAB is almost ubiquitous. For ARM development the situation is much different, there are many popular IDEs. Not all ARM compilers support Cortex-M3 so check with the vendor or refer to the tool list at www.luminarymicro.com.

The IDE with the closest look and feel to MPLAB is either uVision from Keil (now an ARM company) or Embedded Workbench from IAR System. Both are available in evaluation versions that automatically install support packages for Stellaris boards and devices. For a VisualStudio-like environment, consider CrossStudio from Rowley.

Programming and Debugging

Tip 3: Unleash the power of JTAG for development and production.

PIC microcontroller debugging and programming uses a proprietary two-wire communication bus. In practice, access to nPOR and PGM pins can bring the total number of signal lines to four. The ICSP/ICD port essentially operates as either a debugger serial link, or as an in-circuit programming link. Debugging requires some PIC resources, including flash and RAM memory for the debug client. A full emulator (such as ICE4000 for $2500+) is the only debugging solution in applications where the entire memory space is needed by the application.

By contrast, Stellaris devices implement a standard JTAG interface as well as Serial Wire Debug (SWD). JTAG has dedicated signals for data-in (TDI), data-out (TDO), mode control (TMS), and clocking (TCK), so transfer rates are significantly higher than PIC ICSP. SWD is closer to ICSP in design and is presently not widely supported by ARM debuggers. JTAG can be used for debugging, programming, and for boundary-scan test.

Using JTAG to debug Stellaris microcontrollers does not tie up any CPU or memory resources as it does with PIC ICSP. This eliminates side-effect issues which surface occasionally with PIC debugging.

Historically, PIC microcontrollers had only one hardware break-point. New PIC microcontrollers have three, still less than the six available in Stellaris microcontrollers.

The Stellaris microcontroller's JTAG interface does not connect to the flash memory block, instead, the debugger loads a small client program into SRAM to handle the actual flash programming process. The result is a fast and reliable programming mechanism. This process is integrated into the debugger/IDE software (usually as a script) by the tools vendor, just click a toolbar icon to load to flash.

Developing Source Code

Tip 4: Use DriverLib instead of in-line code to access peripherals.

Code reuse with PIC microcontrollers typically involves cutting and pasting code from old projects or application notes, then editing the code to suit the current target device. This approach is largely out of necessity, memory constraints (size and banking issues) make it difficult to write reusable code. This means that when CPU cycles are in short supply, this style of coding allows code to be trimmed for maximum performance.

The same methodology can be applied to Cortex-M3, but you will discover that it is not necessary. Texas Instruments provides a comprehensive set of peripheral drivers in a single library called DriverLib. If you are used to low-level coding, working with Driverlib may take some getting used to. You will soon find the small investment will be worthwhile, both in your first Stellaris project and when you reuse code for future projects—no matter what the target device.

You may wonder why every DriverLib function call requires a Base Address parameter. The Base Address allows a single DriverLib module to handle multiple peripheral instances. So, for example, one set of functions can support Timers 0, 1 and 2. The result is code that is smaller, better structured and more portable.

When debugging code you may want to step down into a DriverLib function call. Because DriverLib is a separately compiled module, the debugger can only display assembly code and symbol information. In order to display and step through DriverLib source code, you will typically need to build DriverLib for debug, and then manually load the source code into the debugger.

Tip 5: If parts of the StellarisWare source code look unfamiliar then read this section.

A lot of time and effort have been invested to ensure DriverLib is clearly commented, bug-free, and portable. There are a few things in the source that might be unfamiliar if you are coming from an 8-bit background. The following examples can be applied to C source for any target, but they are less common on 8-bit platforms.

When browsing through the source for DriverLib, you will see a lot of comments with strange looking syntax (for example, `//! \addtogroup uart_api`), these are doxygen comments. Doxygen formatting provides for automatic generation of source-code documentation so that the documentation is always synchronized with the source. The doxygen syntax is seen as normal comments by the compiler and completely ignored during compilation.

Another thing you will encounter are shorthand conditional expressions that look like this:

```
ulValue = ( (ucExit == 'Y') ? 1 : 0);
```

This is equivalent to:

```
if (ucExit == 'Y')
{
      ulValue = 1;
}
else
{
      ulValue = 0;
}
```

Finally, DriverLib uses quite a few compiler directives and macros. Some of these are for portability across compilers, while others are to assist with debugging. One of these is the ASSERT macro, which is used to check that the arguments passed to a function are within a valid range. The ASSERT macro is defined in the debug.h file which is included in the DriverLib source tree.

```
ASSERT(!(ulAddress & 3));
```

This checks that the parameter ulAddress falls on a 32-bit boundary, but only generates run-time code if the application is built as debug and the error handler function `__error__` is implemented. The error handler typically sends an error message to a UART.

Tip 6: Do not be afraid to change compilers.

As with MPLAB, IDEs that target ARM Cortex-M3 may support several different compilers. Changing PIC compilers is not a simple process, due to deviations from ANSI C, register naming, and other nuances. The good news is that switching from one ARM compiler to another is trivial, typically requiring, at the most, a few conditional defines.

Coding Style

Tip 7: Write structured code without fear.

This application note is not about good and bad coding styles, but it is worth noting that the architecture of Stellaris Cortex-M3 frees you to write elegant code without the concerns of memory banking, tiny stack limits, or exactly what the processor will do with your code.

The PIC microcontroller's stack is not located in program or data memory, so its small size can be very limiting when coding in C. Also, because only the program counter is saved on the hardware stack, the compiler has to make special provision for storing local variables. These issues often force PIC programmers to limit, or at least plan for, using layers of function calls to structure code. ARM

Cortex-M3 has a conventional RAM-based stack that combines with the Link Register (LR) for efficient entry and exit from functions.

Some programming practices that give PIC microcontrollers problems are trivial on Stellaris. Go ahead and use 32-bit integers, complex structures, large arrays, and lots of function calls without fear! The result will be code that has fewer bugs and is easier to maintain.

Peripheral Registers

Tip 8: Access peripheral registers using DriverLib or simple in-line code.

Most designers coming from an 8-bit microcontroller background will choose a bottom-up design approach to developing code. This begins with writing code to set up peripherals using the peripheral configuration registers.

In Stellaris microcontrollers, as with PIC, the peripheral registers are memory mapped. In PIC microcontrollers, the peripheral registers are packed tightly together due to the limited data memory bank size (just 256 bytes). Registers for a given peripheral are not contiguous.

ARM's huge 32-bit address space lets Stellaris microcontrollers organize the peripheral registers more logically. Each peripheral starts at the same address boundary which is called the base address. This organizational feature helps greatly when multiple identical peripherals are present. For example, by changing only the base address, registers in either UART0 or UART1 can be selected. The sample code on page 10 shows how to directly access peripheral registers.

Tip 9: A simple bit mask must be used when reading or writing GPIO ports.

As we have already recommended, DriverLib is the easiest way to configure and access peripherals. However, if you want to access GPIOs without DriverLib, the source code in "Sample Code" on page 10 will be helpful. The minimalgpiodemo.c file configures and controls a general-purpose output pin without requiring external libraries or even header files.

The most important thing to note in this demo is that although the data register for Port C is located at 0x4000.6000, reading or writing this location will have no effect. Stellaris Cortex-M3 microcontrollers use address bits A[9:2] as a bit mask when reading or writing GPIO data registers. This clever feature avoids the overhead of performing a read/modify/write operation, but it can cause confusion if you are not aware of how it works. Since bits A[9:2] in the address 0x4000.6000 are all zeroes, all bits will be masked in read and write operations.

The sample code listing on page 10 shows how to take a bit-mask, shift it left to the A[9:2] position, add it to the base address, and use it to write a GPIO pin. The compiler's pre-processor calculates the actual address automatically, so the resulting code is very small.

The example code writes the value 0xff to set the pin high. Since the mask is 0x20, only bit 5 of the value is significant. Writing a value of 0x20 would have the same effect.

Interrupts

Tip 10: Stellaris microcontrollers use ordinary functions as interrupt handlers—no compiler directive required.

Moving from a device with only one or two interrupt vectors to Stellaris microcontrollers, which have seven system exceptions and up to 27 interrupts, can be daunting. In practice, however, the interrupts in Stellaris microcontrollers are easy to configure. Debugging interrupt issues also becomes much easier as each module generates a unique vector. During PIC development, it could be unclear exactly which module was responsible for the interrupt until numerous flags had been checked.

There are many source-code examples that demonstrate interrupt operation. These are the best reference for writing interrupt configuration and handler code.

The vector table can be located anywhere in flash or SRAM memory but is typically left at its default 0x0000.0000 location. There are several ways to configure the vector table. The most common method is to list the interrupt service routine (ISR) vectors in the startup.s assembly file. Unassigned vectors should be directed to a default interrupt handler.

The vectors in PIC code are actually CALL instructions, whereas ARM Cortex-M3 vectors are the actual 32-bit memory addresses of the handler functions.

Another notable advantage of Cortex-M3 is that interrupt service routines are normal C functions. There is no special entry or exit code so compiler directives, which are necessary with PIC C compilers, are not required.

Clocks

Tip 11: Your system can probably use a much lower clock rate.

A PIC microcontroller, running at full-speed of 40 MHz, requires four clocks or 100 ns for a typical instruction (8 or more for others). By comparison, a Stellaris microcontroller clocked at 40 MHz will execute most instructions in just 25 ns. When you combine this factor with a powerful ARM Cortex-M3 instruction set, it is time to recalibrate thinking when estimating clocking requirements.

Table 1 shows the correlation between clock modes on a typical PIC18F device and equivalent modes in Stellaris microcontrollers.

Table 1. Clock Mode Comparison

PIC18Fxxx Clock Mode	Equivalent Stellaris Clock Configuration
External Clock (EC/ECIO)	External Single-Ended Source
Internal Clock (INTIO)	Internal Oscillator
Crystal (LP, XT, HS)	External Crystal
High Speed with PLL (HSPLL)	Internal or External Clock with PLL Enabled

PIC microcontrollers have a special block of configuration bits which select device configuration at power-on. These bits control clocking, memory protection, the watchdog timer, and other functions.

Because these bits are outside of normal program memory space, they are either configured in MPLAB or by using table access instructions. Stellaris microcontrollers are configured entirely by register writes performed by the application code. This is a more straightforward approach, but you must remember to include the necessary code. A common oversight is forgetting to enable clocking to all in-use peripherals.

Universal Asynchronous Receivers/Transmitters (UARTs)

Tip 12: Do not forget to use the UART receive time-out interrupt.

The Stellaris microcontroller UARTs are much more sophisticated than PIC implementations. The fundamental difference is the presence of 16 byte transmit and receive FIFO buffers. In PIC systems, there is an interrupt event for each byte transferred. With a FIFO, the number of interrupts can be cut drastically as interrupts only occur when programmable FIFO limits are reached. So what happens if only one byte is received—not enough data to trigger a receive interrupt? The solution is the receive time-out interrupt, a programmable timer which generates an interrupt if no additional data has been received. Configure and use the receive time-out interrupt in cooperation with the receive FIFO full interrupt to ensure all data is received.

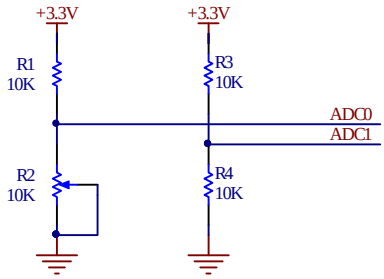
Analog-to-Digital Converter (ADC)

Tip 13: The Stellaris ADC module has a built-in voltage reference.

Both Stellaris and many PIC microcontrollers feature a 10-bit ADC. Unlike PIC microcontrollers, however, the Stellaris microcontroller ADC has an internal precision voltage reference that gives the ADC a fixed full scale of 3.0 V. Therefore, the accuracy figures in Stellaris microcontroller data sheets include gain errors due to the internal reference. In a PIC design, it would be necessary to connect V_{ref+} to a precision voltage reference to get equivalent DC performance.

One limitation of an internal voltage reference is that ratiometric operation is not possible, since the reference for the ADC is independent of the supply rail voltage. In applications where an external voltage reference is not practical, a second ADC channel can be used to sample the supply rail before compensating for variations in software as shown in Figure 1.

Figure 1. Using a Second ADC Channel for Software-based Ratiometric Operation



In Figure 1, each ADC channel should be configured for single-ended operation, rather than as a differential pair.

The Stellaris microcontroller ADC block also includes sophisticated sequencer and hardware averaging capabilities that can reduce software overhead typically needed in a PIC-based design. These are useful when making high-speed conversions on one channel with periodic low-speed conversions on another channel. For improved ADC precision in noisy environments, the ADC also supports internal over-sampling up to 64X. Oversampling using a PIC microcontroller requires a large number of CPU cycles.

Circuit Design

Tip 14: Stellaris microcontroller pins have at most two possible functions—ADC pins are dedicated.

Working with PIC microcontrollers usually involves some amount of I/O assignment wrangling as pins can have 2, 3, or 4 different functions. In contrast, Stellaris microcontrollers assign at most two functions to each pin: GPIO and a special function. Another difference is that Stellaris ADC pins are analog only; they can not be configured as GPIO pins.

Tip 15: Be generous with decoupling capacitors—they are not expensive!

The schematics for Stellaris evaluation boards show five decoupling capacitors for a 28-pin SOIC device—you may be wondering if they are all necessary. Stellaris microcontrollers have an on-chip voltage regulator for supplying power to logic. The regulator requires decoupling, just as an external three-terminal regulator would. Apart from that, the decoupling requirements for systems with Stellaris microcontrollers are no different than PIC microcontrollers—assuming equivalent clock speeds and supply voltages. Decoupling capacitors are low cost and good insurance for a range of system problems.

Tip 16: All input pins are Schmitt trigger types.

Microcontrollers are in close contact with the analog world, so noisy and slow changing signals are inevitable. Stellaris microcontrollers address this by providing Schmitt triggers with 0.5 V hysteresis on all digital input pins. This capability greatly simplifies input signal conditioning circuitry.

Sample Code

```
//*****************************************************************************
//
// minimalgpiodemo.c - Minimal demo for toggling a GPIO pin.
//
// Copyright (c) 2007 Luminary Micro, Inc. All rights reserved.
//
// Software License Agreement
//
// Luminary Micro, Inc. (LMI) is supplying this software for use solely and // exclusively
on LMI's microcontroller products.
//
// The software is owned by LMI and/or its suppliers, and is protected under // applicable
copyright laws. All rights are reserved. Any use in violation // of the foregoing
restrictions may subject the user to criminal sanctions // under applicable laws, as well
as to civil liability for the breach of the // terms and conditions of this license.
//
```

```
// THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED // OR
STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF // MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
// LMI SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR //
CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
//
//*****************************************************************************

//
// Include the LM3S811 header file containing definitions of all peripheral // registers
contained in the IC.
//
#include "lm3s811.h"

//*****************************************************************************
//
// Toggles the User LED on EKK-LM3S811 Evaluation Kit without using DriverLib.
//
//*****************************************************************************
int
main(void)
{
    //
    // Enable Clock gating to GPIO Port C
    //
    SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOC;

    //
    // Configure Port C pin 5 as a General purpose output
    //
    GPIO_PORTC_DIR_R |= 0x0020;

    while (1)
    {
        //
        // Set Port C pin 5 high using ADDR[9:2] as a mask
        //
        GPIO_PORTC_DATA_BITS_R[0x20] = 0xFF;

        //
        // Set Port C pin 5 low using ADDR[9:2] as a mask
        //
        GPIO_PORTC_DATA_BITS_R[0x20] = 0x00;
    }
}
```

Conclusion

A wide range of resources, including comprehensive source code and application notes, make transitioning from 8-bit PIC microcontrollers to 32-bit Stellaris microcontrollers straightforward. And by moving to ARM you may never have to change architecture again.

References

The following documents are available for download at www.luminarymicro.com:

- Stellaris® microcontroller data sheet, Publication Number DS-LM3S*nnn* (where *nnn* is the part number for that specific Stellaris family device)
- *Stellaris® Family Driver Library User's Guide*, Document order number SW-DRL-UG
- Stellaris® Peripheral Driver Library, Order number SW-DRL

In addition, the ARM web site may aslo be useful: www.arm.com

Important Notice

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI 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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

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