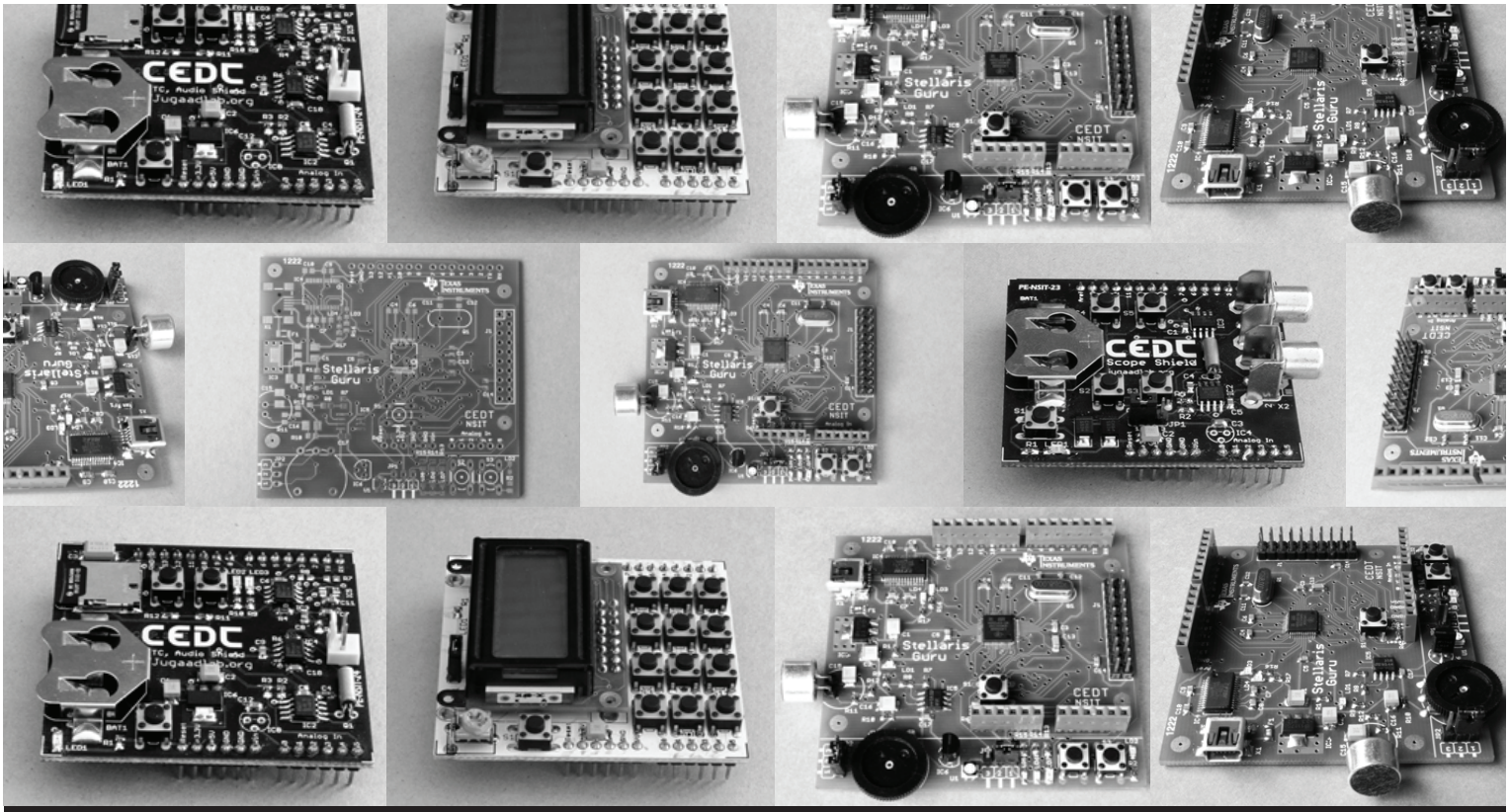
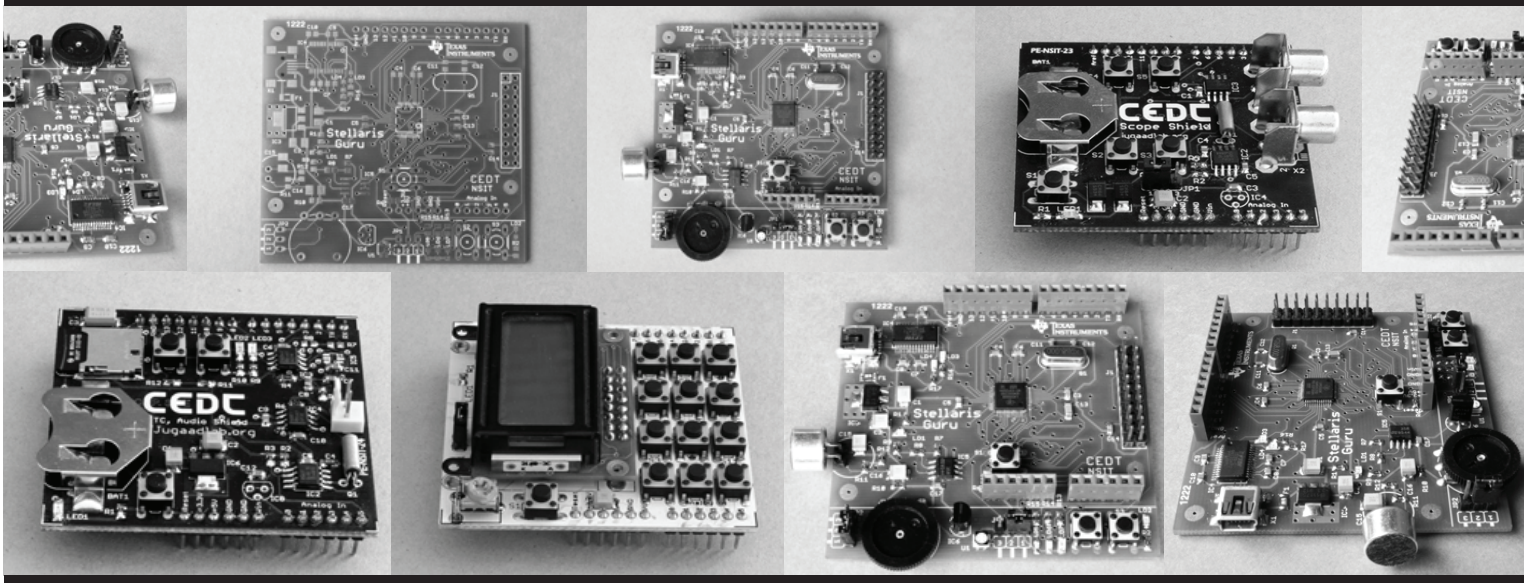


ARM Cortex M3
Stellaris Guru Development Kit
Manual



Centre for Electronics Design and Technology
Netaji Subhas Institute of Technology

ARM® Cortex™-M3 Stellaris Guru Development Kit Manual

Written and edited at the Centre for Electronics Design and Technology, Netaji Subhas Institute of Technology, New Delhi, India.

Cover Page Art by Kanika Kaul.

Table of Contents

1. The Stellaris Guru

1.1 Kit Contents

The kit box contains the following items:

1. Stellaris Guru Kit
2. USB Cable

1.2 Stellaris Guru Kit

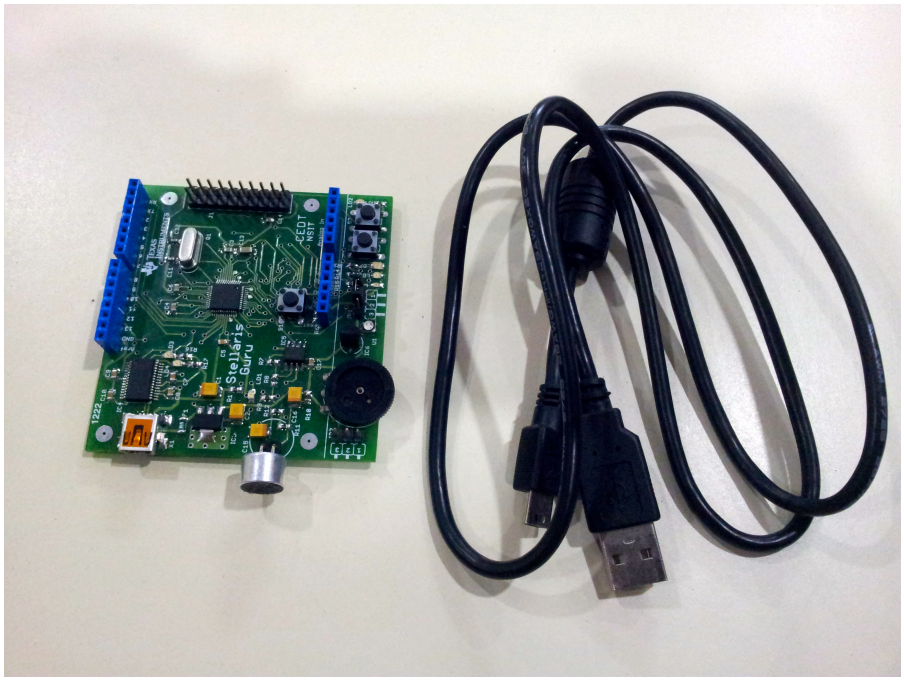


Figure 1.1 Stellaris Guru Kit Contents

1.3 What to Expect?

When you connect the Stellaris Guru kit using the USB cable to your computer a few indicator LEDs turn on/flash indicating the board is in a ready to use state. Refer to the annotated Stellaris Guru Figure 7.1 on page 16 for locating the indicator LEDs.

1. LEDs LD3 and LD4 would light flash momentarily for a second confirming ability to establish serial connection with the computer.
2. The power indicator LED LD1 would be continuously on confirming the board is able to draw power from the USB.

2. Background of ARM and ARM Architecture

2.1 A Brief History

ARM was founded in 1990 as Advanced RISC Machines Ltd., a joint venture of Apple Computer, Acorn Computer Group, and VLSI Technology. In 1991, ARM introduced the ARM6 processor family, and VLSI became the initial licensee. Subsequently, additional companies, including Texas Instruments, NEC, Sharp and ST Microelectronics, licensed the ARM processor designs.

Nowadays ARM partners ship in excess of 2 billion ARM processors each year. Unlike many semiconductor companies, ARM does not manufacture processors or sell the chips directly. Instead it licenses the processor designs to business partners. This business model is commonly called intellectual property (IP) licensing.

2.2 Architecture Versions

Over the years, ARM has continued to develop new processors and system blocks. These include the popular ARM7TDMI processors, more recently the ARM11 processor family and latest being the ARM Cortex Processor family. Table 2.1 summarizes the various architecture versions and processor families.

Architecture	**Processor Family**
ARMv1	ARM1
ARMv2	ARM2, ARM3
ARMv3	ARM6, ARM7
ARMv4	StrongARM, ARM7TDMI, ARM9TDMI
ARMv5	ARM7EJ, ARM9E, ARM10E, XScale
ARMv6	ARM11, ARM Cortex –M
ARMv7	ARM Cortex-A, ARM Cortex-R, ARM Cortex-M

Table 2.1 – ARM architecture versions and processor families

In the manual we focus on the ARMv7 architecture and the ARM Cortex-M processor family. But before moving onto the Cortex-M, it is worthwhile to mention about the Cortex-A and Cortex-M and cite the basic differences.

2.3 ARM Cortex Processor Profiles

In the ARMv7 architecture the design is divided into three profiles –

1. **A Profile (ARMv7-A)** - is designed for high performance open application platforms. They can handle complex applications such as high end embedded operating systems. Example products include high-end smartphones, tablet PCs and PDAs.
2. **R Profile (ARMv7-R)** - is designed for high end embedded systems in which real time performance is needed. Example applications include high-end car braking systems.

3. **M-Profile (ARMv7-M)** – is designed for deeply embedded microcontroller type systems. Processors belonging to this profile are the subject for this manual and are studied in greater detail.

2.4 Evolution of the ARM Instruction Set

Enhancement and extension of instruction sets used by the ARM processors has been one of the key driving forces of the architecture's evolution.

Historically, two different instruction sets were supported on the ARM processor: the **ARM instructions**, which are 32 bits and **Thumb instructions**, which are 16 bits. During program execution, the processor can be dynamically switched between the ARM state and the Thumb state. The Thumb instruction set provides only a subset of the ARM instructions, but it can provide higher code density. It is useful for products with tight memory requirements.

The **Thumb-2** technology extended the Thumb Instruction Set Architecture (ISA) into a highly efficient and powerful instruction set that delivers significant benefits in terms of ease of use, code size and performance. The extended instruction set in Thumb-2 is a superset of the previous 16-bit Thumb instruction set, with additional 16-bit instructions alongside 32-bit instructions. It allows more complex operations to be carried out in the Thumb state, thus allowing higher efficiency by reducing the number of states switching between ARM state and Thumb state.

The Cortex-M3 processor supports only the Thumb-2 (and traditional Thumb) instruction set. Instead of using ARM instructions, as in traditional ARM processors, it uses Thumb-2 instruction set for all operations. As a result, the Cortex-M3 processor is not backward compatible with traditional ARM processors, which use the ARM as well as Thumb instruction set.

The Thumb-2 instruction set is a very important feature of the ARMv7 architecture. For the first time, hardware divide instruction is available on an ARM processor, and a number of multiply instructions are also available.

3. Overview of the Cortex-M3

3.1 Salient Features of the Cortex-M3

- 32-bit microprocessor.
- 32-bit data path, 32-bit register bank and 32-bit memory interfaces.
- Harvard Architecture – separate instruction bus and data bus.
- 3-stage pipeline with branch speculation.
- Thumb-2 instruction set.
- No switching between ARM state and thumb state.
- Instruction fetches are 32 bits. Up to two instructions can be fetched in one cycle. As a result, there's more available bandwidth for data transfer.
- ALU with hardware divide and single cycle multiply.
- Configurable Nested Vector Interrupt Controller (NVIC).
- Maximum of 240 external interrupts can be configured.
- Low gate count, suitable for low power designs.
- Memory Protection Unit (MPU).
- Operation Mode Selection – user and privilege modes.
- Advanced debug components.

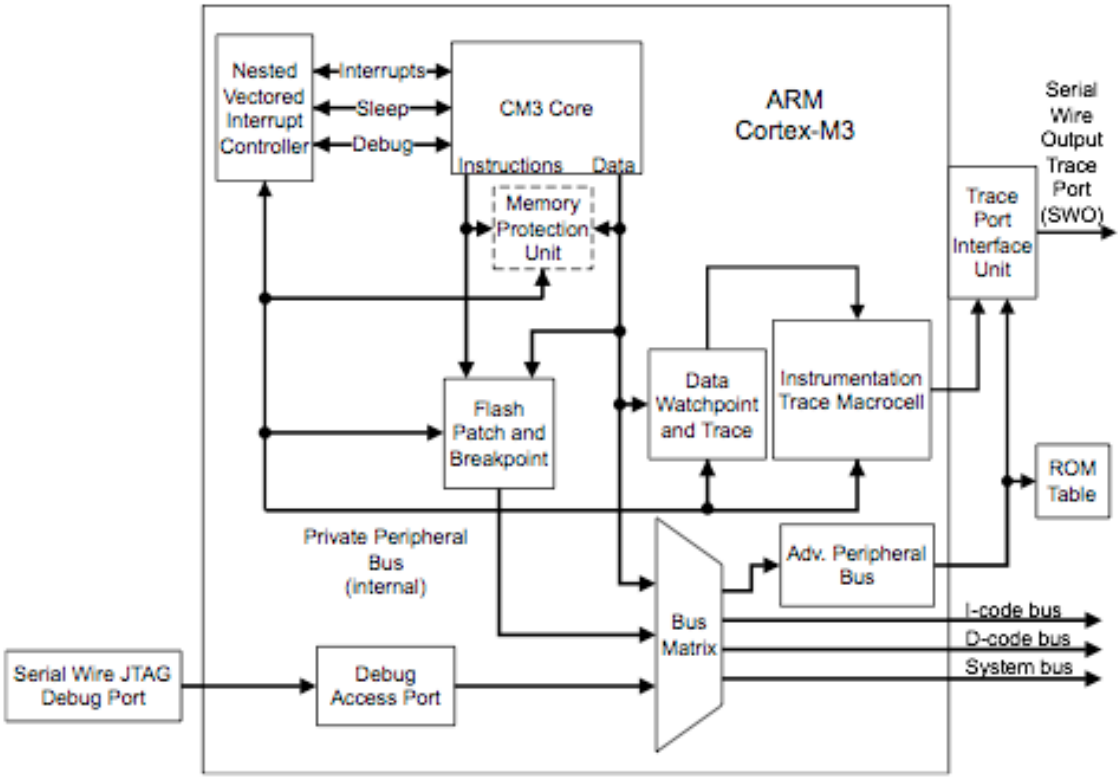


Figure 3.1 ARM Cortex-M3 Architecture View

3.2 Registers

- The Cortex-M3 has registers R0 through R15. R13 (the stack pointer) is banked, with only one copy of the R13 visible at a time.
    - R0 – R12: General Purpose Registers. These are 32 bit registers for data operations. Some 16-bit Thumb instructions can only access a subset of these registers (low registers R0-R7).
    - R13: Stack Pointers. Contains two stack pointers. They are banked so that only one is visible at a time.
        - Main Stack Pointer (MSP) – The main stack pointer used by the Operating system and exception handlers.
        - Process Stack Pointer (PSP) – used by the application code.
    - R14: Link Register. When a subroutine is called, the return address is stored in the link register.
    - R15: The Program Counter. The program counter is the current program address.
- The Cortex-M3 also has a number of special registers. They are -
    - Program Status registers (PSR)
    - Interrupt Mask registers (PRIMASK, FAULTMASK and BASEPRI).
    - Control register (CONTROL)
- The Cortex-M3 has 18 registers in total compared to 37 registers for traditional ARM. Figure 3.2 shows the registers.

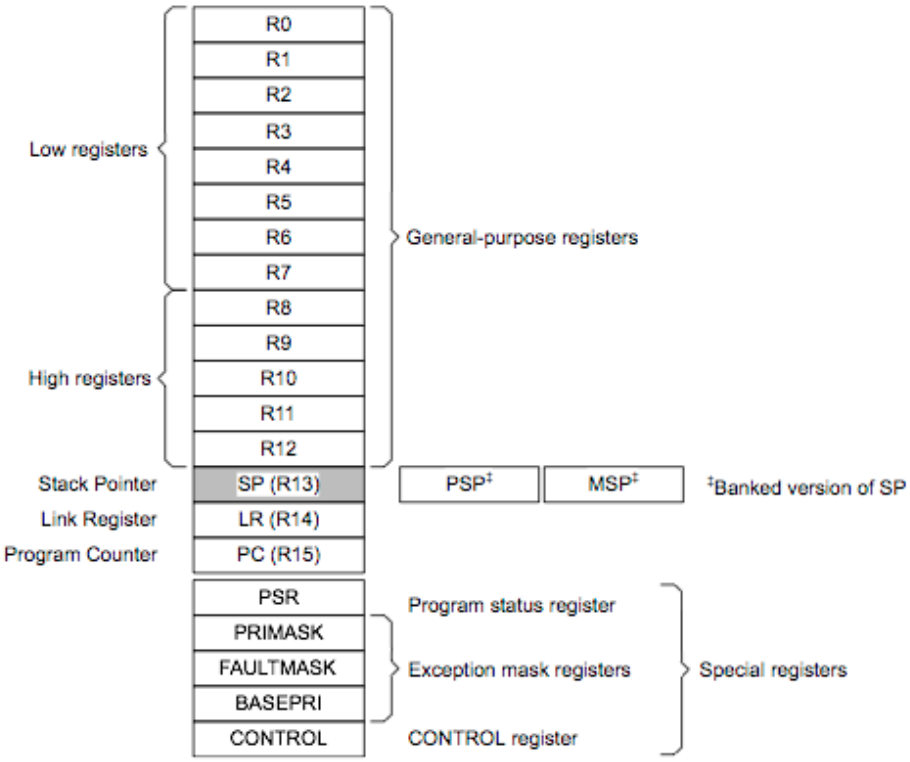


Figure 3.2 Registers in Cortex-M3

3.3 Memory Map

The Cortex-M3 has predefined memory maps, which allows built in peripherals, such as the interrupt controller and debug components, to be accessed by simple memory access instructions. The predefined memory map also allows the Cortex-M3 processor to be highly optimized for speed and ease of integration in system-on-a-chip (SoC) designs.

The Cortex-M3 design has an internal bus infrastructure optimized for this memory usage. In addition, the design allows these regions to be used differently. For example, data memory can still be put into the CODE region, and program code can be executed from an external Random Access Memory (RAM) region. The Cortex-M3 memory map is outlined in Figure 3.3.

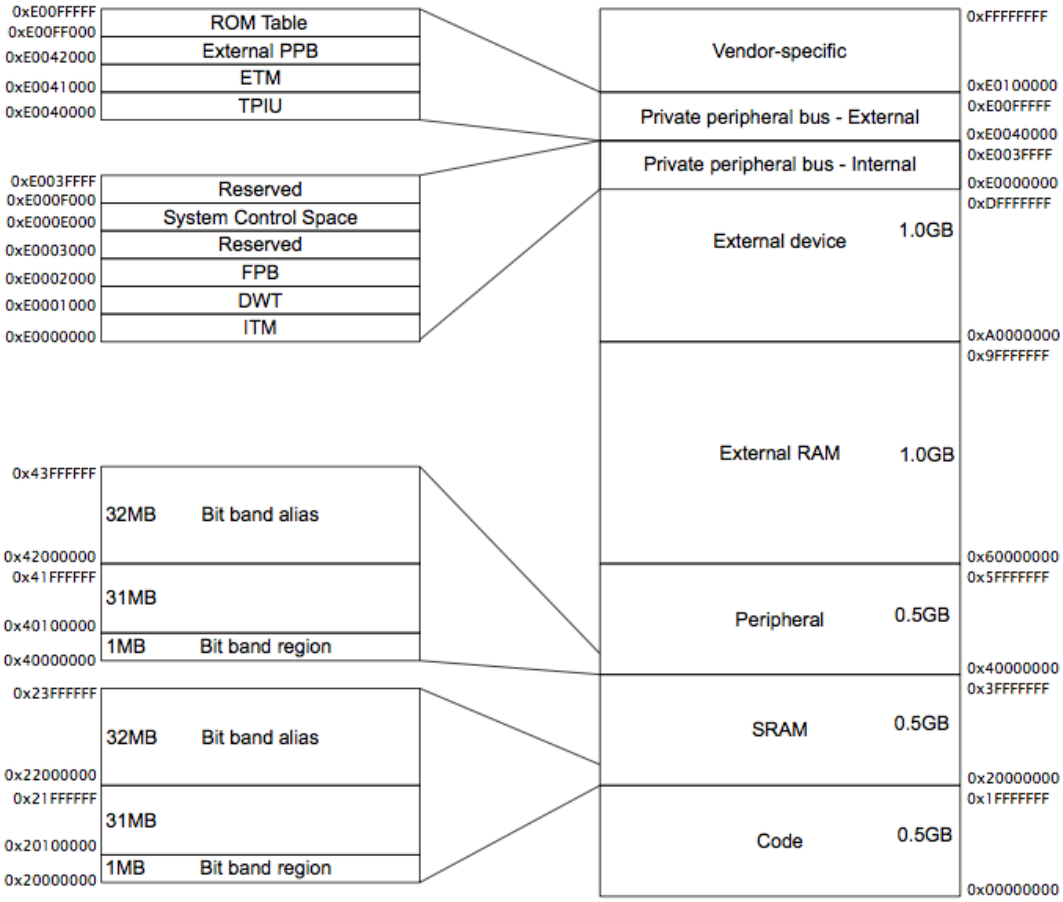


Figure 3.3 Cortex-M3 Memory Map

3.4 Operation Modes

The Cortex-M3 processor has two modes of operation and two privilege levels. The operation modes (thread mode and handler mode) determine whether the processor is running a normal program or running an exception handler like an interrupt handler or system exception handler. The privilege levels provide a mechanism for safeguarding memory accesses to critical regions as well as providing a basic security model.

When the processor is running a main program (thread mode), it can be either in a privileged state or a user state, but exception handlers can only be in a privileged state. When the processor exits reset, it is in thread mode with privileged access right. In this state, a program has access to all memory ranges and can use all supported instructions.

Software in the privileged access level can switch the program into the user access level using the control register. When an exception takes place, the processor will always switch back to the privileged state and return to the previous state when exiting the exception handler. A user program cannot change back to the privileged state by writing to the control register. It has to go through an exception handler that programs the control register to switch the processor back into the privileged access level when returning to thread mode. Figure 3.4 outlines the available operation modes and privilege levels.

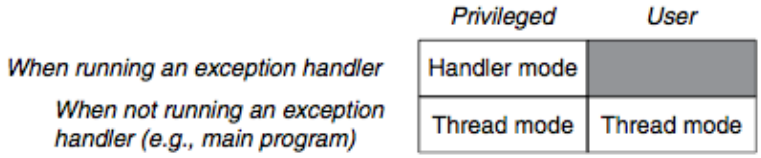


Figure 3.4 Operation modes and Privilege Level in Cortex-M3

3.5 Nested Vector Interrupt Controller (NVIC)

The Cortex-M3 processor includes an interrupt controller called the Nested Vectored Interrupt Controller (NVIC). It is closely coupled to the processor core and provides a number of features as follows:

- Nested interrupt support
- Vectored interrupt support
- Dynamic priority changes support
- Reduction of interrupt latency
- Interrupt masking

Nested Interrupt support - All the external interrupts and most of the system exceptions can be programmed to different priority levels. When an interrupt occurs, the NVIC compares the priority of this interrupt to the current running priority level. If the priority of the new interrupt handler is higher than the current level, the interrupt handler of the new interrupt will override the current running task.

Vectored Interrupt Support - When an interrupt is accepted, the starting address of the interrupt service routine (ISR) is located from a vector table in memory.

Dynamic Priority Changes Support - Priority levels of interrupts can be changed by software during run time. Interrupts that are being serviced are blocked from further activation until the ISR is completed, so their priority can be changed without risk of accidental reentry.

Reduction of Interrupt Latency - The Cortex-M3 processor also includes a number of advanced features to lower the interrupt latency. These include automatic saving and restoring some register contents and reducing delay in switching from one ISR to another.

Interrupt Masking - Interrupts and system exceptions can be masked based on their priority level or masked completely using the interrupt masking registers BASEPRI, PRIMASK, and FAULTMASK.

4. Texas Instruments® Stellaris® 32-bit ARM Cortex-M Family

In the Stellaris Guru Development board we use an ARM Cortex-M3 based microcontroller from the Stellaris family manufactured by Texas Instruments. Before getting into the details of the controller used in the Stellaris Guru kit, we would like to highlight some of the prime features available on the Stellaris family of microcontroller by Texas Instruments.

- **ARM® Cortex™ -M3 v7-M Processor Core**
  - Up to 80 MHz
  - Up to 100MIPS (at 80 MHz).
- **On-Chip Memory**
  - 256KB Flash, 96KB SRAM.
  - ROM Loaded with Stellaris Driver-Lib, Boot-loader, AES Tables and CRC.
- **External Peripheral Interface (EPI)**
  - 32-bit dedicated parallel bus for external peripherals.
  - Supports SDRAM, SRAM/Flash, M2M.
- **Advanced Serial Integration**
  - 10/100 Ethernet MAC and PHY.
  - 3 CAN 2.0 A/B Controllers.
  - USB Full Speed, OTG/Host/Device.
  - 3 UARTs with IrDA and ISO 7816 support.
  - 2 I^2Cs.
  - 2 Synchronous Serial Interfaces (SSI).
  - Integrated Interchip Sound (I^2S).
- **System Integration**
  - 32 Channel DMA Controller.
  - Internal precision 16MHz oscillator.
  - 2 watchdog times with separate clock domains.
  - ARM Cortex SysTick timer.
  - 4 32-bit timers with RTC capability.
  - Lower power battery backed hibernation module.
  - Flexible pin-muxing capability.
- **Advanced Motion Control**
  - 8 advanced PWM outputs for motion and energy applications.
  - 2 Quadrature Encoder Inputs (QEI).
- **Analog**
  - 2 x 8 Channel 10-bit ADC (for a total of 16 channels).
  - 3 analog comparators.
  - On-chip voltage regulator (1.2V internal operation).

The controller used on the Stellaris Guru board may not have all the features outlined above. We discuss the features and capabilities of the Stellaris LM3S608 ARM Cortex-M3 based microcontroller, which is the heart of the Guru development kit. But before we hop on to the LM3S608, we provide you a brief showcase of the entire TI Stellaris Family.

Family	Flash (kB)	SRAM (kB)	Max Speed (MHz)	I/O Pins	Package	Features
x00 Series (Real Time MCUs)	8kB to 64kB	2kB to 8kB	20Mhz to 50Mhz	Upto 36	48LQFP, 48QFP	10 bit ADC, USART, I²C, SSI, Motion Control
1000 Series (Real Time MCUs)	64kB to 512kB	19kB to 96kB	25Mhz to 80Mhz	Upto 67	64LQFP, 100LQFP, 108BGA	10/12 bit ADC, USART, I²C, SSI, Motion Control, Hibernation Module
2000 Series (CAN Connected MCUs)	64kB to 512kB	19kB to 96kB	25Mhz to 80Mhz	Upto 67	64LQFP, 100LQFP, 108BGA	10/12 bit ADC, USART, I²C, SSI, Motion Control, CAN, Hibernation Module
3000 Series (USB Connected MCUs)	16kB to 256kB	6kB to 64kB	50Mhz	Upto 61	64LQFP, 100LQFP	10 bit ADC, USART, I²C, SSI, Motion Control, USB O/H/D, Hibernation Module
5000 Series (USB + CAN MCUs)	16kB to 512kB	8kB to 96kB	50Mhz to 80Mhz	Upto 72	64LQFP, 100LQFP, 108BGA	10/12 bit ADC, USART, I²C, SSI, Motion Control, USB O/H/D, CAN, Hibernation Module
6000 Series (Ethernet Connected MCUs)	64kB to 512kB	16kB to 64kB	25Mhz to 80Mhz	Upto 46	100LQFP, 108BGA	10/12 bit ADC, USART, I²C, SSI, Ethernet, Motion Control, Hibernation Module
8000 Series (Ethernet + CAN MCUs)	96kB to 512kB	64kB	50Mhz to 80Mhz	Upto 46	100LQFP, 108BGA	10/12 bit ADC, USART, I²C, SSI, Ethernet, CAN, Motion Control, Hibernation Module
9000 Series (Ethernet + CAN + USB MCUs)	128kB to 512kB	64kB to 96kB	80Mhz	Upto 72	100LQFP, 108BGA	10/12 bit ADC, USART, I²C, SSI, Ethernet, CAN, USB O/H/D, Motion Control, Hibernation Module

The LM3S608 belongs to the 600 Series Family and has 32kB Flash, 8kB SRAM, max speed of 50Mhz, 10 bit ADC, dual USART, I²C, SSI, Motion Control and is available in a 48LQFP package.

5. Texas Instruments® Stellaris® LM3S608 Microcontroller - Features

A Texas Instruments® Stellaris® LM3S608 microcontroller forms the heart of the Stellaris Guru board. The underlying sections describe the various features available on the controller and its pin out.

- 32-bit RISC Performance
  - 32-bit ARM® Cortex™-M3 v7-M architecture.
  - SysTick timer, providing a simple 24-bit clear-on-write, decrementing, wrap-on-zero counter.
  - Thumb® compatible Thumb-2 only instruction set.
  - 50-MHz Operation
  - Integrated Nested Vector Interrupt Controller.
  - 23 interrupt with eight priority levels.
- Internal Memory
  - 32KB single cycle flash.
  - 8KB single cycle SRAM.
- GPIOs
  - 5-28 GPIOs, depending on configuration.
  - 5V tolerant input configuration.
  - Fast toggle capable of a change every two-clock cycle.
  - Programmable control for GPIO interrupts.
  - Programmable control for GPIO pad configuration.
- General Purpose Timers
  - Three general-purpose timers modules each of which provides two 16-bit timers/counters. Each can be conFigured independently:
    - As a single 32-bit timer.
    - As one 32-bit Real-time Clock.
    - For Pulse Width Modulation.
- ARM FiRM-compliant Watchdog Timer.
- ADC
  - Eight analog input channels.
  - Single ended and differential input configurations.
  - On-chip internal temperature sensor.
  - Sample rate of 500 thousand samples/second.
- UART
  - Two fully programmable 16C550-type UARTs.
  - Separate 16x8 transmit (TX) and receive (RX) FIFOs to reduce CPU interrupt service loading.
  - Fully programmable serial interface characteristics
    - 5,6,7 or 8 data bits.
    - Even/Odd/Stick or No-Parity generation.
    - 1 or 2 stop bit generation.
- Synchronous Serial Interface
  - Master of Slave Operation.
  - Programmable clock bit rate and operation.

    - Programmable data frame size from 4 to 16 bits.
- I^2C
    - Devices on the I2C bus can be designated as either master or a slave.
    - Four I2C modes
        - Master transmit
        - Mater receive
        - Salve transmit
        - Slave receive
    - Two transmission speeds, standard (100kbps) and Fast (400kbps).
    - Master and slave interrupt generation.
- Analog Comparators
    - One integrated analog comparator.
    - Configurable for output to drive an output pin, generate an interrupt or initiate an ADC sample sequence.
- Power
    - On-chip Low Drop (LDO) voltage regulator with programmable output user adjustable from 2.25V to 2.75V.
    - Low power options on controller: Sleep and Deep Sleep Modes.
    - Low power options for peripherals: software controls shutdown of individual peripherals.
    - 3.3V supply brownout detection and reporting via interrupt or reset.
- Flexible Reset Sources
    - Power-on Reset (POR)
    - Reset pin assertion.
    - Brownout (BOR) detector alerts to system power drops.
    - Software reset.
    - Watchdog timer reset.
- JTAG
    - IEEE 1149.1-1990 compatible Test Access Port (TAP) controller.
    - Four bits Instruction Register (IR) chain for storing JTAG instructions.
    - IEEE standard instructions: BYPASS, IDCODE, SAMPLE/PRELOAD, EXTEST and INTEST.
    - Integrated ARM Serial Wire Debug (SWD).

6. Texas Instruments® Stellaris® LM3S608 Microcontroller – Pin out

The soul of the Stellaris Guru is the LM3S608 microcontroller. It follows the ARM Cortex-M 32-bit RISC architecture. The LM3S608 used on the kit is in the 48-pin Thin Quad Flat Package (TQFP). The pin assignment is shown is Figure 6.1.

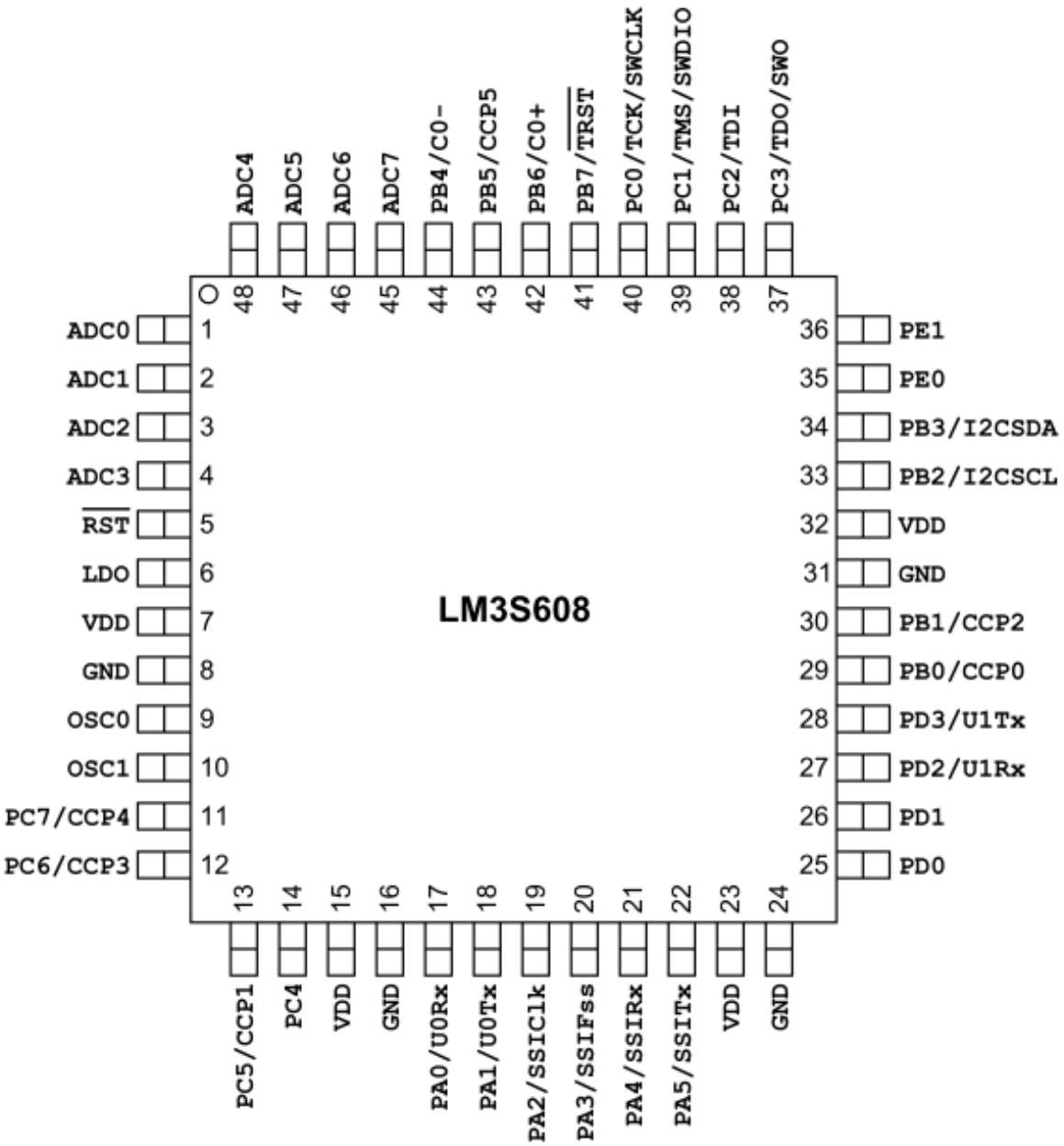


Figure 6.1 LM3S608 Pin Configuration (TQFP Package)

7. Introduction to the Stellaris Guru Kit

The Stellaris Guru kit has unique and substantial features, which can be tested and evaluated, right out of the box.

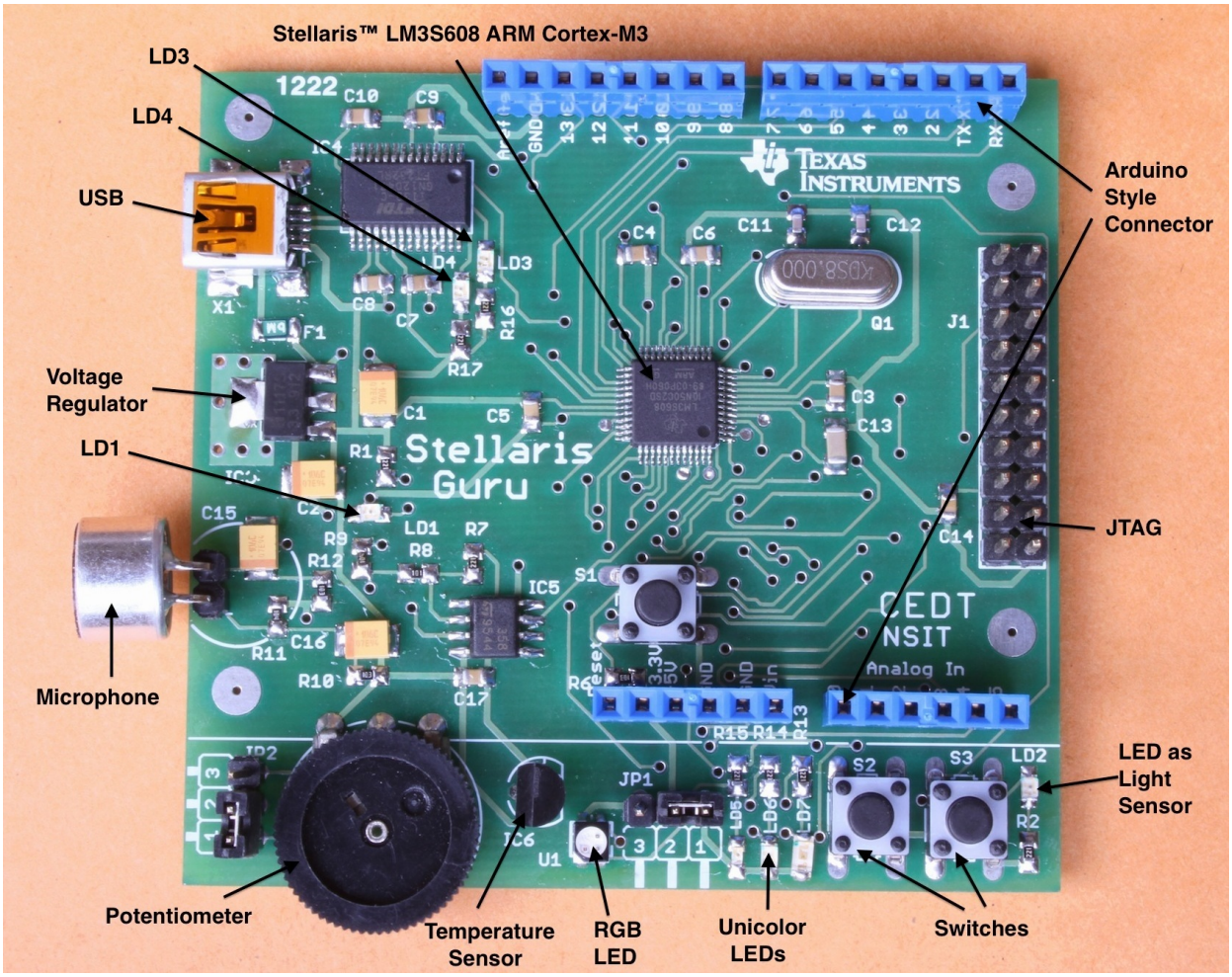


Figure 7.1 Stellaris Guru Kit

7.1 Features Overview

- Based on LM3S608 microcontroller family from the TI Stellaris 600 series family.
- User programmable push buttons and ultra-bright LEDs, both unicolor and RGB.
- Reset pushbutton and power indicator LED.
- A LM35 temperature sensor for taking temperature readings using the Analog to Digital Convertor.
- Thumbwheel potentiometer for reading analog voltage through one of the ADC channels of the microcontroller.

- Microphone amplifier with high gain and sensitivity connected to an independent ADC channel of the microcontroller.
- Ambient light sensor using a LED operated in reverse bias.
- Standard ARM 20-pin JTAG debug connector.
- Arduino compatible interface connector.
- UART0 accessible through a USB virtual COM port (VCP).
- Programmable through UART using preinstalled boot loader.
- USB interface for all communication and power.

7.2 Board Layout

Figure 7.2 shows the board layout for the kit from the component side.

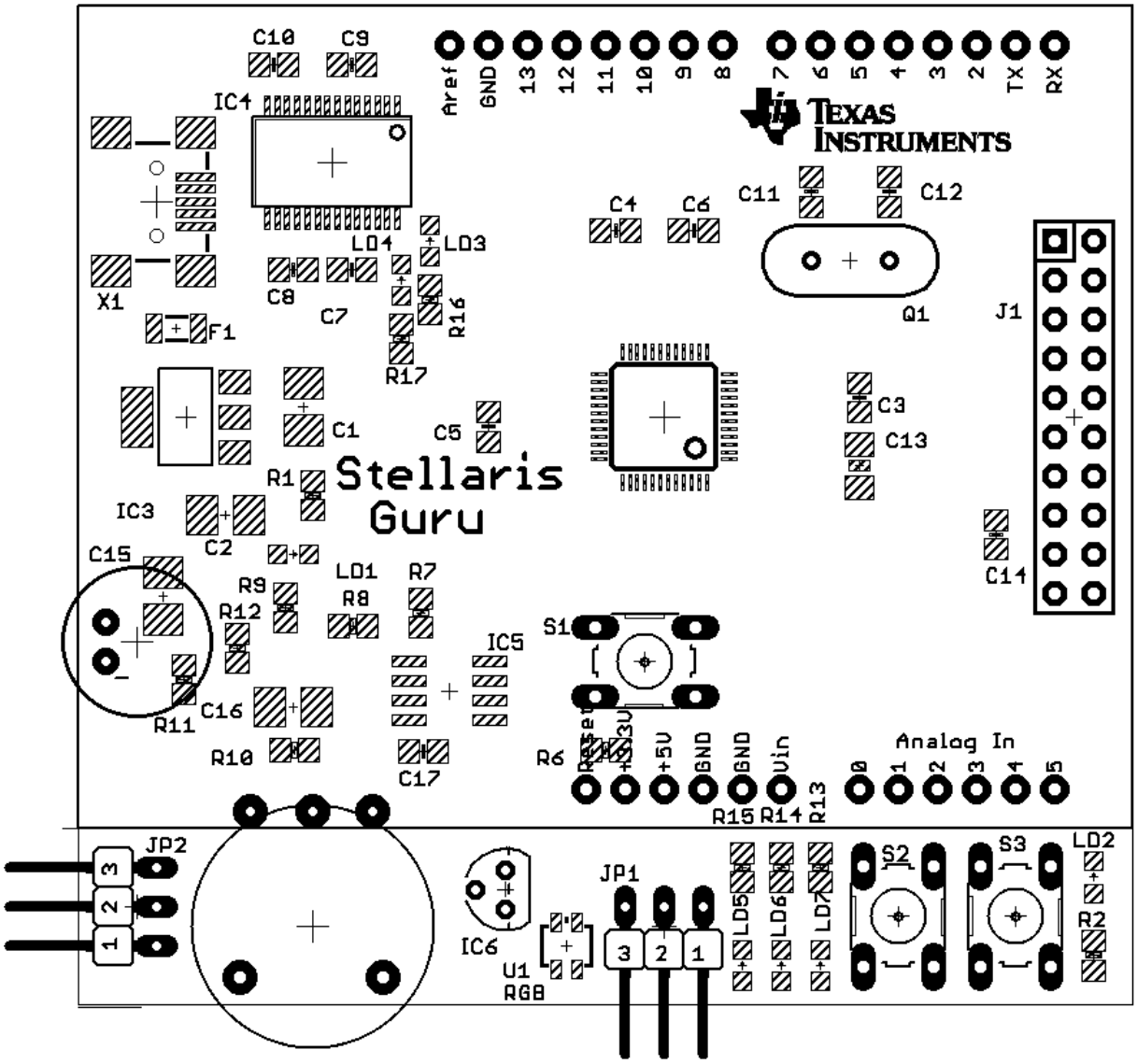
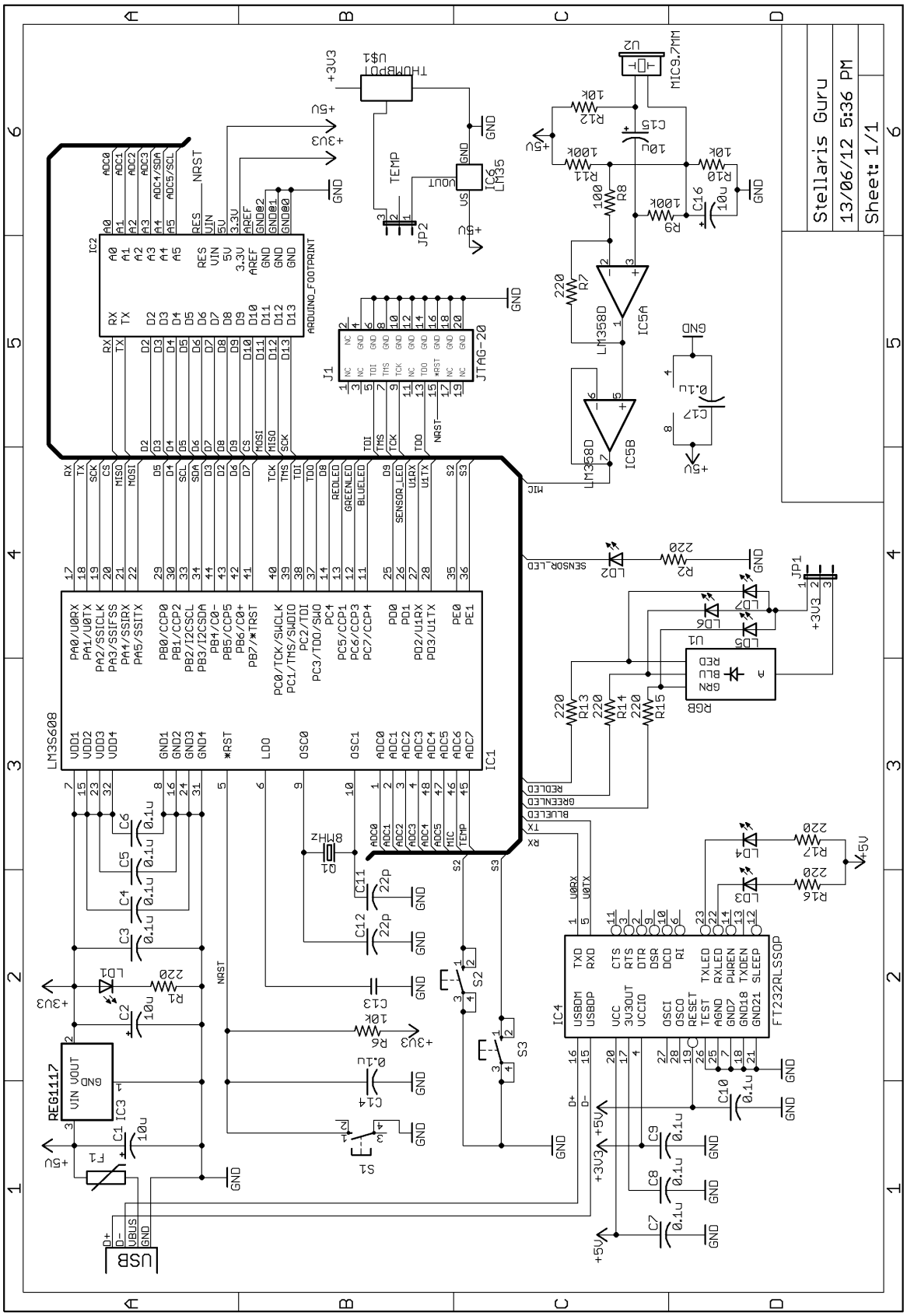


Figure 7.2 Board layout of the Stellaris Guru Kit (component side)

7.3 Schematic



8. Components of the Kit

This section divides the schematic from the last section into modules and explains the purpose and operation of each module and the hardware design.

8.1 Power Supply

The power supply section of the kit is shown below in Figure 8.1. Power is drawn directly from the USB bus of a computer, which is at +5V potential. This voltage is then fed to a Low Drop Out voltage regulator LM1117-3.3 (IC3) to generate +3.3V required by the microcontroller and other kit components. Tantalum capacitors C1 and C2 provide input and output filtering respectively. LED1 connected in series with the resistor R1 provides visual indication for power being supplied to the kit.

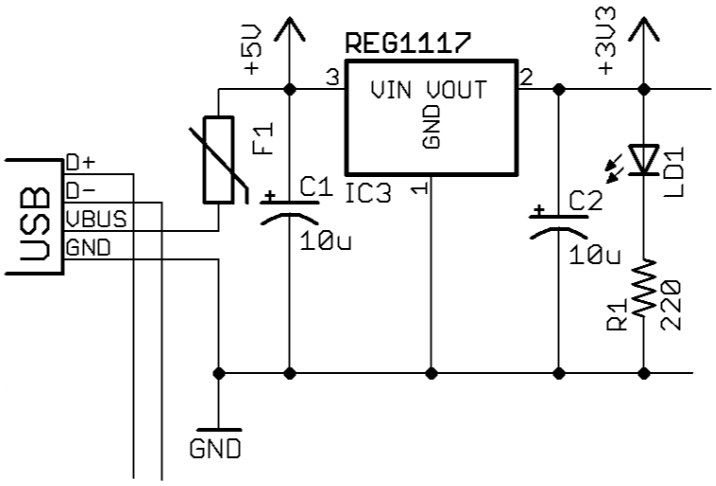


Figure 8.1 Power Supply

8.2 Microcontroller Connections

The connections related to the LM3S608 microcontroller are shown in Figure 8.2. The microcontroller (IC1) is powered by the +3.3V output of the LM1117-3.3V LDO regulator connected to the several VCC pins on the chip. Capacitors C3, C4, C5 and C6 act as bypass capacitors for power supply filtering. They are also used as tank capacitors in high speed switching. A crystal (Q1) of 8MHz is used for clocking of the micro controller. Capacitors C11 and C12 of 22pF are used to bias the crystal. A capacitor (C13) of value 1uF is used to bypass the internal LDO of the microcontroller to GND. An active low pushbutton (S1) is connected to the Reset pin of the microcontroller. The required JTAG pins on the microcontroller have been brought out on a 20-pin expansion header (J1) for In-Circuit Debugging. Pins which are not tied to functions on the board have been brought out on an Arduino-compatible interface connector (IC2).

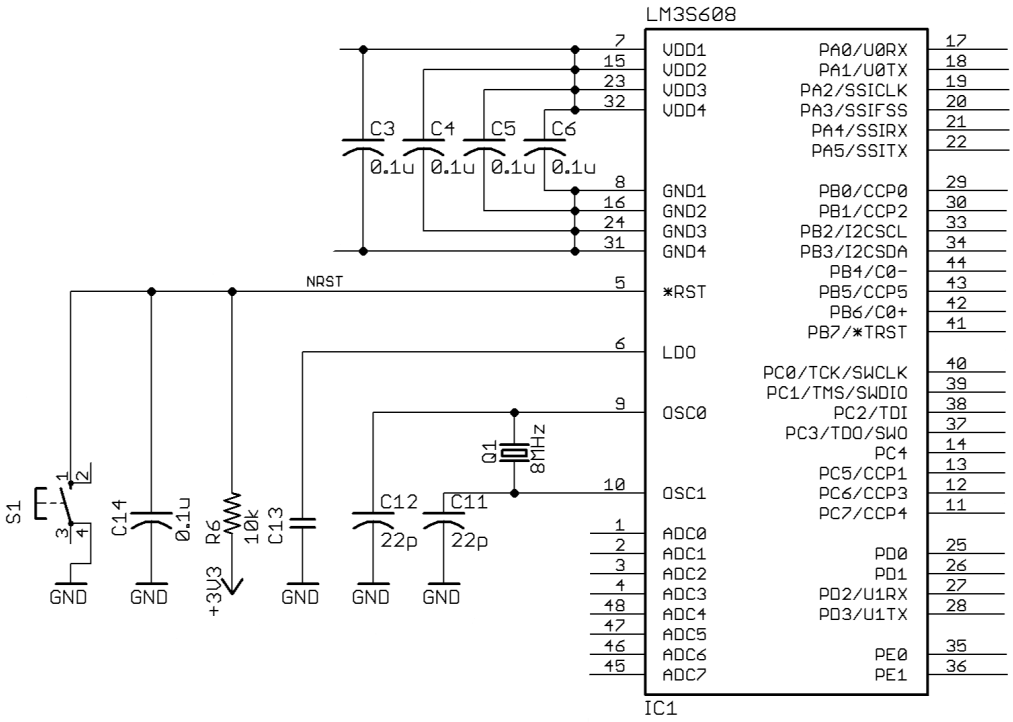


Figure 8.2 Microcontroller Connections

8.3 ARM 20-Pin JTAG Connector

The JTAG pins of the microcontroller have been brought out to a 20-pin ARM-compliant JTAG connector. The connections for the JTAG connector are shown in Figure 8.3.

8.4 Arduino Compatible Interface Connector

The Stellaris Guru has Arduino style connector pins, which allow the user to interface the kit to over 300 existing Arduino shields. Connections for the Arduino style connector are shown in Figure 8.4. Table 8.1 lists the interconnections made between the Arduino style connector and the LM3S608 microcontroller.

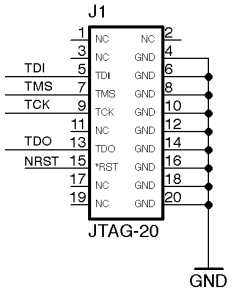


Figure 8.3

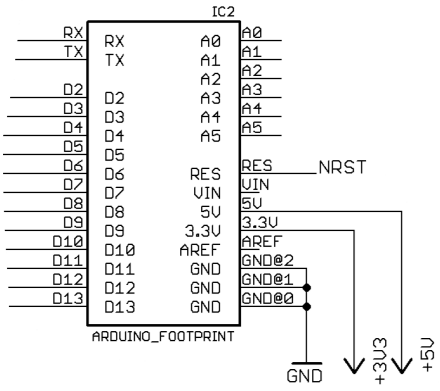


Figure 8.4

Arduino Pin	Microcontroller Pin	Arduino Pin	Microcontroller Pin
RX	PD2/U1RX	D11	PA5/SSITX
TX	PD3/U1TX	D12	PA4/SSIRX
D2	PB5/CCP5	D13	PA2/SSICLK
D3	PB4/C0-	A0	ADC0
D4	PB1/CCP2	A1	ADC1
D5	PB0/CCP0	A2	ADC2
D6	PB6/C0+	A3	ADC3
D7	PB7/*TRST	A4	ADC4 & PB3/I2CSDA
D8	PC4	A5	ADC5 & PB2/I2CSCL
D9	PD0	Reset	*RST
D10	PA3/SSIFSS	-	

Table 8.1 Interconnections – Arduino Compatible Interface Connectors and LM3S608

8.5 USB Virtual COM Port

The Stellaris Guru kit uses a FT232RL (IC4) USB-to-Serial bridge connected between UART0 of the microcontroller and the USB. This helps in field programming of the microcontroller over UART using the supplied boot loader. The port can also be used to send UART values to a PC application, viz. Hyper Terminal, Bray++, etc. The connections for the FT232RL are shown in Figure 8.5.

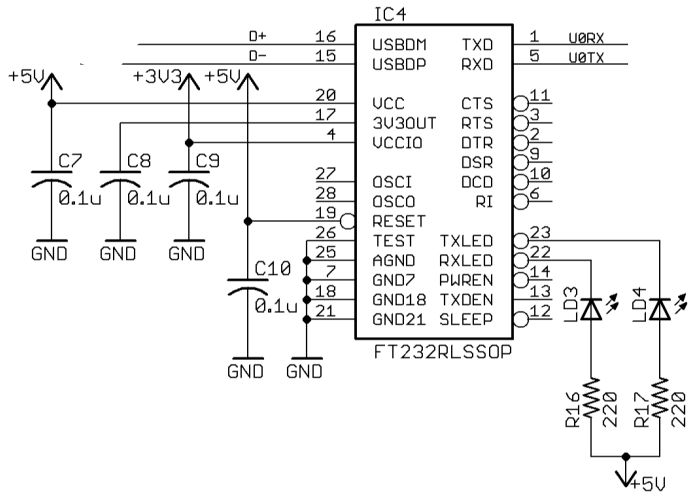


Figure 8.5 USB Virtual COM Port

8.6 Audio Input and Amplifier

The Audio amplifier is a two-stage, ac-coupled microphone amplifier. The microphone that we have used is a capacitive microphone and it requires a biasing voltage of +5V supplied through R12. The supply voltage to the LM358 is +5V and is taken directly from the USB bus. The LM358 is a two-stage amplifier, the first stage is in non-inverting configuration and the other is a voltage follower. Resistors R10 and R11 provide a bias

voltage equal to one-tenth the supply voltage to input of the first op-amp. But the DC gain of the op-amp is very low, by virtue of high resistance R9 and the capacitor C15. For alternating voltage, the capacitive impedance is low and provides high AC gain determined by resistors R7 and R8. The output of the LM358 is connected to ADC6 of the microcontroller.

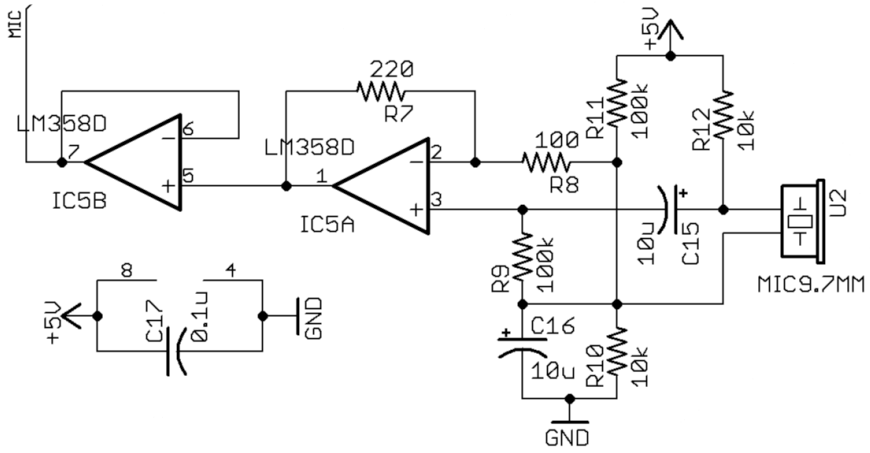


Figure 8.6 Audio Input and Amplifier

8.7 Temperature Sensor and Thumbwheel Potentiometer

To evaluate the analog to digital conversion capabilities of the microcontroller, a LM35 temperature sensor(IC6) and a thumbwheel potentiometer(POT1) are provided on the board. The two are connected to the same analog channel, ADC7 of the microcontroller. Either can be selected by using jumper J2. The LM35 is powered by +5V coming from the USB bus. Figure 8.7 shows the circuit.

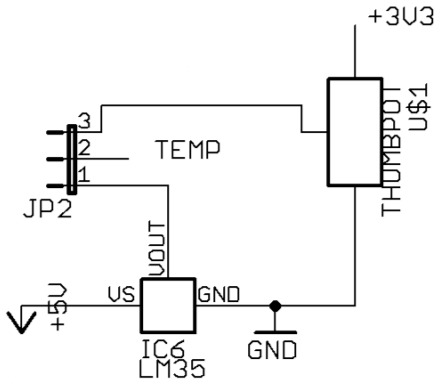


Figure 8.7 Temperature Sensor and Potentiometer

8.8 Light Sensor

The kit uses a LED as a sensor (LED2) to measure light intensity. The LED is reverse biased such that its internal capacitance charges to the voltage applied. It is then connected to an input pin, which measures the time until the voltage across the internal capacitance

becomes zero again. This measure of the counter/time is inversely proportional to the light falling on the LED. The circuit is shown in Figure 8.8.

8.9 I/O Peripherals – Switches and LEDs

The kit consists of three unicolor LEDs: LD5, LD6 and LD7, one RGB LED: RGB1 and two user pushbutton: S2 and S3. S2 and S3 are connected to PE0 and PE1 respectively. The cathodes of LEDs are connected to pins PC5, PC6 and PC7 of the micro controller. Choice can be made between the three unicolor LEDs and the RGB LED by placing the appropriate jumper on J1. The anodes of all LEDs are connected to +3.3V. The LEDs are connected to the Capture/Compare/PWM (CCP) and hence can be used with Pulse Width Modulation for intensity control. The connections are shown in Figure 8.9.

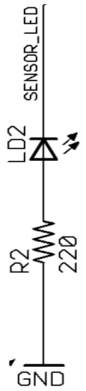


Figure 8.8

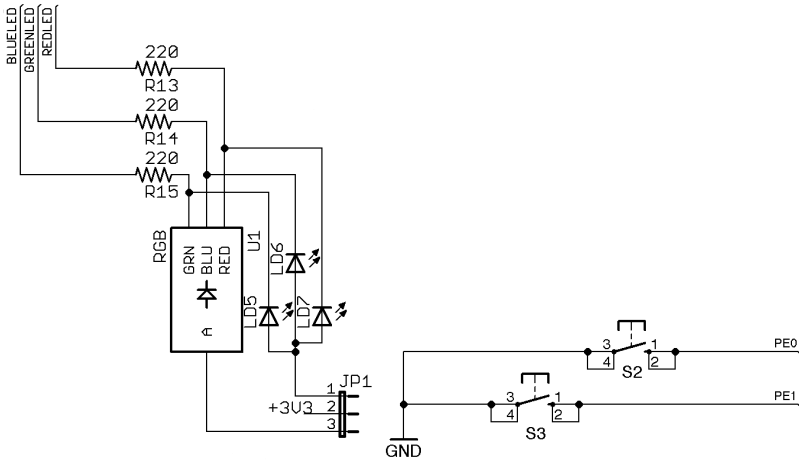


Figure 8.9

9. Stellaris Guru Component List

Component	Value	Component	Value
R1	220 ohm, 0805	F1	PTC Resettable, 500mA, 0805
R2	1k ohm, 0805	IC1	TI LM3S608 MCU, TQFP48
R3	10k ohm, 0805	IC2	Arduino Compatible Interface
R4	10k ohm, 0805	IC3	LM1117-3.3 LDO Regulator
R5	10k ohm, 0805	IC4	FTDI FT232RL, 28SSOP
R6	10k ohm, 0805	IC5	LM358 Dual Op-amps, SO08
R7	220 ohm, 0805	IC6	LM35 Temp Sensor, TO92
R8	100 ohm, 0805	J1	JTAG 20-Pin
R9	100k ohm, 0805	JP1	LED Selector
R10	10k ohm, 0805	JP2	Analog Sensor Selector
R11	100k ohm, 0805	LED1	Red 0805
R12	10k ohm, 0805	LED2	Red 0805
R13	220 ohm, 0805	LED3	Red 0805
R14	220 ohm, 0805	LED4	Red 0805
R15	220 ohm, 0805	LED5	Green 0805
R16	220 ohm, 0805	LED6	Blue 0805
R17	220 ohm, 0805	LED7	Red 0805
C1	10uF/16V Tantalum	POT1	Thumb pot
C2	10uF/16V Tantalum	Q1	8MHz
C3	0.1uF, 0805	RGB1	LED Tricolor, PLCC4
C4	0.1uF, 0805	S1	10XX Pushbutton - Reset
C5	0.1uF, 0805	S2	10XX Pushbutton
C6	0.1uF, 0805	S3	10XX Pushbutton
C7	0.1uF, 0805	U2	9.7MM Microphone
C8	0.1uF, 0805	X1	Mini USB-B
C9	0.1uF, 0805		
C10	0.1uF, 0805		
C11	22pF, 0805		
C12	22pF, 00805		
C13	1uF, 1206		
C14	0.1uF, 0805		
C15	10uF/16V Tantalum		
C16	10uF/16V Tantalum		
C17	0.1uF, 0805		

10. Software Tools

Now we come to developing software for your ARM-based microcontroller board, the Stellaris Guru. This and the next section document the entire tool chain installation process and how to acquire the various software components.

In this manual, we describe how to set up your very own ARM tool chain using free and open source tools unbounded by any trial periods or code size limitations.

The tools required to set up your tool chain are:

- The Eclipse Integrated Development Environment (IDE)
- Java Runtime Edition (required by Eclipse IDE).
- CodeSourcery G++ Lite cross compilers.
- GNU ARM Eclipse Plugin.
- Texas Instruments Development Package which includes the Stellaris Peripheral Driver Library and LM Flash Programmer required by the Stellaris Guru development board.
- FTDI Driver for USB-Serial interaction with the target board.

The next section describes how to set up the tool chain on your computer. To successfully set up the IDE, follow each and every step and compare your screen output with the screen snapshots presented in this manual.

11. Setting up the Development Environment

Note: Operating System Required is Windows XP/Vista/7

Before starting with setting up the development environment or tool chain for Cortex M3 microprocessors, we need to first download some of the software modules.

11.1 Downloading Software Modules

1. **Java**
   Install Java from the link: ***http://goo.gl/ekp1M***

2. **Eclipse IDE**:
   Download Eclipse from the link: ***http://goo.gl/9SCXC***
   Move the zip file to "Softwares_Stellaris" folder on your Desktop. Extract it to "C:\Eclipse".

3. **GNUARM Plugin for Eclipse:** Download the plugin from the link: ***http://goo.gl/h5Pj4***
   Move the plugin file to "Softwares_Stellaris" folder on your Desktop.

4. **TI Development Package:** (Stellaris Peripheral Library + FTDI Drivers + LM Flash Programmer): Download the "StellarisWare Driver Library Standalone Package", "Stellaris FTDI driver" and the "LM Flash Programmer" from this link: ***http://goo.gl/PLhVz***
   Install the .exe under "C:\StellarisWare". Install "LM Flash Programmer" under "C:\Program Files".

5. **CodeSourcery G++ Lite:** Download the "Sourcery Code Bench Lite Edition EABI Release" from this link: ***http://goo.gl/za8ta***
   Install the .exe under "C:\Program Files".

6. **FTDI USB Serial Driver:** Download the "VCP Drivers" for supported Architecture from this link: ***http://goo.gl/bPtLX***
   Extract the zip file to "Softwares_Stellaris" folder on your desktop.

11.2 Installation Instructions

11.2.1 Hello Board

Plugging in the board for the first time., the device manager shows up as shown in Figure 11.1.

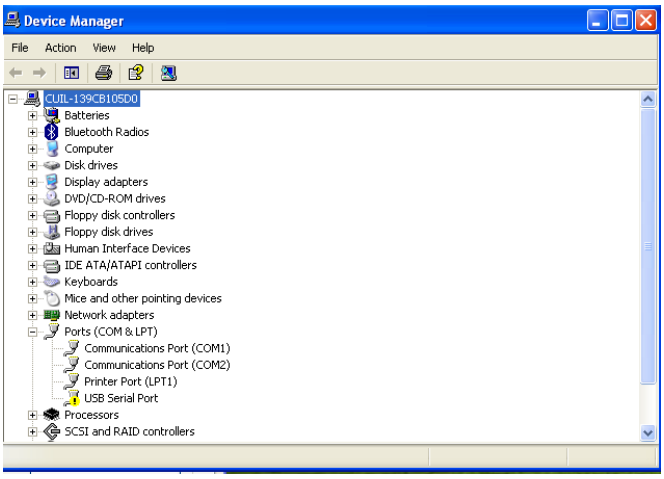


Figure 11.1 Device Manager

In the Ports (COM & LPT) section, an unidentified device named USB Serial Port will show. This is your target board. Install the driver by specifying the location of the "FTDI VCP" subfolder in the Software_Stellaris package. This should enable it to pick up the FTDI driver on its own.

11.2.2 Setting up Eclipse and LM Flash Programmer

i) Fire up the Eclipse IDE. A window similar to that shown in Figure 11.2 will crop up.

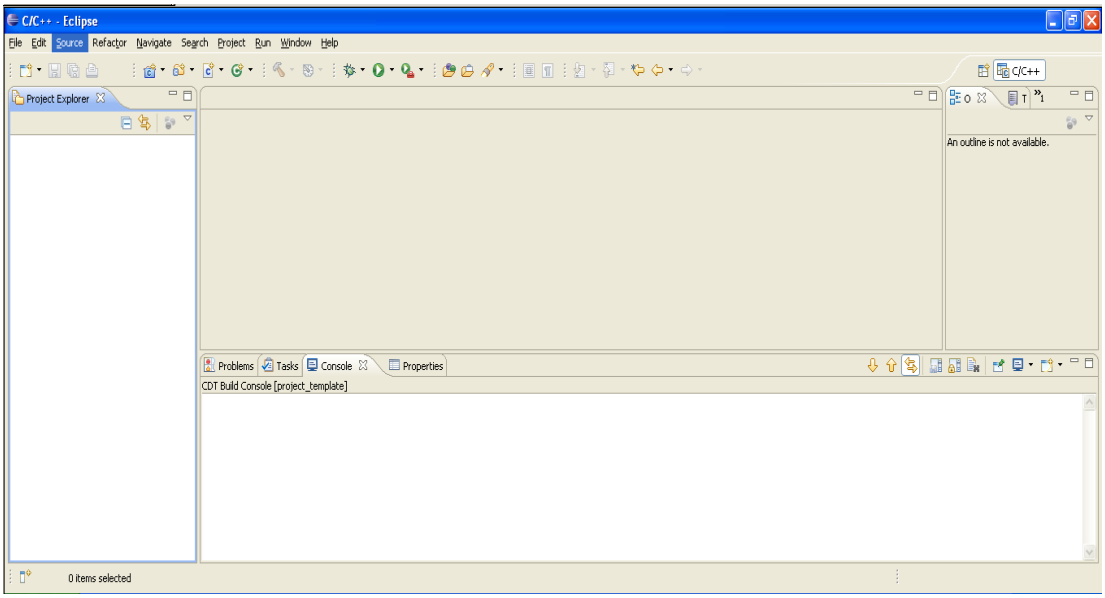


Figure 11.2 Eclipse IDE

ii) Installing the GNU ARM Plugin

- Go to Help -> Install New Software -> Add -> Archive
- A window similar to that shown in Figure 11.3 will show.
- Browse to the GNUARM plugin zip file downloaded under “Softwares_Stellaris”.
- Click Ok and Next to install the plugin.
- A window similar to that shown in Figure 11.4 will show.

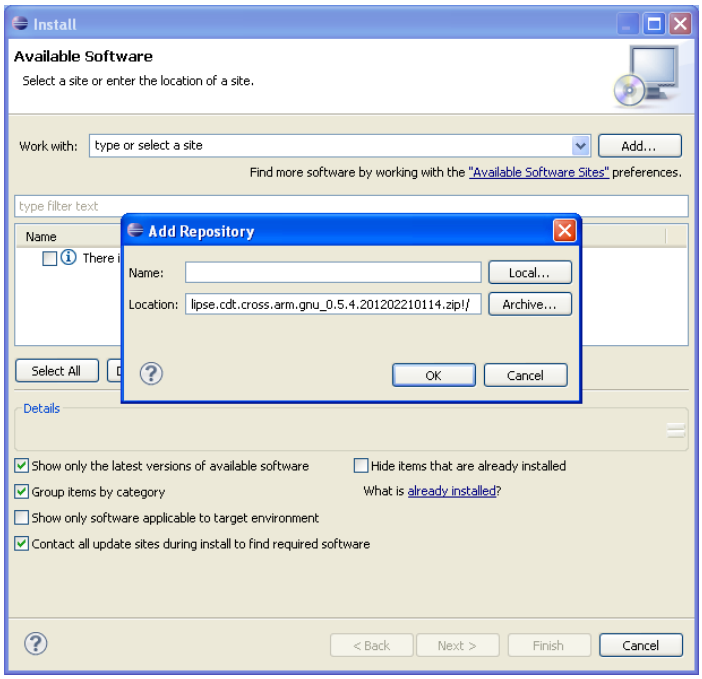


Figure 11.3 Add Repository

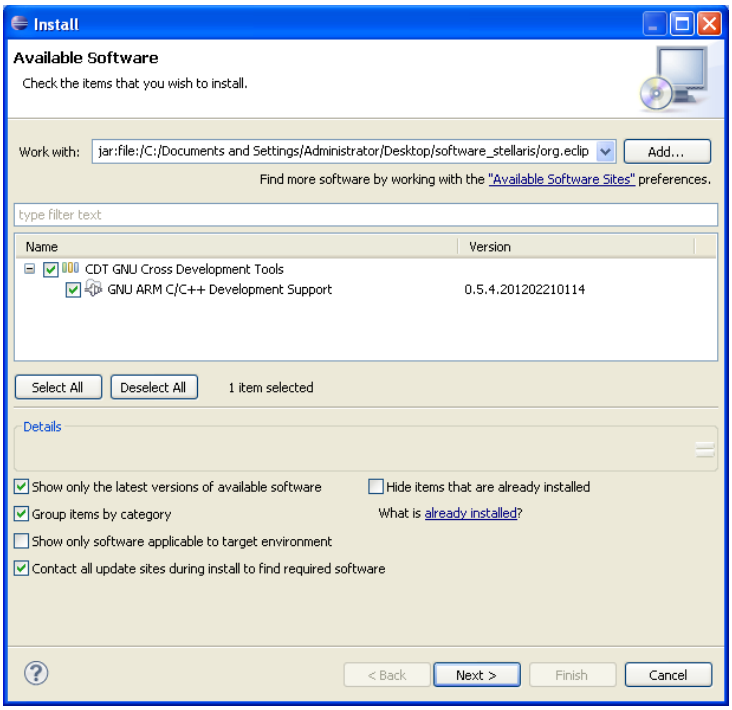


Figure 11.4 Plugin Install Option

iii) Creating your first project

- Go to File -> New -> C Project.
- Enter the project name. Make sure the selections are as shown in Figure 11.5.
- Click Next.
- Make sure the selection are same as shown in Figure 11.6.
- Then, press Finish.

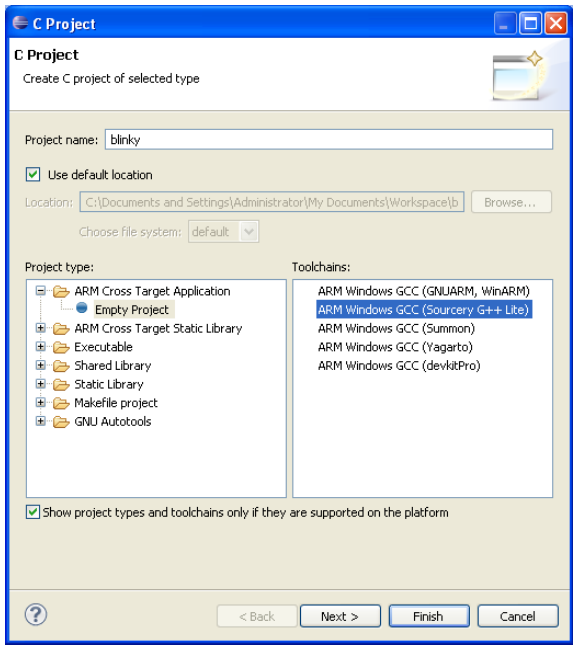


Figure 11.5 New Project Window

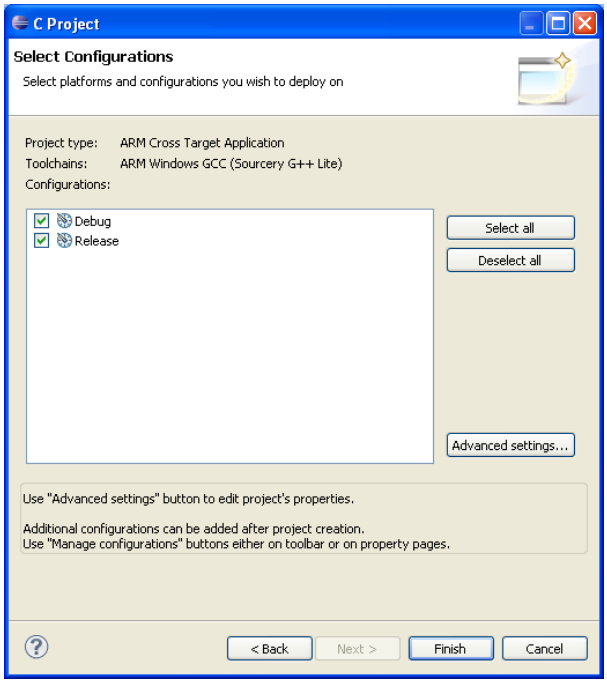


Figure 11.6 Configuration Window

iv) Importing the existing file system.

- Go to File -> Import -> General -> File System.
- A window similar to that shown in Figure 11.7 will crop up.
- Press Next.
- Browse and select the any project from the lm3s608 folder under "Softwares_Stellaris" on your desktop. Make sure same option as shown in Figure 11.8 are selected. Press Finish.

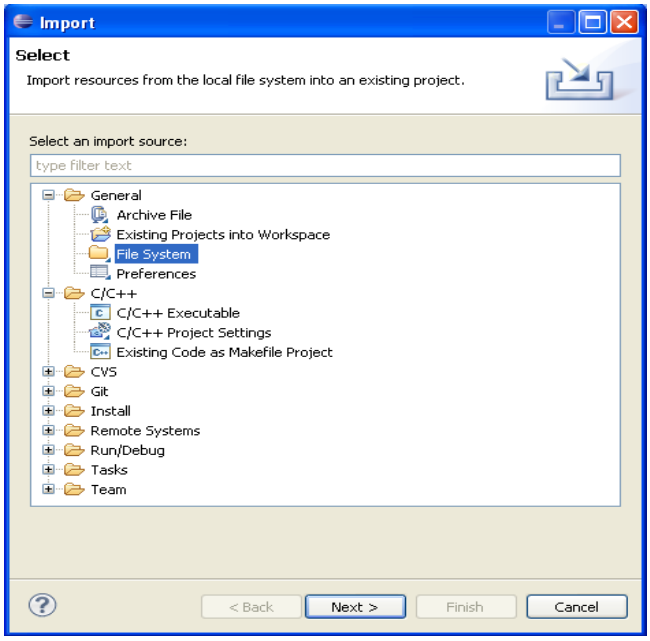


Figure 11.7 Import Window

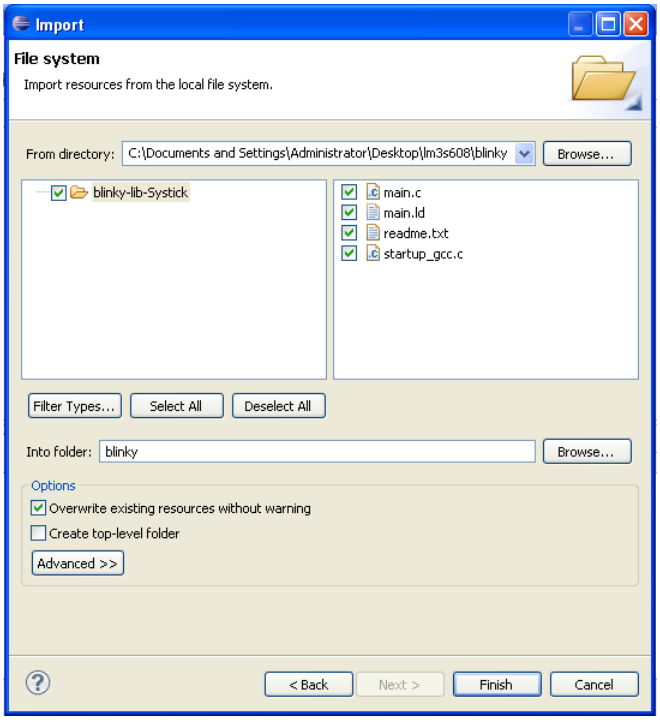


Figure 11.8 File System Window

So till now, we have set up a new project and instructed the Eclipse IDE as to what cross compiler to use for development. Your screen should look similar to that shown in Figure 11.9.

.

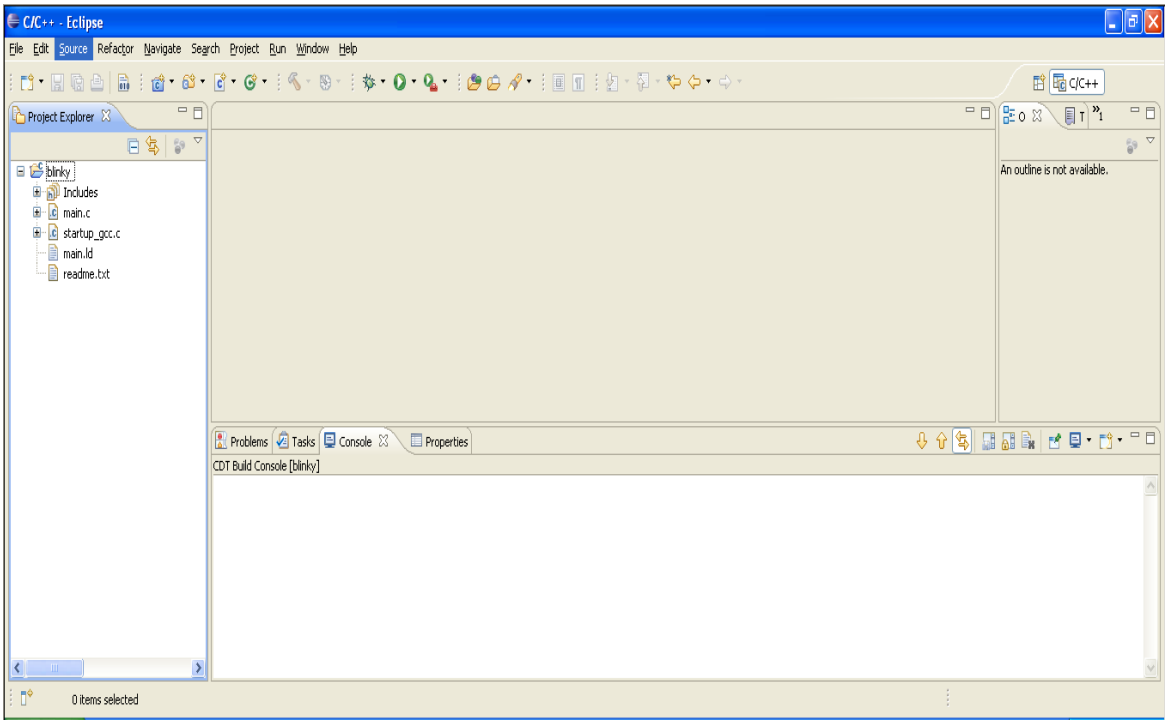


Figure 11.9 Eclipse IDE Window

v) Setting up project properties

- Go to Project Properties.
- Under Settings -> Tool Settings -> ARM Sourcery Windows GCC C Compiler -> Directories. Include the directory paths as shown in Figure 11.10 and 11.11.

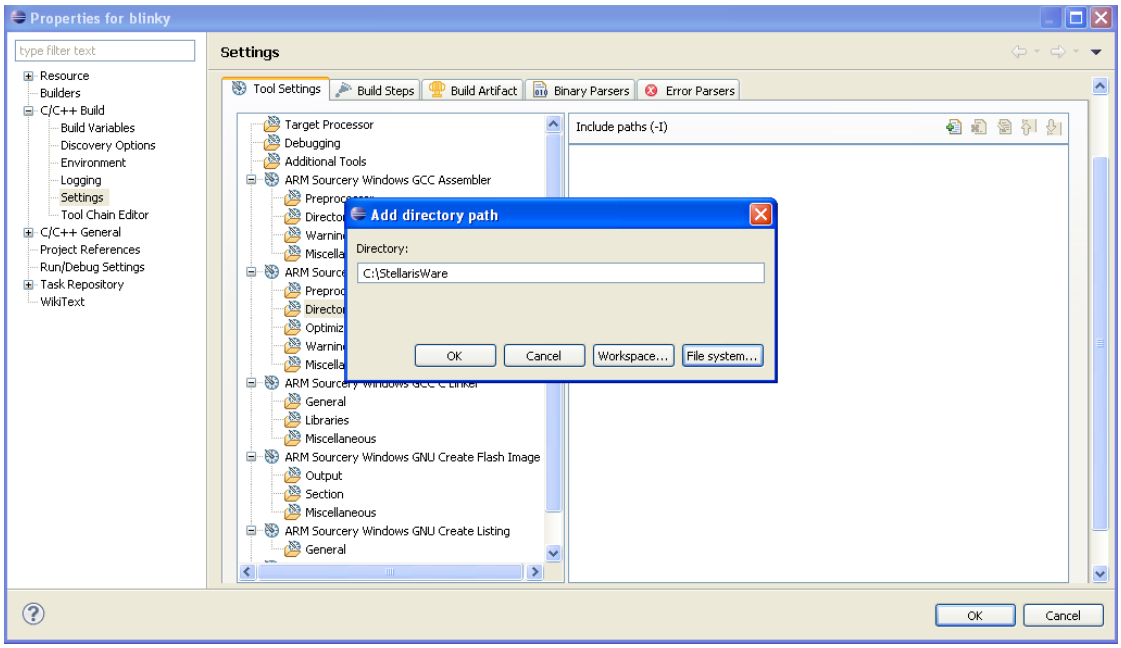


Figure 11.10 Project Properties

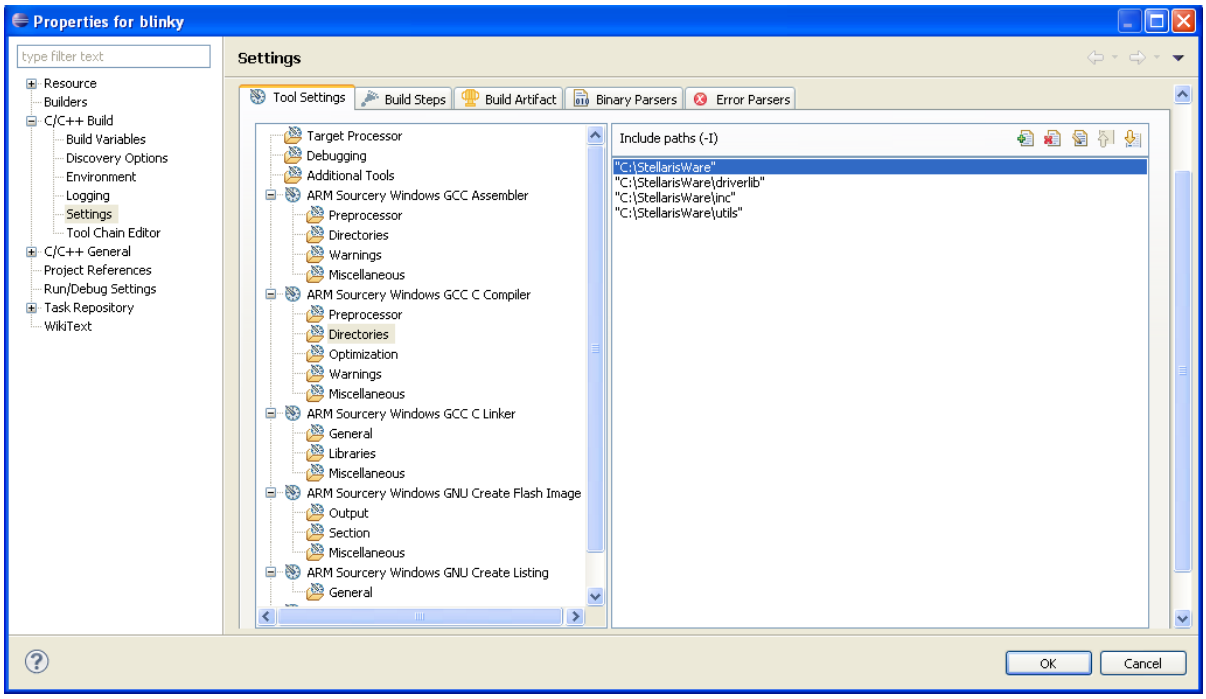


Figure 11.11 Directories

- In the Project Properties window, go to Tool Setting -> ARM Sourcery Windows GCC C Compiler -> Preprocessors and define the symbol “gcc” as shown in Figure 11.12.

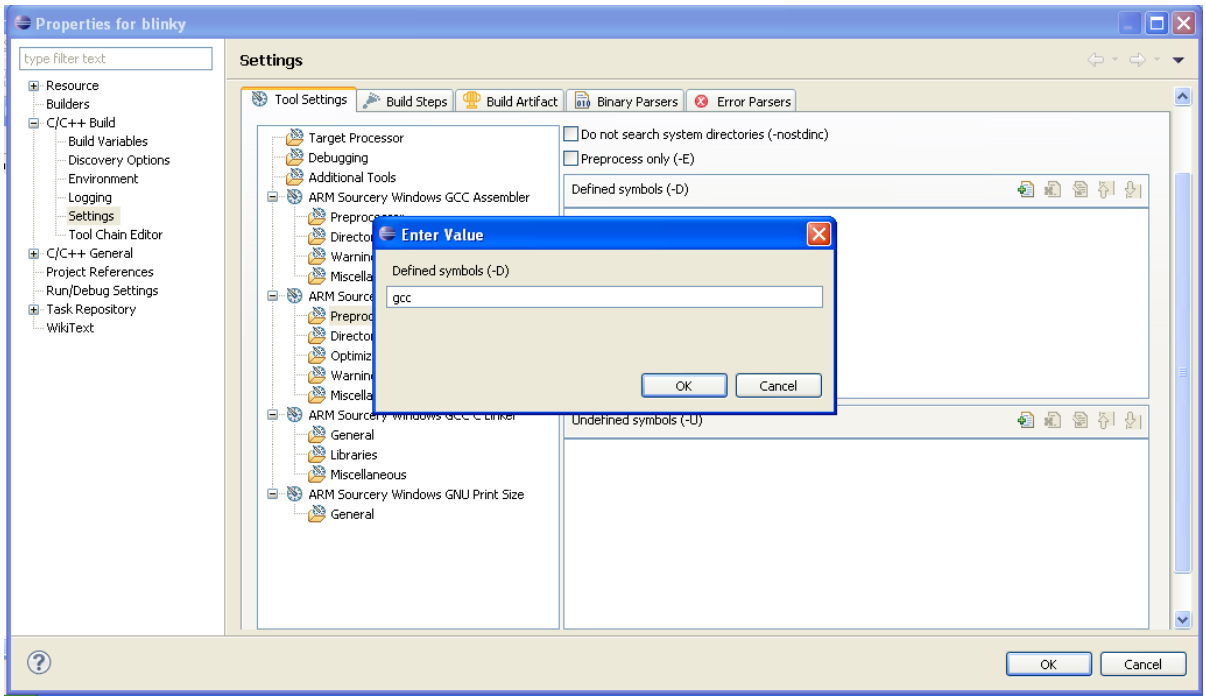


Figure 11.12 Preprocessors

- Setting up code optimization. In the Project Properties window, go to Tool Setting -> ARM Sourcery Windows GCC C Compiler -> Optimization.
- From the drop down menu next to Optimization Levels, select "None (-O0)" and make selections as shown in Figure 11.13.

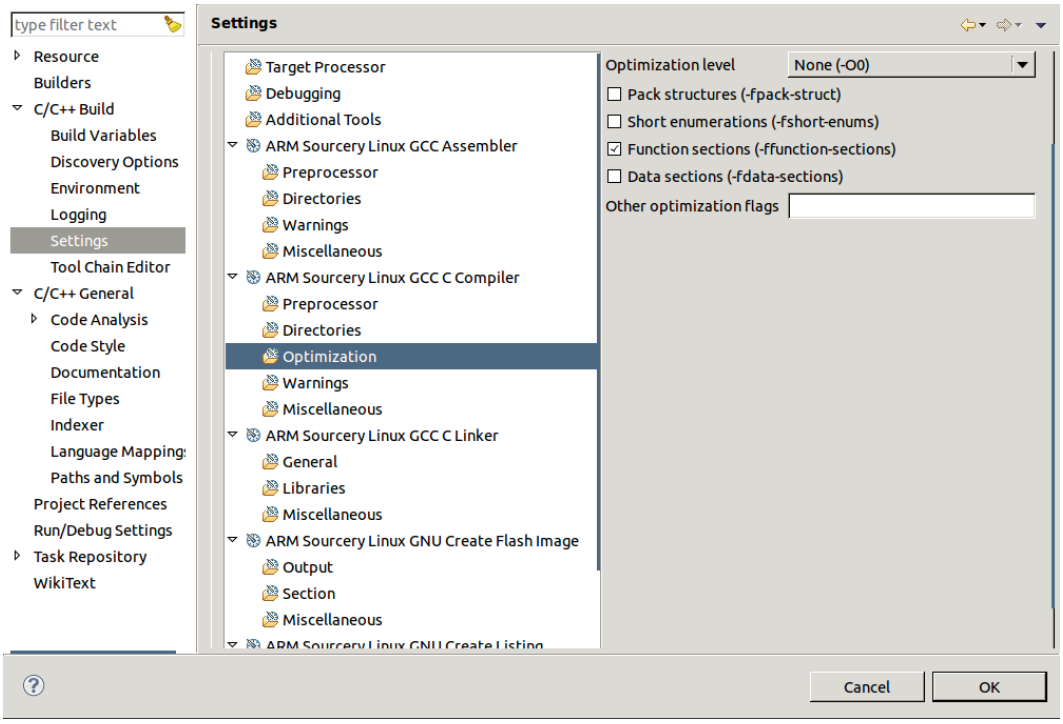


Figure 11.13 Optimization

- Including the linker. In the Project Properties window, go to Tool Setting -> ARM Sourcery Windows GCC C Linker -> General.
- Browse and select the linker file copied to the workspace. In our case, it is “main.ld”.
- Make sure the same options are selected as shown in Figure 11.14.

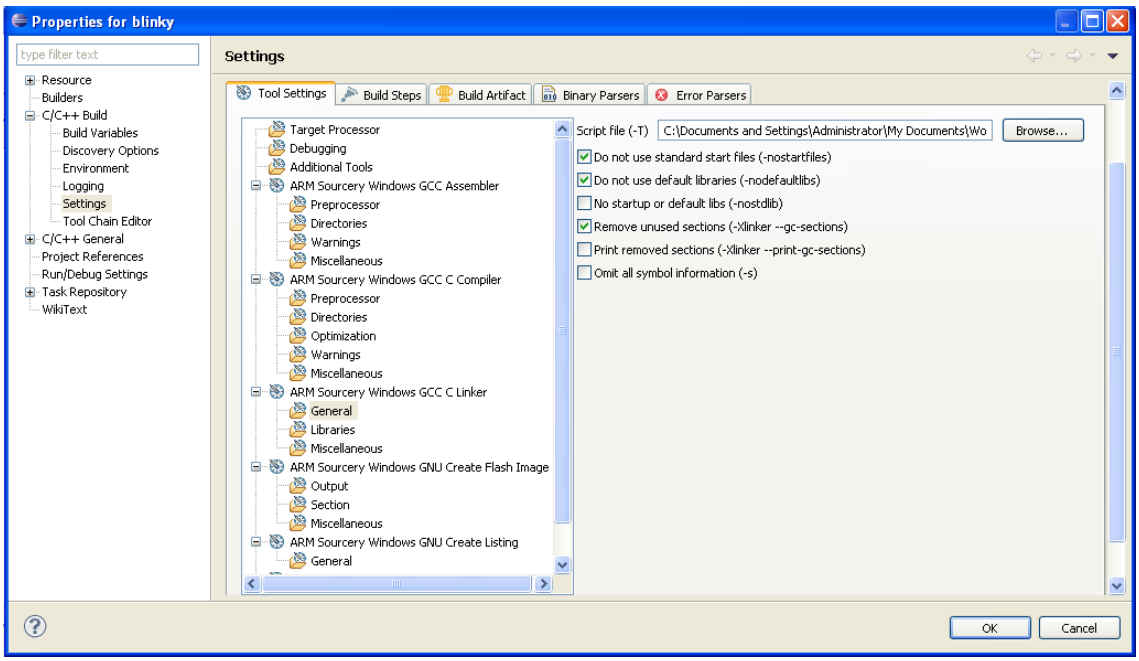


Figure 11.14 Linker File

- Add the libraries. In the Project Properties window go to Tool Setting -> ARM Sourcery Windows GCC C Linker ->Libraries.
- Select Add Directory Path. A window similar to that shown in Figure 11.15 will be displayed.
- Add the two directory paths as shown in Figure 11.16.
    - C:/StellarisWare
    - C:/StellarisWare/driverlib/gcc-cm3

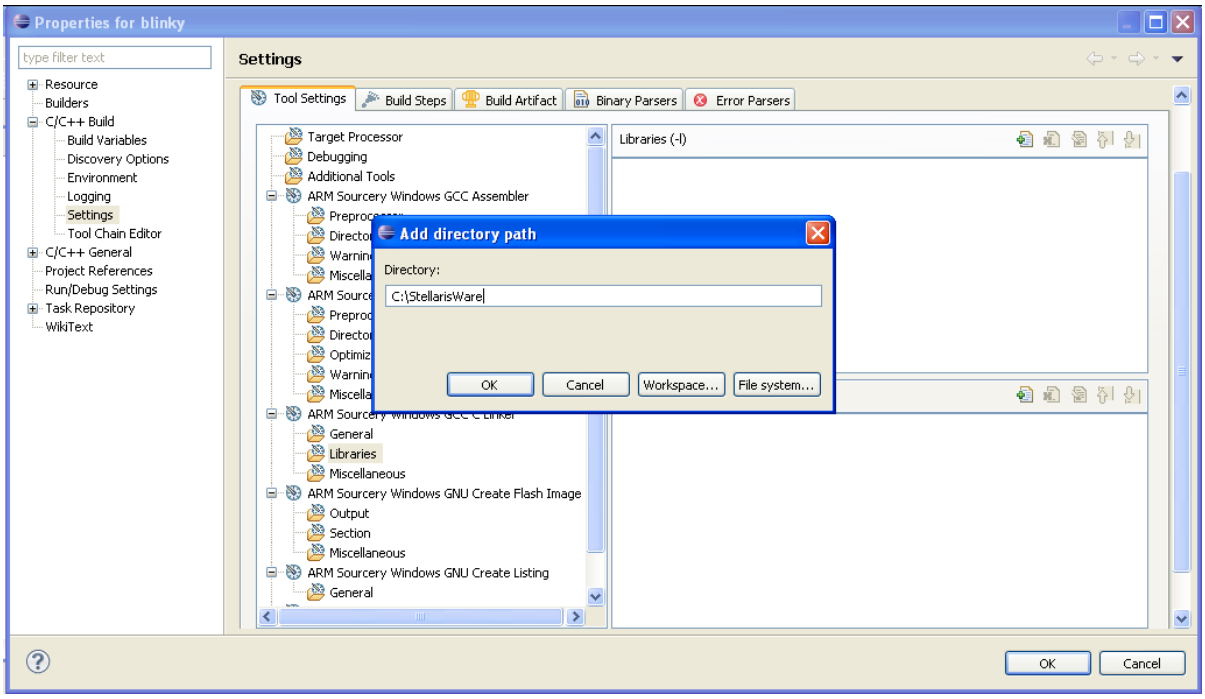


Figure 11.15 Add Directory Path

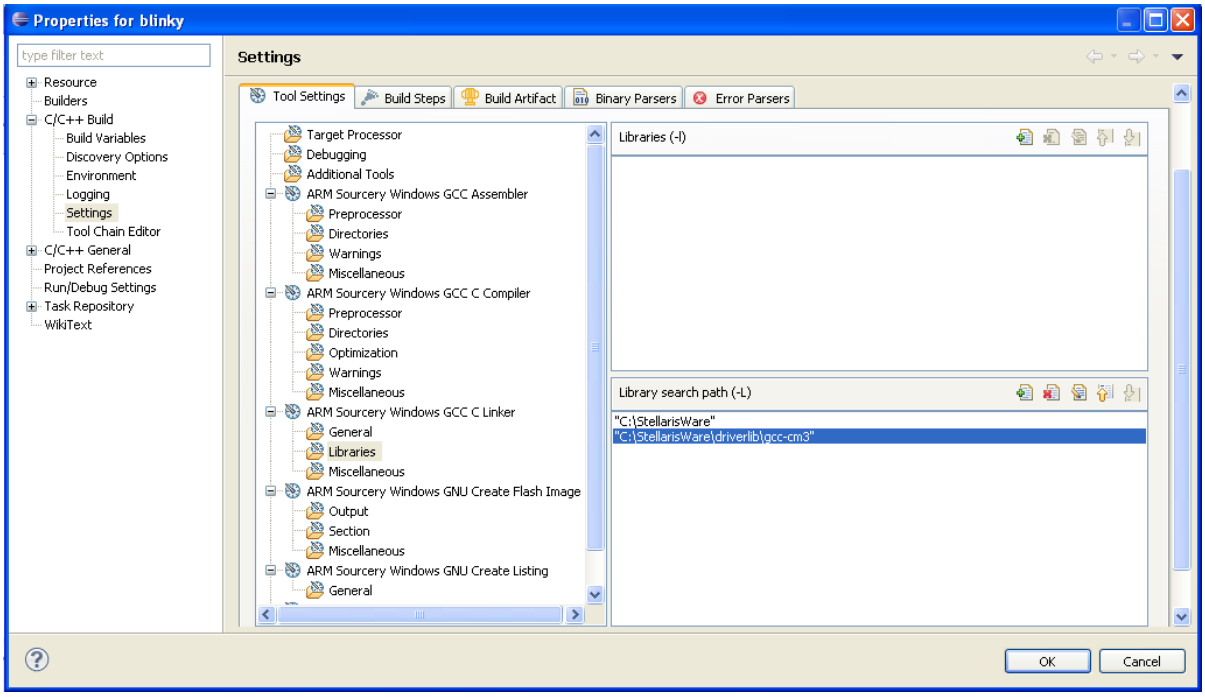


Figure 11.16 Add Directory Path

- Include source location. In the Project Properties window, go to C/C++ General -> Paths and Symbols -> Source Location -> Link Folder. Include the *driverlib* and *utils* folder, which contain the C source files. These are placed inside the StellarisWare folder in the C: drive. This is shown in Figures 11.17 and 11.18.

- Include the files as shown in Figure 11.19 and select Apply.

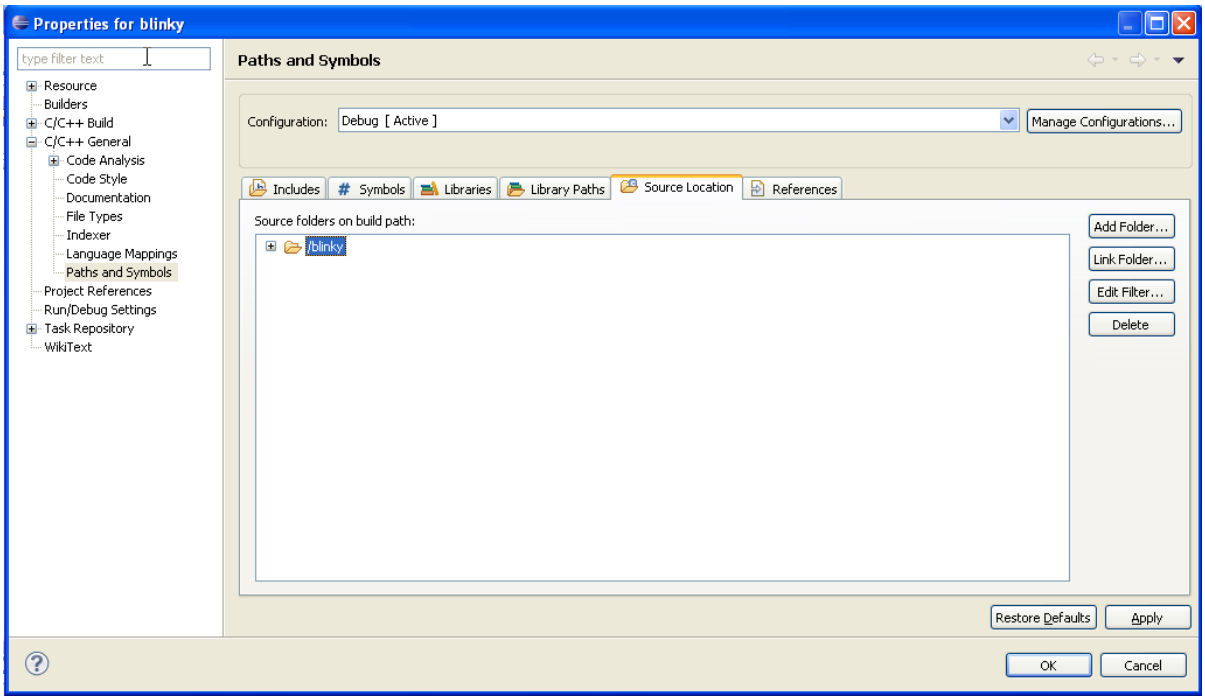


Figure 11.17 Paths and Symbols

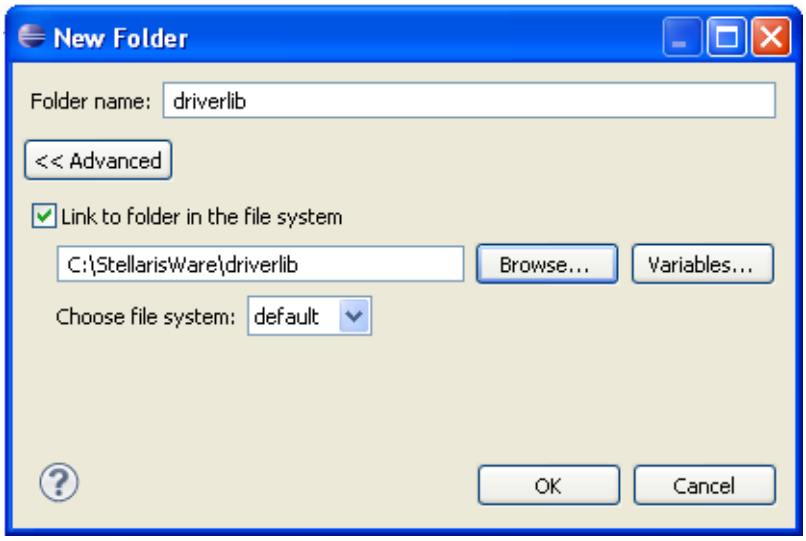


Figure 11.18 Link Folder

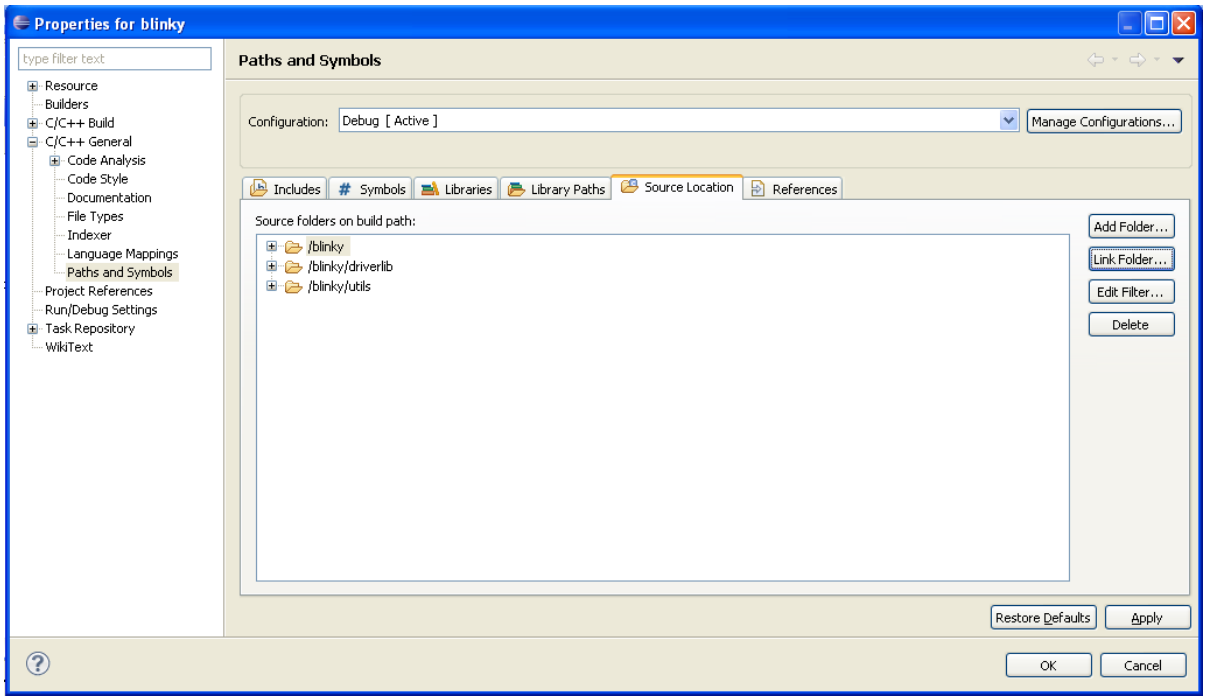


Figure 11.19 Paths and Symbols

- Post Build Steps. In the Project Properties window, go to C/C++ Build -> Settings -> Build Steps -> Post Build Steps Command and type the following:

  arm-none-eabi-objcopy -S -O binary "${ProjName}.elf" "${ProjName}.bin"

- Your windows should look similar to that shown in Figure 11.20. Press OK.

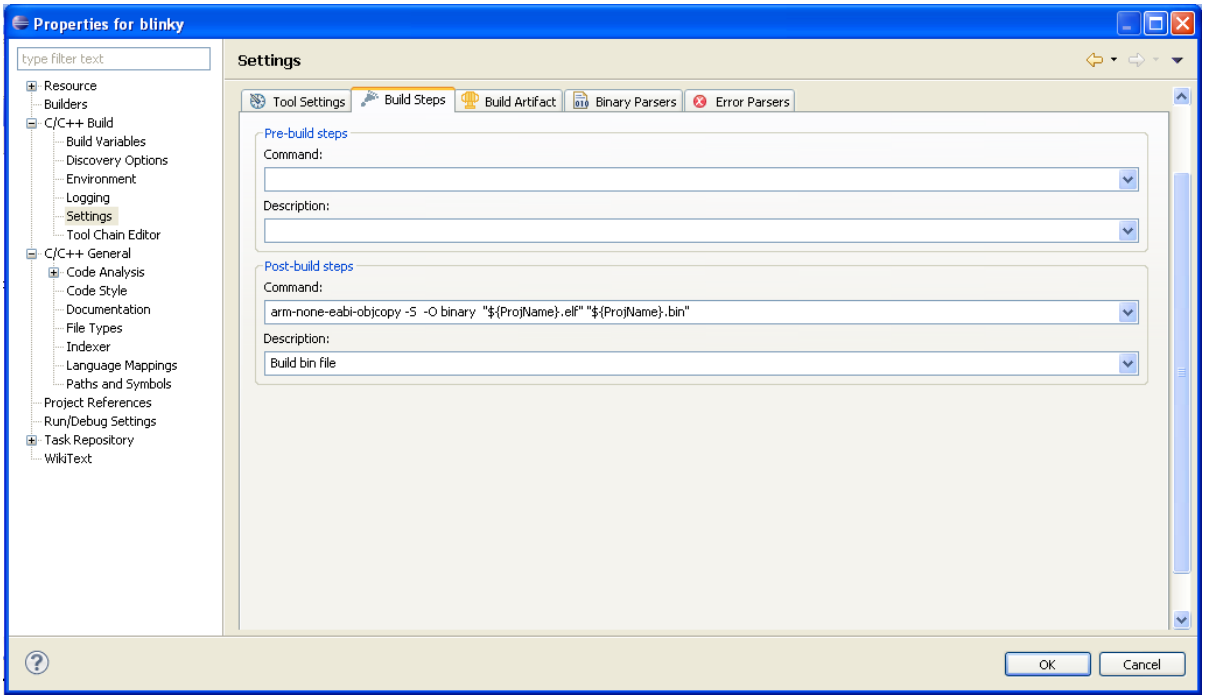


Figure 11.20 Post Build Steps

- After returning back to the main Eclipse IDE window, in your workspace, select main.c
- Your screen should look similar to that shown in Figure 11.21

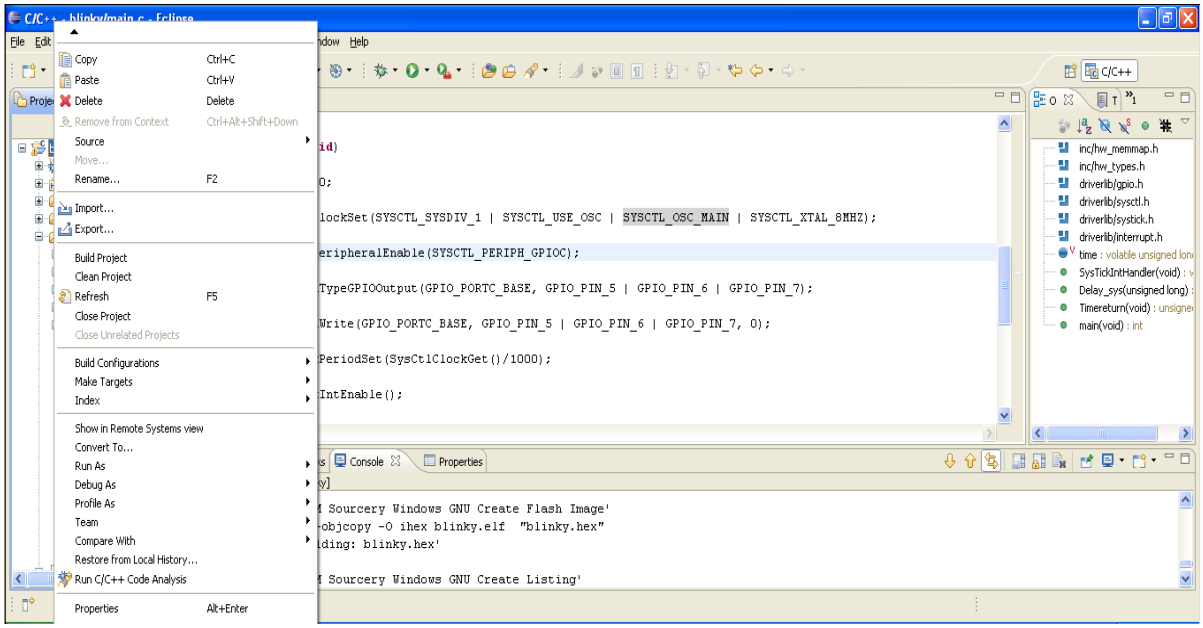


Figure 11.20 Main Eclipse IDE Window

- Right click on the project opened in the workspace, go to Build Configurations and build and clean the project. Shown in Figure 11.21. If everything goes right, it will build and clean with ease.

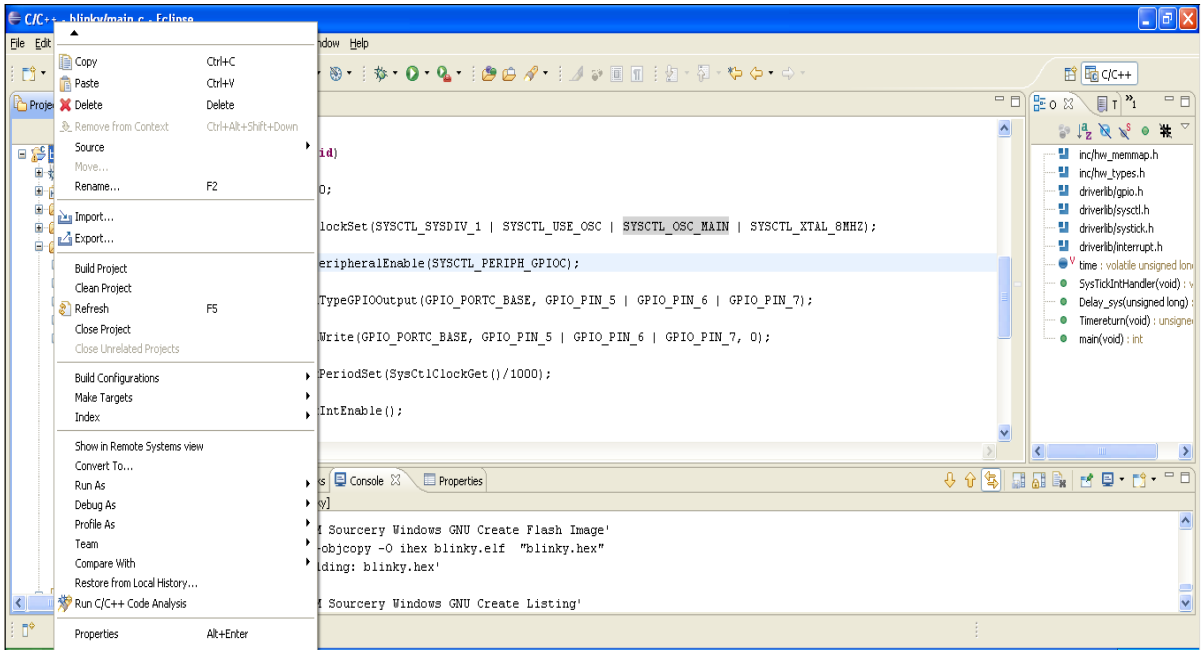


Figure 11.21 Build and Clean Project

vi) Integrating LM Flash Programmer with Eclipse IDE

- Go to Run -> External Tools -> External Tools Configuration as shown in Figure 11.22.
- A window similar to that shown in Figure 11.23 should be displayed.
- In the window, set the location to LMFlash.exe located under “C:/Program Files/Texas Instruments/”
- Clicking Run launches the LM Flash Programmer.

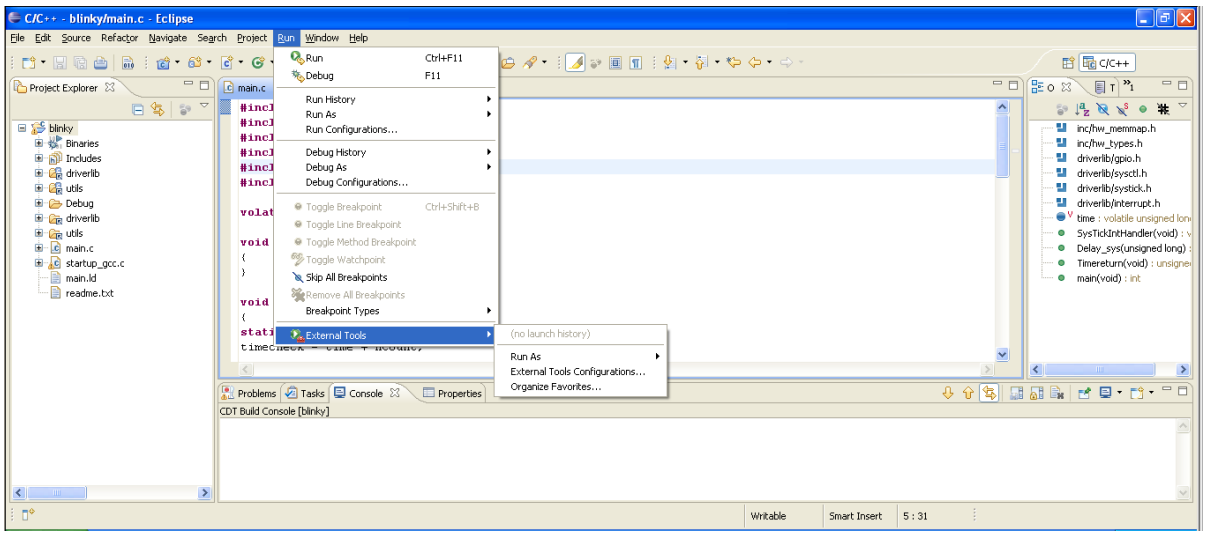


Figure 11.22 External Tool

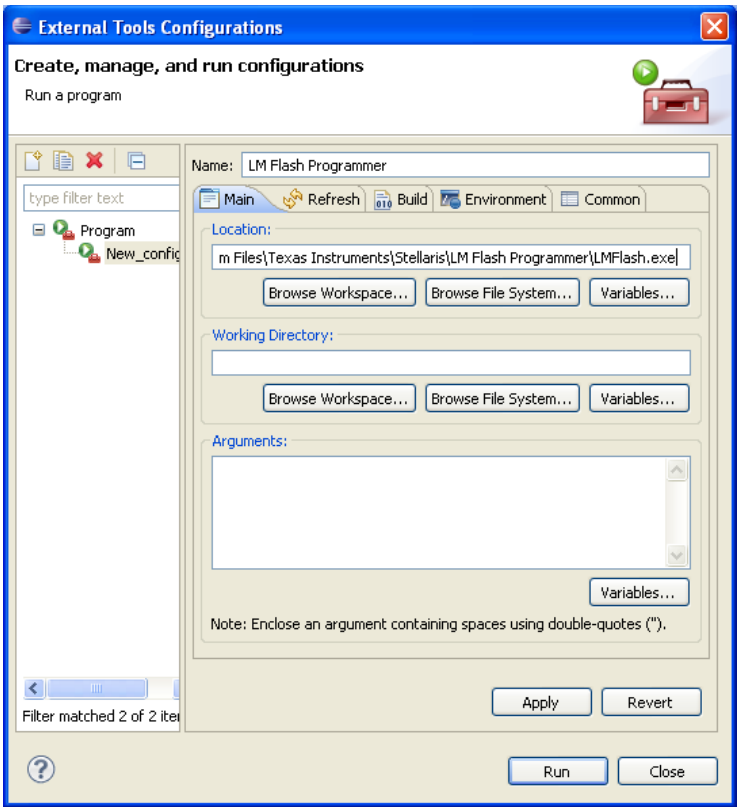


Figure 11.23 External Tools Configuration

11.2.3 LM Flash Programmer

- After clicking on Run, a window similar to that shown in Figure 11.24 should display on the screen. This is the LM Flash Programmer.
- In the configuration tab, chose the COM port depending on the port the device is attached to.

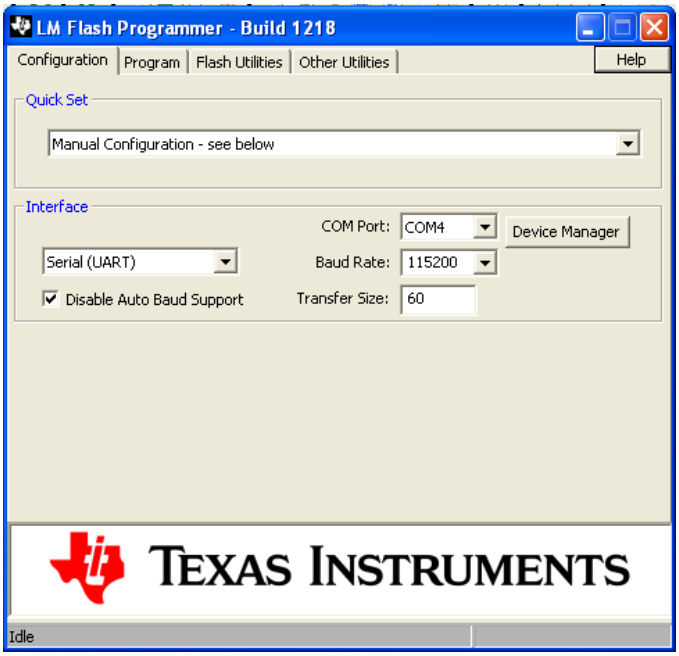


Figure 11.24 LM Flash Programmer (Configuration Tab)

- Go to the Program tab, and select the .bin file by browsing to your project in the Eclipse workspace folder.
- Chose the same options as shown in Figure 11.25 with the same parameters. Make sure the address offset is 0800.

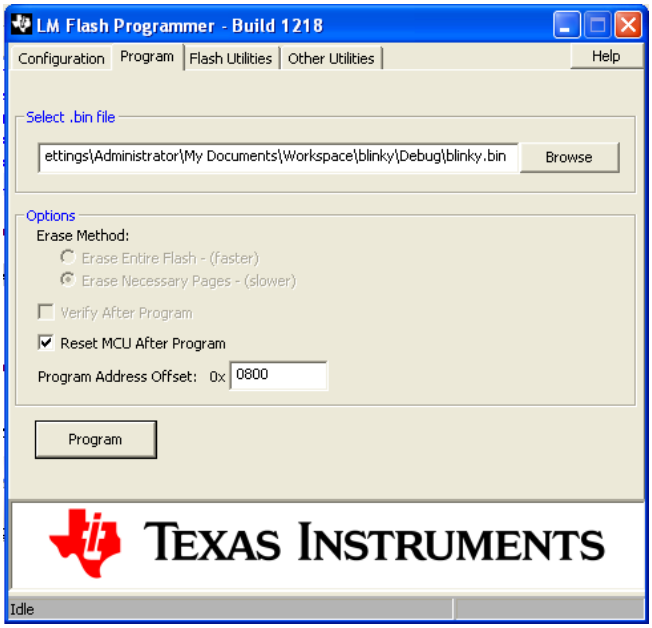


Figure 11.25 LM Flash Programmer (Program Tab)

- **To upload your application program to the Stellaris Guru kit you need to get the kit in the "Program Update" mode. Pressing the Reset and S2 switches simultaneously and then releasing the Reset switch followed by S2 brings the microcontroller on the Stellaris Guru kit in the program update mode.**
- Clicking on Program starts transferring the application binary file to the Stellaris Guru board as shown in Figure 11.26.

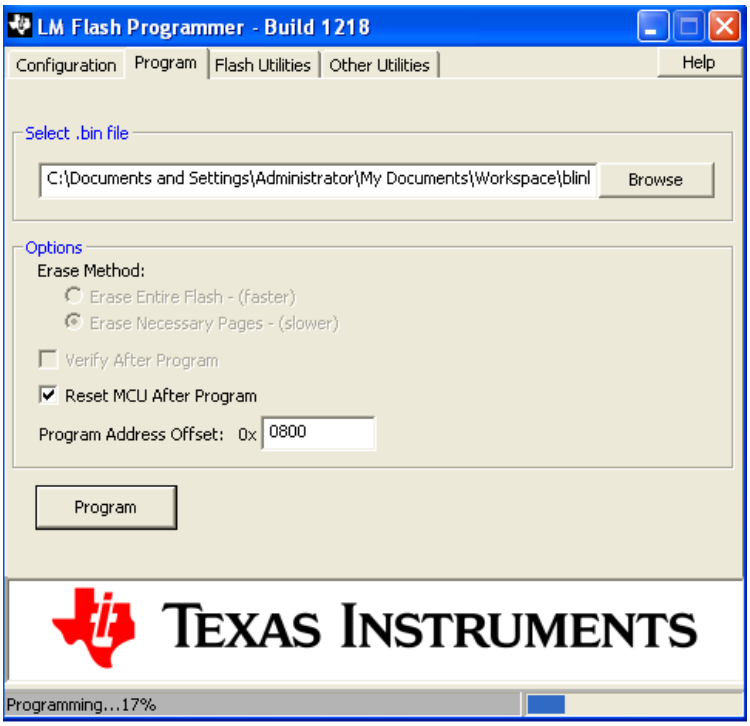


Figure 11.26 Programming the Target

- If you followed the earlier steps correctly, you have just programmed your very first ARM-powered microcontroller. Congratulations!

11. List of Possible Experiments

We have compiled a small to-do experiment list around the Stellaris Guru kit, which helps in evaluating each and every component on the board and every feature available on the Stellaris ARM core like timers, watchdog, brownout detector etc. Of course, a lot more experiments with the kit can be done apart from those listed.

1. Blink an LED.
2. Blink an LED using delay generated using the SysTick timer.
3. Blink LEDs in a controlled pattern.
4. System clock real time alteration using the PLL modules.
5. Control intensity of an LED using PWM implemented in software.
6. Control intensity of an LED using PWM implemented in hardware.
7. Control intensity of an RGB LED to generate composite colors using PWM implemented in software.
8. Control intensity of an RGB LED to generate composite colors using PWM implemented in hardware.
9. Control an LED using a switch by polling method.
10. Control an LED using a switch by interrupt method and flash the LED once every five switch presses.
11. UART Echo Test.
12. Control intensity of an LED on parameters received over UART.
13. Take analog readings on rotation of a rotary potentiometer connected to an ADC channel.
14. Temperature indication on an RGB LED.
15. Display temperature on PC by sending values over UART.
16. Mimic light intensity sensed by the light sensor by varying the blinking rate of an LED.
17. Plot light intensity sensed by the light sensor on PC.
18. Evaluate the various sleep modes by putting core in sleep and deep sleep modes.
19. System reset using the Watchdog timer in case something goes wrong.
20. Sample sound using a microphone and display sound levels on LEDs.
21. Filter the sound input using three filters of high, medium and low frequencies and show output on LEDs, one for each filter.
22. Sample sound and plot amplitude v/s time on PC.
23. Sample sound and analyze its spectrum using FFT and plot amplitude v/s frequency results on a PC.
24. Generate a Real time clock using the 32-Bit timers and output time over UART.

From the above list, we have included two experiments in this manual to help you get started quickly with the StellarisWare Peripheral Driver Library. Experiment numbers one and eleven have been explained in the next section.

12. Sample Experiments

12.1 Blink a LED

/*This experiment blinks the LED connected on pin PC5 at a constant rate with delay of one second using the System Control function*/

```
/* Defines the base address of the memories and peripherals */
#include "inc/hw_memmap.h"
/* Defines the common types and macros */
#include "inc/hw_types.h"
/* Defines and Macros for GPIO API */
#include "driverlib/gpio.h"
/* Prototypes for the system control driver */
#include "driverlib/sysctl.h"

int main(void)
{
    /* Set the clocking to directly run from the crystal at 8MHz */
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
    SYSCTL_XTAL_8MHZ);

    /* Set the clock for the GPIO Port C */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);

    /* Set the type of the GPIO Pins as Output on which the LEDs are connected*/
    GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7);

    /* GPIO Pins 5, 6, 7 on PORT C initialized to be LOW */
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, 0);

    while(1)
    {
        /* Make Pin Low */
        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0);

        /* Delay for a second using System Control function*/
        SysCtlDelay(SysCtlClockGet());

        /* Make Pin High */
        GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, GPIO_PIN_5);

        /* Delay for a second using System Control Function */
        SysCtlDelay(SysCtlClockGet());
    }
}
```

12.2 UART Echo Test

/*This experiment evaluates the basic UART functionality. If the character ‘a’ is received over UART, the controller responds back with the character ‘b’. */

```
/* Defines the base address of the memories and peripherals */
#include "../../../inc/hw_memmap.h"
/* Defines the common types and macros */
#include "../../../inc/hw_types.h"
/* Defines and Macros for GPIO API */
#include "../../../driverlib/gpio.h"
/* Prototypes for the system control driver */
#include "../../../driverlib/sysctl.h"

int main(void)
{
    /*Set the clocking to directly run from the crystal at 8MHz*/
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
    SYSCTL_XTAL_8MHZ);

    /*Enable the GPIO port to which the UART0 module is attached and enable the UART0
    module*/
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    /* Make the GPIO pins be UART peripheral controlled. */
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    /*Set UART configuration to run at 115.2kbps, one byte length, no stop bits and parity
    none and clock equal to the System clock.*/
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200
            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
             UART_CONFIG_PAR_NONE));

    while(1)
    {
            */If ‘a’ is received over UART, reply back with a ‘b’*/
            if(UARTCharGet(UART0_BASE)==’a’)
                    UARTCharPut(UART0_BASE,’b’);
    }

}
```

14. Further Reading

If you wish to read more about the ARM Processor Family and specifically more about the Cortex-M3, we recommend you some good reading material in no specific preference order. These books and articles have been used in the development of this manual too.

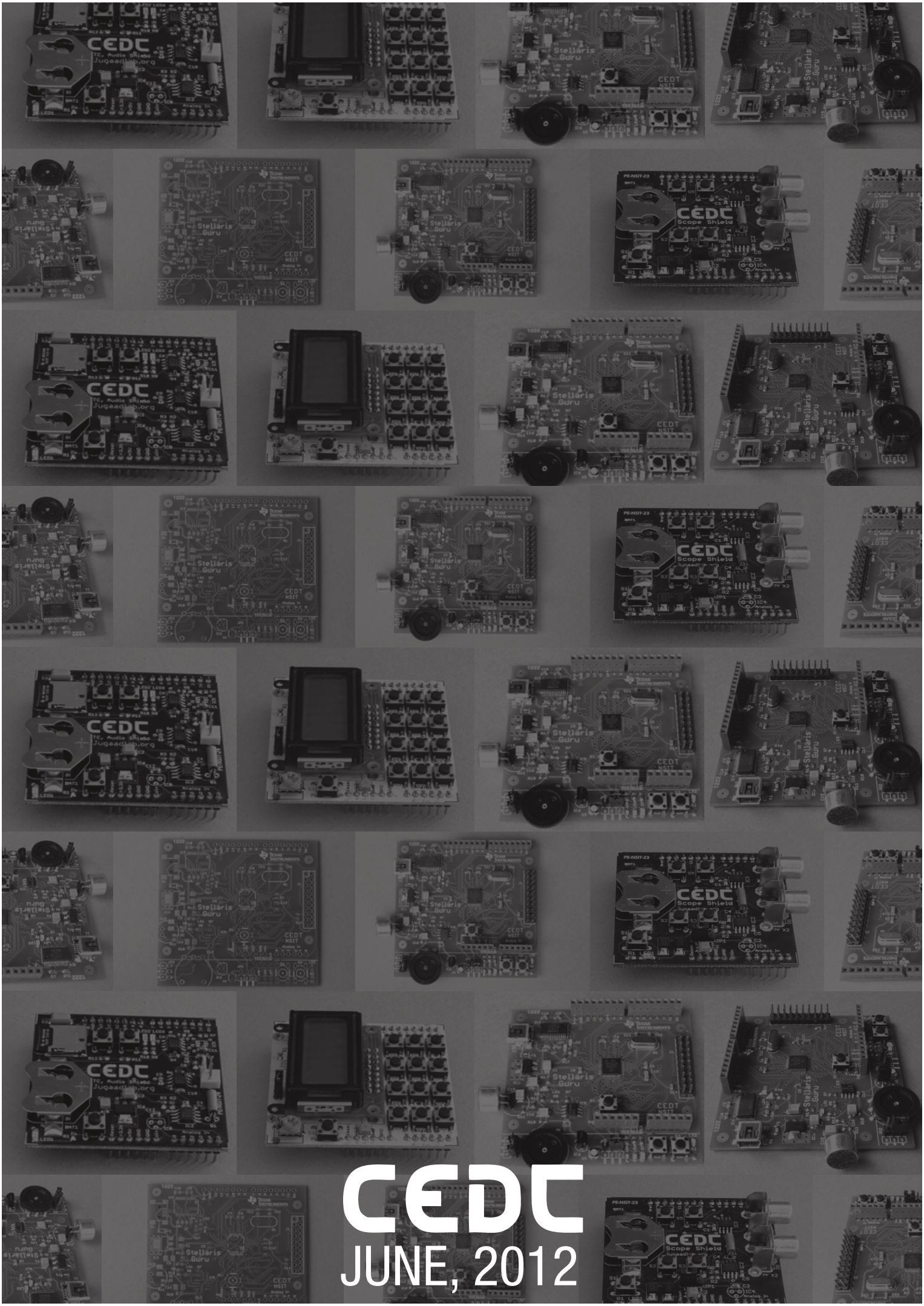
1. *"Definitive Guide to the ARM Cortex M3",* Joseph Yiu, Second Edition, 2010. Elsevier Inc.
2. *"ARM System-On-A-Chip Architecture",* Steve Furber, Second Edition, 2000, Addison Wesley.
3. *"ARM System Developers Guide: Designing and Optimizing System Software",* Andrew N. Sloss, Dominic Symes, Chris Wright, 2004, Elsevier Inc.
4. *"ARM Assembly Language",* William Hohl, 2009, CRC Press
5. *"Embedded System and Real Time Interfacing to the ARM Cortex-M3",* Jonathan M. Valvano, 2011, CreateSpace.
6. *"Stellaris Ware Peripheral Driver Library User Guide",* Build 8555, Texas Instruments

15. Contact Us

In case, you want any help on the ARM processor family, Texas Instruments Stellaris family of microcontroller or any software issues or glitches, which may crop up, you can contact any of the people listed below.

- Dhananjay V. Gadre, Associate Professor, NSIT. *dhananjay (dot) gadre (at) gmail (dot) com*
- Rohit Dureja, UG, NSIT. rohit (dot) dureja (at) gmail (dot) com
- Shanjit Singh Jajmann, UG, NSIT. *shanjitsingh (at) gmail (dot) com*

You can also post your queries on the TI e2e Community and get them solved instantly by an expert team of TI design engineers and product enthusiasts and ARM supporters.


CEDT
JUNE, 2012