TI-RSLK MAX

Texas Instruments Robotics System Learning Kit

TEXAS INSTRUMENTS

Module 3

Introduction: ARM Cortex M

Introduction: ARM Cortex M Architecture

Educational Objectives:

REVIEW Cortex M architecture
UNDERSTAND registers, memory, assembly instructions
DEVELOP logic and arithmetic functions in assembly
LEARN how functions work, and where data is stored
DESIGN, BUILD & TEST A COMPONENT
Nonlinear conversion function for an IR distance sensor

Prerequisites (Module 1)
- Running code on the LaunchPad using CCS (Module 1)

Recommended reading materials for students:
- Chapter 3, **Embedded Systems: Introduction to Robotics**, Jonathan W. Valvano, ISBN: 9781074544300, copyright © 2019

In this class, we will use the TI Launchpad Development Kit with the MSP432 microcontroller, which includes a Cortex-M processor and a suite of input/output devices derived from the MSP430 family of low power microcontrollers. **Architecture** is the manner with which the processor, random access memory (RAM), read only memory (ROM), and input/output (I/O) ports are combined to create the microcontroller. See Figure 1.

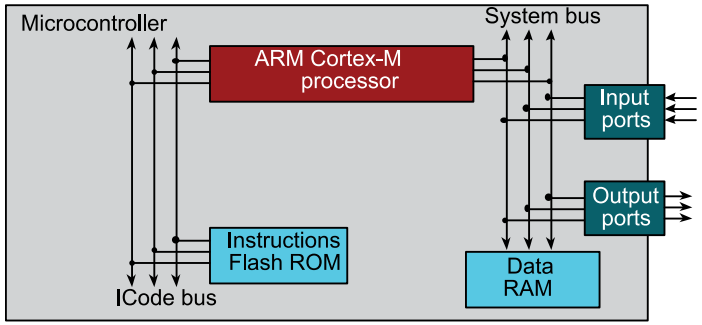

Figure 1. Architecture of an ARM Cortex M microcontroller.

This module serves as a brief introduction to the Cortex M microcontroller. Even though we typically program embedded systems in C, it makes sense to understand a little bit how the microcontroller executes software. Understanding some of these low level details will make it easier to make high level software design decisions. Examples where knowing low-level details make for better high-level decisions include: local verses global variables, numbers verses pointers, numerical overflow during calculations, numerical dropout during divide and right shift operations, integer versus floating point calculations, and interrupts.

There are two reasons we must learn the assembly language of the computer with which we are using. Sometimes, but not often, we wish to optimize our application for maximum execution speed or minimum memory size, and thus writing pieces of our code in assembly language is one approach to such optimizations. The most important reason, however, is that by observing the assembly code generated by the compiler for our C code we can truly understand what our software is doing. Based on this understanding, we can evaluate, debug, and optimize our system. So the goal of this module is not for you to become proficient in assembly language, but rather to learn enough so you can interpret the assembly code generated by the C compiler.

An **assembler** is system software that converts low-level assembly language program (human readable format) into object code (machine readable format). Typically, one line of assembly language creates one machine instruction, and this translation is simple and obvious. Writing in assembly exposes the low-level details of the architecture.

A **linker** builds a single software system by connecting (linking) software components. In CCS, the **build** command performs both assembly and linking. In an embedded system, the **loader** will program object code into flash ROM. We place object code in ROM because ROM retains its information if power is removed and restored. In CCS, the **Debug** command performs a load operation and starts the debugger.

A **debugger** is a set of hardware and software tools we use to verify system is operating correctly. The two important aspects of a good debugger are control and observability.

In the lab associated with this module, you will develop and test an assembly function typical of one the robot might use to perform a numerical calculation. In particular, the function will convert ADC measurements from a sensor into distance to the wall. In developing and debugging this function, you will gather important insights on how the Cortex-M processor executes software.

ti.com/rslk

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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