Computer Engineering 2DX3

2024-2025 Laboratory Manual

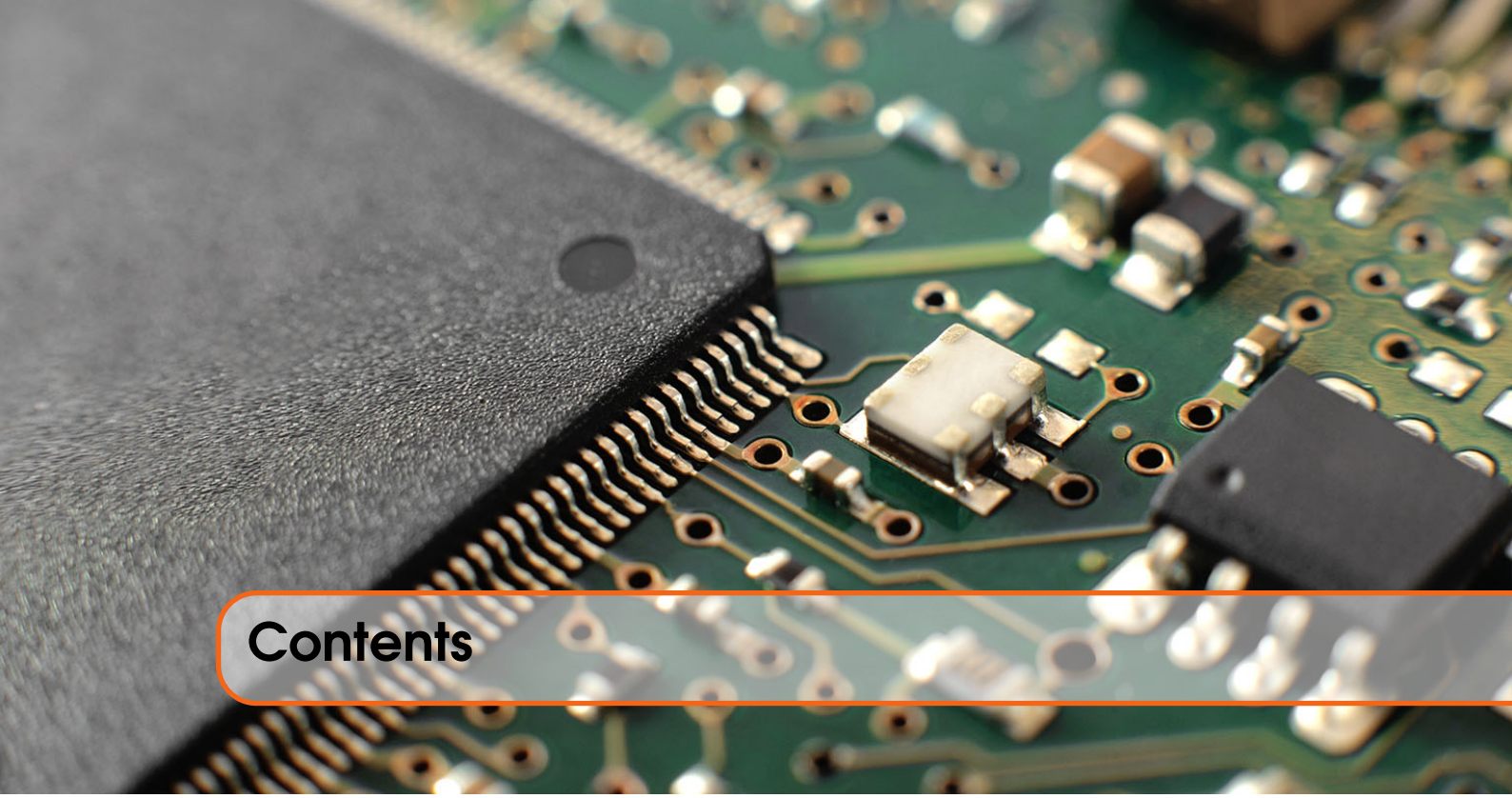
Last Update: January 29, 2025

Dr. Shahrukh Athar, Dr. Thomas E. Doyle, Dr. Yaser Haddara

. . . the designer of a new system must not only be the implementor and the first large-scale user; the designer should also write the first user manual. . . If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important. Donald E. Knuth

PUBLISHED BY DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, MCMASTER UNIVERSITY

January 2025



Contents

I Part One: Overview

1 Overview 11

1.1 Philosophy 11

1.2 Using this Lab Manual 12

1.3 Evaluation 12

1.4 Lab Submission Format/Requirements 13

1.4.1 Cover Page 13

1.4.2 Team Submission 13

1.4.3 Pre-Lab Report 13

1.4.4 Milestone Demonstrations 13

1.4.5 Lab Summary Report 13

1.4.6 Theme Report 14

1.5 Lab Delivery Format: In-Person 14

1.6 Lab Team/Partners 15

1.7 Bonus Mark Opportunities 15

1.8 Penalties 15

1.9 Missed Labs 16

1.10 Technical and Inclusive Language 16

II Part Two: Introduction to 2DX

2 Lab 0: Introduction to 2DX Labs 19
2.1 Objective 19
2.1.1 Resources 19
2.2 Pre-Laboratory Preparation 20
2.3 Integrated Circuits & Components 20
2.4 Laboratory Exercises and Milestones 20
2.4.1 Lab Safety Quiz 20
2.4.2 Form Teams 21
2.4.3 Moving from Manual to Automata 21
2.4.4 Microcontroller Bit "Switching" 22
2.4.5 Troubleshooting 25
2.5 Summary of Milestones & Evaluation Rubric 25
2.6 Submission Requirements 25
2.7 Grading Summary Table 26

III Part Three: Observe

3 Lab 1: Digital Signals 29
3.1 Objective 29
3.1.1 Resources 29
3.2 Pre-Laboratory Preparation 29
3.3 Integrated Circuits & Components 30
3.4 Laboratory Exercises and Milestones 30
3.4.1 Hello World Assembly 30
3.4.2 Pushing your button(s) Assembly 31
3.5 Summary of Milestones & Evaluation Rubric 32
3.6 Submission Requirements 32
3.7 Grading Summary Table 33

4 Lab 2: Finite State Machine and Digital Design 35
4.1 Objective 35
4.1.1 Resources 35
4.2 Finite State Machine 35
4.3 Pre-Laboratory Preparation 36
4.4 Integrated Circuits & Components 36
4.5 Laboratory Exercises and Milestones 36
4.5.1 Parallel I/O – Combinational Lock 36
4.5.2 Sequential I/O – Sequential Lock 37
4.5.3 BONUS: Troubleshooting 37

4.6 Summary of Milestones & Evaluation Rubric **38**
4.7 Submission Requirements **38**
4.8 Grading Summary Table **38**

5 Lab 3: Analog Signals 39
5.1 Objective **39**
5.1.1 Resources 39
5.2 Pre-Laboratory Preparation **39**
5.3 Integrated Circuits & Components **40**
5.4 Laboratory Exercises and Milestones **40**
5.4.1 Analog to digital conversion (ADC) of DC Signal 40
5.4.2 ADC of Sinusoidal Waveform 41
5.4.3 ADC of Square Waveform 42
5.4.4 BONUS: Frequency Counter 42
5.5 Summary of Milestones & Evaluation Rubric **42**
5.6 Submission Requirements **43**
5.7 Grading Summary Table **43**
5.8 Supplied Reference Code **44**

IV Part Four: Reason

6 Lab 4: Duty Cycle and Pulse Timing 49
6.1 Objective **49**
6.1.1 Resources 49
6.2 Pre-Laboratory Preparation **49**
6.3 Integrated Circuits & Components **50**
6.4 Laboratory Exercises and Milestones **50**
6.4.1 Variable Duty Cycle for PWM of an LED 51
6.4.2 Stepper Motor 51
6.4.3 BONUS: RGB LED 52
6.5 Summary of Milestones & Evaluation Rubric **52**
6.6 Submission Requirements **52**
6.7 Grading Summary Table **53**

7 Lab 5: Peripheral Interfacing 55
7.1 Objective **55**
7.1.1 Resources 55
7.2 Pre-Laboratory Preparation **55**
7.3 Integrated Circuits & Components **56**
7.4 Using the Microcontroller's Internal Pull-up Resistors **56**
7.5 Laboratory Exercises and Milestones **56**
7.5.1 4x4 Keypad Button Identification 56
7.5.2 Key Decode 57

7.5.3 LED Display . . . 57
7.5.4 BONUS: 7-Segment Display . . . 58
7.6 Summary of Milestones & Evaluation Rubric 58
7.7 Submission Requirements 58
7.8 Grading Summary Table 59

8 Lab 6: Project Deliverable 1 Early Integration . . . 61
8.1 Objective 61
8.2 Instructions for Deliverable 1 61
8.2.1 Logistics . . . 61
8.2.2 Demo . . . 62
8.2.3 Interview . . . 64

V Part Five: Act

9 Lab 7: Interrupts and Event Programming . . . 67
9.1 Objective 67
9.1.1 Resources . . . 67
9.2 Pre-Laboratory Preparation 67
9.3 Integrated Circuits & Components 67
9.4 Laboratory Exercises and Milestones 67
9.4.1 Periodic Interrupt . . . 68
9.4.2 GPIO Interrupt . . . 68
9.4.3 BONUS: Interrupt Buttons for transmission to PC . . . 69
9.5 Summary of Milestones & Evaluation Rubric 69
9.6 Submission Requirements 70
9.7 Grading Summary Table 70

10 Lab 8: Collecting Distance Data . . . 71
10.1 Objective 71
10.1.1 Resources . . . 71
10.2 Pre-Laboratory Preparation 71
10.3 Integrated Circuits & Components 72
10.4 Laboratory Exercises and Milestones 72
10.4.1 Read Time-of-Flight Sensor ID and Module Type . . . 72
10.4.2 Time-of-Flight Sensor Measurement via I2C . . . 73
10.4.3 BONUS 1: Time-of-Flight Sensor Measurement to Python . . . 74
10.4.4 BONUS 2: Visualize Time-of-Flight Measurement Data . . . 74
10.5 Summary of Milestones & Evaluation Rubric 74
10.6 Submission Requirements 75
10.7 Grading Summary Table 75

VI Appendix A

11 Getting to Hello 79
11.1 Getting Started... 79
11.2 macOS – Virtualization 80
11.3 macOS – Bootcamp 80
11.4 Windows 81
11.4.1 Keil MDK 81
11.4.2 USB Debug Firmware 81
11.4.3 Configure MDK Flash Tools 84
11.5 Say Hello 85
11.6 Software Version Summary 86

VII Appendix B

12 Creating A New Project 89
12.1 Create a New Project from Scratch 89
12.1.1 Assembly Language 89
12.1.2 C Language 89

VIII Appendix C

13 Keil Debugging 93
13.1 Video Instruction 93
13.2 Exporting Data From the Keil IDE 93
13.3 Memory Value Capture Function 95

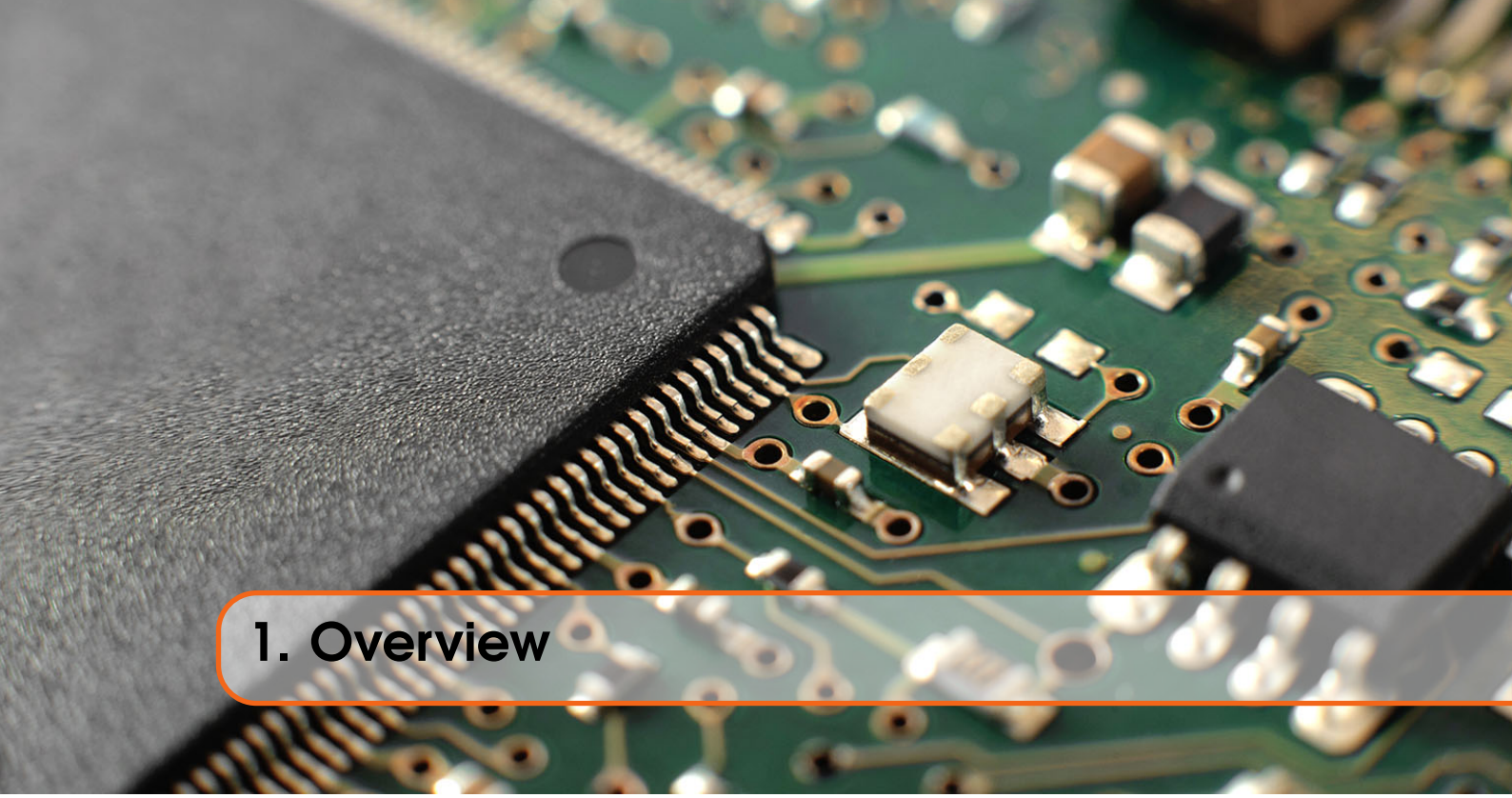
IX Appendix D

14 Change Log 99

Part One: Overview

1 **Overview** 11

1.1 Philosophy
1.2 Using this Lab Manual
1.3 Evaluation
1.4 Lab Submission Format/Requirements
1.5 Lab Delivery Format: In-Person
1.6 Lab Team/Partners
1.7 Bonus Mark Opportunities
1.8 Penalties
1.9 Missed Labs
1.10 Technical and Inclusive Language



1. Overview

1.1 Philosophy

Intelligent devices are ubiquitous and advancing computationally at an accelerated rate. Traditional courses in Microprocessors have approached the topic from the computer architecture perspective and then as an appendix included the characteristics that permit computing to be “intelligent".

2DX has been designed as a project based course where lecture theory is connected with hands on experience in studio and integrated in weekly labs. Each week constitutes a knowledge thread that combines the themes of Observe, Reason, and Act.

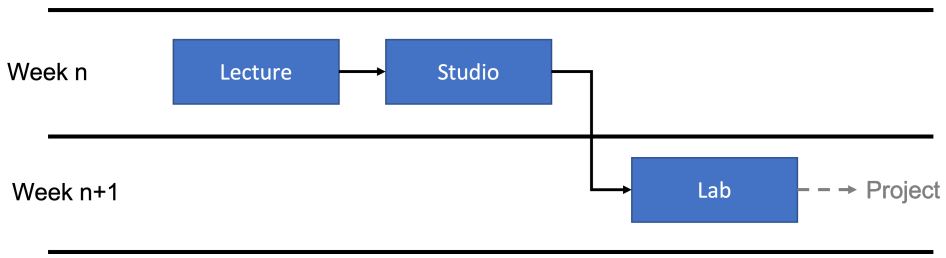


Figure 1.1: 2DX Knowledge Thread

Figure 1.1 illustrates the concept of a knowledge thread in this course. Lectures are intended to provide context and studios are hands-on lessons. At the start of the course the lab exercises follow the related lecture and studio by one week. As the course progresses there can be more than one week between the lab and the related lecture and studio. The knowledge to complete a lab is always intended to be provided in the prior week(s), thus labs are expected to be started before attending the scheduled lab section. Scheduled lab sections are intended for trouble shooting and milestone assessment.

Each lab is also intended to provide a piece of the solution for the final project. The structure of this course is such that you must start working on the project before you have all of the pieces of the puzzle. Your instructors will help guide you in the early stages, but it is imperative that you start early.

1.2 Using this Lab Manual

This lab manual was created using the typeset document preparation system called LaTeX. This file contains embedded and linked media content. The embedded content is designed to run inside the Adobe Acrobat Reader. Please download a copy of the manual and open with Adobe Reader on your local computer. This manual will see continuing updates throughout the term, so for this reason we do not recommend that you print this manual. The Adobe reader is free and can be downloaded here: `https://get.adobe.com/reader/`.

Throughout this manual highlighted remarks will be identified using this notation.

Milestone 1.1 Lab exercises will have milestones that are to be demonstrated to a TA. Milestones are identified using this notation. ■

1.3 Evaluation

The final mark in the Microprocessor Systems Project course consists of the components listed in the course syllabus. The laboratory components and weights of the course mark are also defined in the course syllabus. The breakdown of marks within each lab, including bonus marks (see Section 1.7), is given in the individual lab chapters in this manual.

Labs run every week of the course. You are required to check the course website daily to confirm the schedule and receive corrections and/or clarifications to the lab exercises.

It should be stressed that due to high course enrollment, students shall not be admitted to other laboratory sections than those assigned. An automatic zero will be applied to students attending the incorrect section.

Please enter and exit the lab punctually and pay close attention to the time limits that your TAs give you for submitting your lab exercises. Failure to have your lab execution checked because the lab time expired may result in a 0 for the lab. Failure to exit the lab punctually may result in being assigned an automatic zero for the entire lab exercise. TAs are instructed not to accept/review work that is late.

1.4 Lab Submission Format/Requirements

1.4.1 Cover Page

Each submission must have a cover page clearly indicating the lab, title, date, student name(s), and student number(s). In addition, the cover page must have the following statement:

> As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is our own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario.

1.4.2 Team Submission

A team (See Section 1.6) must remain consistent for all the deliverables of a given lab. What this means is that you cannot submit a pre-lab for a given lab with one partner but submit the lab summary report with a different partner. However, if one of the team members does not participate in part of the work, the other team member may submit that part individually. In addition, teams are to remain consistent through the entire term.

R Only one member of the team shall submit the work. Both members of the team must have their own copy of work for future reference.

R If you cannot answer TA questions about the lab or your information is not on the cover page then you will not receive credit for the work.

R All submitted work needs to be clear and concise. Work that is not legible, unclear, or is too verbose will be penalized accordingly.

1.4.3 Pre-Lab Report

In addition to the cover page, include the text of the original question followed by your clear and concise answer. Answers to design or calculation questions will receive a 0 if they are not accompanied by supporting work.

Team: Yes, see Section 1.6
Deadline: **Sunday at 11:59pm prior to the start of the lab week for all sections**.

1.4.4 Milestone Demonstrations

Milestones are demonstrated for marks. To receive the marks each team member should be able to execute, explain, and answer questions about demonstration.

Team: Yes, see Section 1.6
Deadline: Before the end of the lab.

1.4.5 Lab Summary Report

The lab summary report should have the following sections:

1. Purpose: state the purpose of the lab

2. Background: provide 1-2 paragraphs providing context for the lab
3. Method: provide a concise outline of solution(s) (e.g., code presented as flowchart)
4. Results: record the validation results
5. Observations and Conclusions: statements of outcome(s) that is normally tied back to purpose
6. References: IEEE format

Team: Yes, see Section 1.6
Deadline: The summary report is for your own records. Note carefully, your Theme Reports will require inclusion of exemplars of lab milestone execution, validation, and debugging from your own lab work. You will require documented records of this lab work to include in the Theme reports, otherwise you will need to re-run the experiments to gather the necessary data.

1.4.6 Theme Report

This course is based around three themes under the context of intelligent systems: Observe, Reason, and Act. The Observe theme includes lab exercises 1-3, the Reason theme includes lab exercises 4-6, and the Act theme includes lab exercises 7-8.

The purpose of each report is for the synthesis and reflection of the related theme. Your report must have the following sections:

1. Theme: state the theme of the report and define its meaning in the context of intelligent systems.
2. Background: provide 2-3 paragraphs providing context for the theme and how it ties into intelligent systems.
3. Theme Exemplar(s): select 1-2 excellent examples from *your own lab work* related to the theme and present their method and validation results. For example, select exemplar(s) from one of the three labs you completed under the Observe theme. If you have documented these adequately then you should only need to select examples from your lab records. If you have not adequately documented then you will need to repeat portions of the labs on your own to have an exemplar.
4. Debugging Exemplar: describe and demonstrate a relevant debugging method you used during this theme's labs. The method of debugging must be different for each theme report.
5. Synthesis: explain how your selected theme exemplar(s) illustrate the theme in the context of intelligent systems.
6. Reflection: concisely describe how the exemplar(s) contributed to *your* understanding of a) this theme, and b) intelligent systems and/or microprocessors
7. References: IEEE format

Team: No, submitted individually
Deadline: Announced on Avenue to Learn. Submit to Avenue Dropbox.

1.5 Lab Delivery Format: In-Person

In-person labs will be run in the designated room(s) on campus. As you complete the milestones during your scheduled lab section you will need to request a TA to evaluate and record your work.

At the beginning of each lab the TAs will provide a brief overview of milestone expectations and objectives.

Please be mindful that your TAs will need to visit multiple student groups. It is important to come prepared and have your questions and/or demonstrations ready.

Remember that the lab work builds directly on the studio exercises from the prior week(s). Participating in Studio is fundamental to your learning experience in this course because you obtain hands-on experience with the material. You are expected to start your lab before the scheduled lab section. The lab section is intended for getting help and debugging and finalizing your work.

Do not forget to upload your final work to Avenue by the deadline. Check that you have uploaded the correct file(s). We cannot accept late files.

1.6 Lab Team/Partners

Students will be permitted to work in **teams of two members for labs**. The project remains an individual component of the course. Teams must be from the same scheduled lab. If you need help forming a team please contact your lab TA(s) and ask for assistance. Students are cautioned against the *divide-and-conquer* approach because both team members should expect to be examined on all lab submission/milestone content – if you cannot answer the question(s) then you cannot be awarded the marks. You also require full knowledge of the labs to complete the individual final project.

1.7 Bonus Mark Opportunities

Laboratory exercises may offer bonus mark(s) for extra and/or advanced work by the student. **The bonus mark(s) will only apply to lab submissions that make a clear attempt to meet all non-bonus criteria (including pre-lab, if required)**. For example, if a core milestone in a given lab has not been attempted, the bonus deliverable(s) will not be considered in the assessment of that lab. It should also be noted that, although the bonuses may make it possible to achieve greater than 100% in the lab component of 2DX, the maximum grade assigned to the lab component will not be greater than 110%.

When submitting work for a bonus milestone you may be contacted later by the TA or instructor for follow up questions.

1.8 Penalties

Any submission within 10-minutes late will be assigned a 0% penalty. Any submission more than 10-minutes late will be assigned a 100% penalty. The Avenue to Learn time stamp is considered the official submission time.

10% penalty on entire lab grade if cover page is missing or incorrect (not according to lab manual)

10% penalty on entire lab grade if images are hand-drawn or submission is handwritten.

Double check your submitted files because we cannot accept any files late.

1.9 Missed Labs

If you miss a lab then you should initiate the MSAF process. The 2DX lab policy for *SELF-REPORT (TYPE A)*[1] MSAF is that the lab weight does not go to final. Instead, it will increase the relative weight of all other regular labs. Note that for team situations the MSAF applies only to the individual student. Instructions on submitting MSAFs will be posted on Avenue to Learn. MSAFs submitted to the wrong email will not be processed.

There are no makeup labs. If you know in advance of an absence, your instructor may be able to arrange alternate lab time, but only if coordinated prior to scheduled lab and space is available.

The labs in this course are essential building blocks towards the final project. If you have to miss a lab, the MSAF process helps you reallocate the marks for that lab. But you will still need to do the lab exercises and understand them well in order to be able to complete your project. If you find yourself falling behind, please approach your instructors for help with the course material.

1.10 Technical and Inclusive Language

Words matter. In the technical domain, the terms *master* and *slave* have long been used to describe *primary* and *secondary* objects in a system, respectively. These terms have obvious negative connotations and apart from their use as descriptors of legacy systems there is recognition that they should be deprecated and no longer employed. The IEEE Standards Association (IEEE SA) agrees that IEEE standards should be written in such a way as to avoid non-inclusive and insensitive terminology. As of December 2021, IEEE P1588g is developing a consensus on alternative terminology [2]. Final recommendations are expected to be delivered to the IEEE Standards Review Committee in December 2022. At the time of publishing this manual in January 2023, the Review Committee has not published their update.

Until a standards body, like the IEEE SA, formally adopts new language, there are a number of possible alternatives. While we cannot modify existing technical documents (e.g., data sheets, user manuals, textbooks) to remove these legacy descriptions, we will endeavor to make changes to our own documents and descriptions to instead use *Controller/Peripheral* or *Leader/Follower*.

We are grateful to students who took the course in Winter 2021 for thoughtfully pressing this issue. We encourage you to be engaged with all facets of your learning, including ethical, professional, and social implications not only technical aspects. Thoughtful, constructive engagement enriches our community and improves the learning environment for all students.

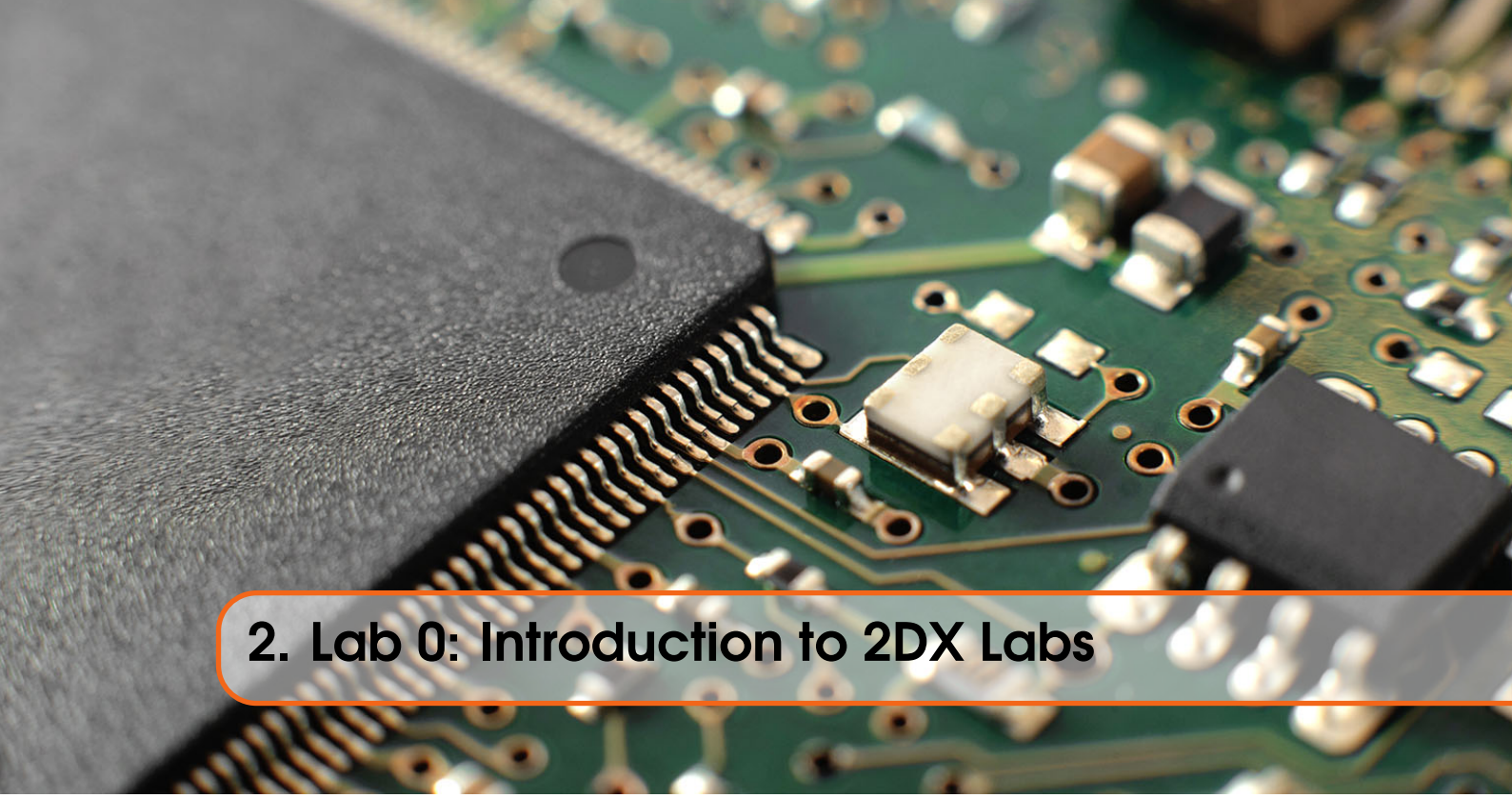
[1] `https://secretariat.mcmaster.ca/app/uploads/Requests-for-Relief-for-Missed-Academic-Term-Work-Policy-on.pdf`

[2] For additional reading, see `https://en.wikipedia.org/wiki/Master/slave_(technology)`, `https://standards.ieee.org/project/1588g.html`, and `https://www.allaboutcircuits.com/news/how-master-slave-terminology-reexamined-in-electrical-engineering/`.

Part Two: Introduction to 2DX

2 Lab 0: Introduction to 2DX Labs 19

2.1 Objective
2.2 Pre-Laboratory Preparation
2.3 Integrated Circuits & Components
2.4 Laboratory Exercises and Milestones
2.5 Summary of Milestones & Evaluation Rubric
2.6 Submission Requirements
2.7 Grading Summary Table



2. Lab 0: Introduction to 2DX Labs

2.1 Objective

To introduce students to the 2DX microcontroller (TI MSP432E401Y), the development environment (Keil), form lab teams of two, and perform the equivalent of Hello World.

R Student to note for Lab 0 – For this lab, there is no prelab. All future labs will require a prelab to be submitted for the team **prior to the lab cycle starting**.

2.1.1 Resources

You should review the following resources to prepare yourself for this lab and the future labs in this course:

1. Text:“Embedded Systems: Introduction to ARM Cortex-M Microcontrollers", 5th edition, 6th printing, January 2019, by Valvano – Chapters 1-3
2. Technical: MSP432E401Y - Microcontroller Data Sheet
3. Technical: MSP432E4 SimpleLink Microcontrollers - Technical Reference Manual
4. Technical: MSP432E401Y - User’s Guide
5. Technical: Cortex - M3/M4F Instruction Set Technical User’s Manual
6. Technical: Cortex-M4 Technical Reference Manual

R Refer to chapter 11 to install the IDE(MDK) on your own computer.

R Each lab exercise will begin by outlining relevant reference documentation and reading for the successful completion of your work. Technical data is available to you on Avenue to Learn and should generally be used as reference materials.

2.2 Pre-Laboratory Preparation

[0 marks total]

All the following laboratories will require a prelab to be submitted at the beginning of the lab week. Prelab will not be accepted if it is late.

R While your textbook is a valuable resource, the manufacturer of the device(s) always provide official technical references and normally these are free. Similar to books you buy on software, the textbook is a guide written *from* these technical references. The textbook can never cover the same depth as the original references and you will certainly see this with microcontrollers. It is imperative you become familiar with the technical manuals from the beginning of this course.

R Pre-lab work is designed to ensure you come prepared to lab and also to help you complete the lab. This is to be done with your lab partner. If you do not have a lab partner or your partner is away please note you are still responsible to complete this work. Ensure both partners have a copy of the lab and pre-lab after submission.

2.3 Integrated Circuits & Components

The following integrated circuits are to be used in this laboratory:

Table 2.1: Integrated Circuits and Components for Lab 0

Identification	Description
ARM Cortex-M4F	TI MSP432E401Y Microcontroller

Obtain the data sheets for each of the above digital devices. Familiarize yourself with their logical and electrical characteristics and bring a copy to your lab session.

2.4 Laboratory Exercises and Milestones

Read the following experiments and study the circuits as shown. Pre-filling your summary with the necessary truth tables and structuring your work such that you only need to record experimental output will allow you to focus on the milestones.

Every lab will require you to demonstrate specific milestones to your TA. If you do not demonstrate these milestones before the conclusion of your scheduled lab then you will not receive grades for the milestone. To receive the milestone marks, it is your responsibility to fully demonstrate, be able to explain, and answer questions about each milestone.

R **ENSURE YOUR MILESTONES ARE VISUALLY CHECKED AND RECORDED BY A TA – THERE ARE NO MARKS AFTERWARDS.**

2.4.1 Lab Safety Quiz

As you cannot perform the labs without completing the lab safety quiz, please successfully complete the quiz now as your first milestone. To prepare for the quiz, you can go through the *ECE Lab Safety* document located on Avenue in the *Lab Resources* section.

Milestone 2.1 To participate in lab you must complete the lab safety quiz. To take the quiz, go to Avenue > Assessments > Quizzes > ECE Lab Safety Quiz. Complete the lab safety quiz and show to TA.

0 marks ■

2.4.2 Form Teams

Either before or during this lab you will need to partner with one other student to form your lab team. Please see the lab TA if you do not have a partner and TAs will pair you. It is expected that you will continue in this team throughout the term.

Milestone 2.2 Form a team of two and inform your TA that you are partners.

0 marks ■

2.4.3 Moving from Manual to Automata

Combinational and sequential digital logic generally requires the operator to manually control the system using buttons and switches.

Manual control of small simple tasks is not unreasonable; however, repeated or complex tasks provides a need for intelligent automation. Modern embedded processors, like the ARM Cortex-M4F (the one you are using), use 32-bits for memory width, which can be thought of as 32 toggle switches (binary bits). Similar to the manual process you worked through within COMPENG 2DI4, would you want to be manually moving the 32 toggle switches every time you needed to control your digital system? Early embedded programmers did use similar switching[1]. Programming thankfully progressed from machine language (simply writing binary 1's and 0's), to mnemonics, to cross assemblers. To get started with our embedded programming we will install a development environment to help us program and debug our processor (note the lab computers already have the development environment installed). When beginning to work with a new development platform, it is imperative to confirm the environment works correctly before testing complex programs. Typically a programmer will attempt to assemble the equivalent of a "Hello World!" program. This is a simple program that is known to work that outputs a message to the programmer indicating a successful program has been executed. In the appendix of this manual, refer to the section "Getting to Hello" to establish your own development environment works.

Milestone 2.3 The lab computers already have a functional development environment installed (Keil). On the lab computer, assemble and run the "blinkLED" project. Recall the MDK /IDE output will look like Figure 11.5 when it is run successfully. Figure 11.6 shows the debug environment.

For this milestone, create your own project. To learn how to create your own project watch the YouTube video `https://www.youtube.com/watch?v=ZLzT955KfGY&feature=youtu.be`. You also did this in *Pre-Studio 0*. The project must be named `Lab0_X`, where `X` is your first name or the concatenation of your first names (if working in a group of 2). For this milestone you must create a new project using the `L0_ExampleIO` code (a modified `BlinkLED`) that will visually increase or decrease the time delay, thus slow or speed up the flash of the LED. Create a flowchart describing each function block of the new code. You can download `L0_ExampleIO.c`

[1] Refer to the IMSAI – https://en.wikipedia.org/wiki/IMSAI$_8$080

from Avenue under *Lab 0* in the *Lab Resources* folder. You need to demonstrate the blinking LEDs to your TAs who may ask you to increase or decrease the associated delay.

30 marks ■

R If your code compiles and loads without issue, but the LED doesn't flash, please check that your Keil IDE has set its compiler optimization to 0. Some optimization options will remove code that the compiler deems unnecessary - like an empty loop that we used for a time delay.

2.4.4 Microcontroller Bit "Switching"

As noted earlier, manually setting toggle switches would be tedious for controlling complex systems, such as controlling a data path and/or memory access. The MSP432E401Y Microcontroller offers extensive functionality and it does so by mapping external pins to multiple internal functions. From the MSP432E401Y Data Sheet, partially illustrated in Figure 2.1, we observe the numerous functions available per pin.

In this part of the lab exercise, instead of manually setting toggle switches, we will modify the provided code to set output lines from the MSP432E401Y Microcontroller board – specifically port M bits 0-7.

Milestone 2.4 In this milestone you will constinue using the Keil IDE and the supplied C code that you used in the previous milestone, i.e., `L0_ExampleIO.c` (see Listing 2.1). It should be noted that for initial familiarity we will use the C-language version of the "Hello World!" for this lab. Labs 1, 2 and 3 will use ARM assembly language.

Instead of manually setting toggle switches you will write code to output bit values to the eight General Purpose Input/Output (GPIO) lines of Port M, i.e., bits 0-7 or PM0-PM7. Wire these GPIOs to LEDs on the lab's digital board. Ensure you make note of the Most Significant bit (MSb) and Least Significant bit (LSb). Continuing with the work done in Milestone 2.3, you will see that in the `L0_ExampleIO.c` code (line 83), we have set Port M bits 0-7 to be all set to 1. Change the line of code that sets the "toggle switch" values (line 83) to a binary value of your choice, reassemble and re-load the code on the MSP432E401Y. You should see the same binary value on the eight LEDs on the digital board.

Your TA will ask you to modify line 83 and demonstrate the code outputs the value you programmed on to the LEDs.

40 marks ■

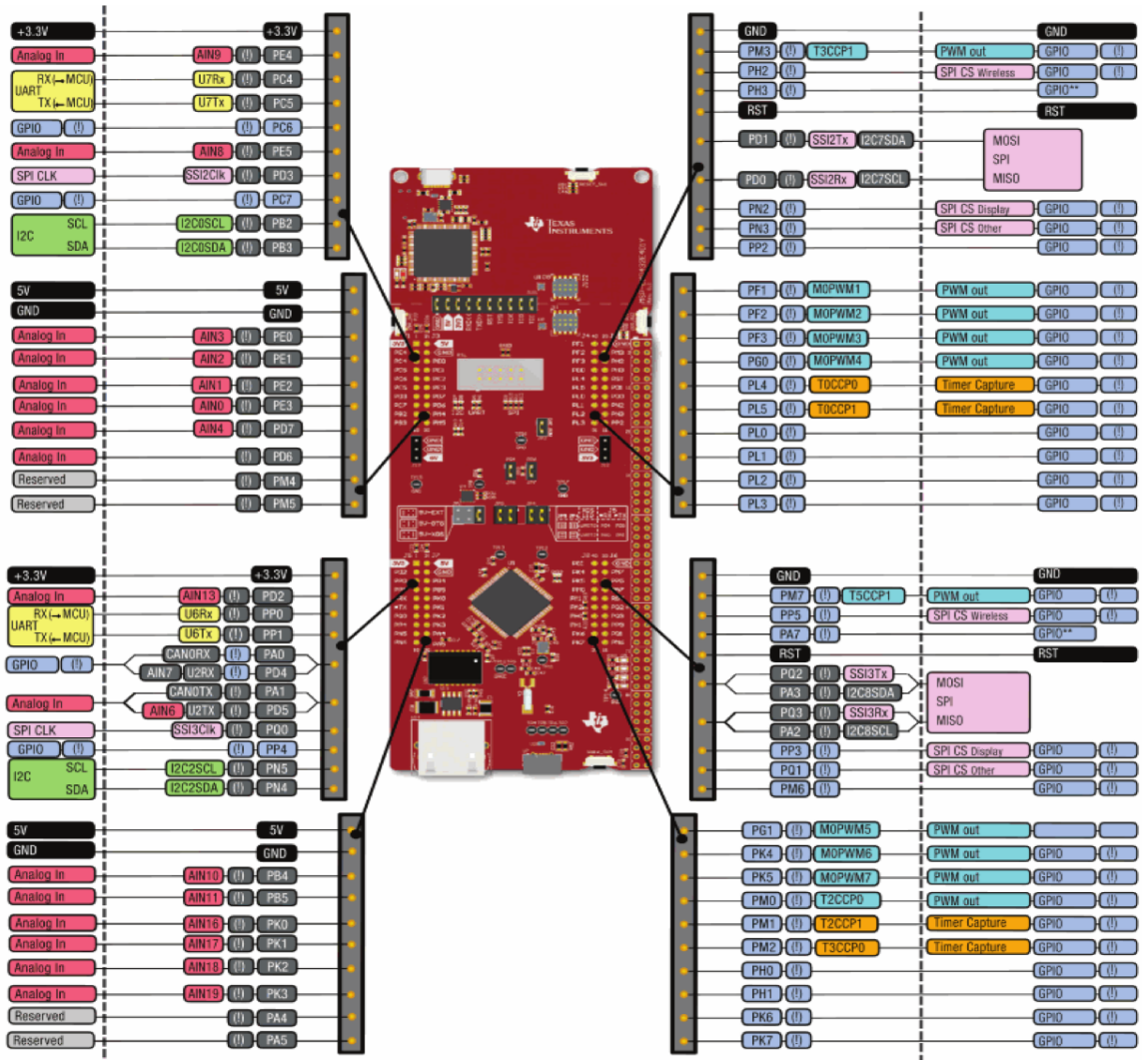


Figure 2.1: MSP432E401Y Headers and Available Pins

```
//0. Documentation
//*****************************************************************************
// Simple output C program based on the modification of the blinkLED.c code
// provided by TI.
//
// There are 7 steps to initialize a GPIO port for general use
//    i. Activate the clock for the port by setting the corresponding bit in
//         the RCGCGPIO register and then wait for status bit in PRGPIO
//         register to be true.
//   ii. Unlock port if using PD7.
//  iii. Disable analog function of the pin. (Upon reset default = disabled)
//   iv. Clear bits in PCTL to select regular digital function (see textbook
//         tables 4.1&4.2, noting upon reset the default function is digital).
//    v. Set the data direction register (i.e., input or output).  In the DIR
//         register, a bit set to: 0 = input, 1 = output.
//   vi. Clear bits in the alternate function register (remember pins / ports
//         can have alternate functionality when configured as such.
//  vii. Enable the digtal port
//
//  Note: Step i must be done first, but remaining steps may be performed in
//    any order.  If your intention is to use a GPIO pin in its default
//    digital state, then steps iii, iv, and vi can be omitted
```

```
//    (also ii if not using PD7).
//
//  TEDoyle
//  October 26, 2019

//*****************************************************************************
//1. Preprocessor Directives
//*****************************************************************************
// There are library files that define the peripherals, ports, and pins.
// Review the msp432e401y.h file to become familiar with the symbolic labels
// that have been preassigned to help simplify your programming. These labels
// can be different than your textbook, thus library familarity is important.

#include "msp.h"    // this libabry file chooses the approproate MSP library

//2. Functions
//*****************************************************************************

// For now, no functions outside of main().

//3. Main
//*****************************************************************************

int main(void)
{
  // step i.
  // Enable GPIO clocks for N & M peripheral
  SYSCTL->RCGCGPIO |= SYSCTL_RCGCGPIO_R12;  // Port N (onboard LED)
  SYSCTL->RCGCGPIO |= SYSCTL_RCGCGPIO_R11;  // Port M (GPIO for Lab 0 EPROM)

  // Check if the peripheral access is enabled.
  while(!(SYSCTL->PRGPIO & SYSCTL_PRGPIO_R12)); //N
  while(!(SYSCTL->PRGPIO & SYSCTL_PRGPIO_R11)); //M

  //skip steps ii, iii, iv

  // step v.
  /* Enable the GPIO pins for the onboard LEDs (PN1 = out, PN0 = out). */
  GPION->DIR = 0x03;

  /* Enable the GPIO pins for the EPROM (PM7:0 = out). */
  GPIOM->DIR = 0xFF;

  //skip steps vi

  // step vii.
  GPION->DEN = 0x03;
  GPIOM->DEN = 0xFF;

  // main code
  uint32_t counter = 0;

  GPION->DATA = 0x01;   //initialize the bit values

while(1)
  {
      // Toggle GPIO pin value
      GPION->DATA ^= BIT0;
      GPION->DATA ^= BIT1;
      for(counter = 0; counter < 1000000; counter++) {} //orig set to 200000
      GPIOM->DATA = 0xFF;                               //bits 0-7 of Port M
  }
}
```

Listing 2.1: C code example for Lab 0 - Download from Avenue under Lab 0

2.4.5 Troubleshooting

A key requirement for any embedded developer is the ability to troubleshoot your code. The Keil debug environment is a powerful tool in testing and troubleshooting code.

Milestone 2.5 Using the provided code listing (continuing your work from Milestone 2.4) demonstrate the following to your TA (all at same time):

1. Step through one line of your code.
2. Set a breakpoint immediately after your delay loop ends and demonstrate it works.
3. Using Keil, directly inspect the value of the bits associated with Port M.

30 marks ■

2.5 Summary of Milestones & Evaluation Rubric

TA must visually verify and record your successful completion of:

1. Milestone 1
2. Milestone 2
3. Milestone 3
4. Milestone 4
5. Milestone 5

Be aware, if the TA does not have a record of the execution then a mark of zero is assessed.

Table 2.2: Evaluation Rubric

Criteria	Mark
Student successfully demonstrates and explains to TA a fully correctly working milestone	100%
Student demonstrates and explains a coherent attempt at milestone, but incorrect result	50%
No demonstration, cannot explain, or non-coherent attempt/demonstration	0

2.6 Submission Requirements

Refer to section 1.3.

2.7 Grading Summary Table

Component	Weight	Grade
Pre-Lab Questions	0	
Milestone 1	0	
Milestone 2	0	
Milestone 3	30	
Milestone 4	40	
Milestone 5	30	
Total	**100**	
Deductions		
Final Score		

III Part Three: Observe

3 Lab 1: Digital Signals 29

3.1 Objective
3.2 Pre-Laboratory Preparation
3.3 Integrated Circuits & Components
3.4 Laboratory Exercises and Milestones
3.5 Summary of Milestones & Evaluation Rubric
3.6 Submission Requirements
3.7 Grading Summary Table

4 Lab 2: Finite State Machine and Digital Design 35

4.1 Objective
4.2 Finite State Machine
4.3 Pre-Laboratory Preparation
4.4 Integrated Circuits & Components
4.5 Laboratory Exercises and Milestones
4.6 Summary of Milestones & Evaluation Rubric
4.7 Submission Requirements
4.8 Grading Summary Table

5 Lab 3: Analog Signals 39

5.1 Objective
5.2 Pre-Laboratory Preparation
5.3 Integrated Circuits & Components
5.4 Laboratory Exercises and Milestones
5.5 Summary of Milestones & Evaluation Rubric
5.6 Submission Requirements
5.7 Grading Summary Table
5.8 Supplied Reference Code

3. Lab 1: Digital Signals

3.1 Objective

The objective of this lab is to gain a better understanding of the embedded hardware architecture and how to program it to detect and manipulate simple digital signals using assembly language.

3.1.1 Resources

You should review the following resources to prepare yourself for this lab and the future labs in this course:

1. Text: Digital Logic textbook on combinational and synchronous sequential logic design – see Design at the Register Transfer Level
2. Text:“Embedded Systems: Introduction to ARM Cortex-M Microcontrollers", 5th edition, 6th printing, January 2019, by Valvano – Chapters 1-4
3. Lab Manual: “Getting to Hello" Chapter 11
4. Technical: MSP432E401Y - Microcontroller Data Sheet
5. Technical: MSP432E4 SimpleLink Microcontrollers - Technical Reference Manual
6. Technical: MSP432E401Y - User’s Guide
7. Technical: Cortex - M3/M4F Instruction Set Technical User’s Manual
8. Technical: Cortex-M4 Technical Reference Manual

R Review how to create a new project in Keil in assembly (not C).

3.2 Pre-Laboratory Preparation

[20 marks total]

1. Record the exact name of the reference documentation for:
   (a) the core processing unit and its operation codes/language for the MSP432E401Y. [2]
   (b) the microprocessor logic systems and peripherals for the MSP432E401Y. [2]
2. Briefly explain the relationship between *machine language*, *op code*, and mnemonic with an example. [6]

3. In relation to the MSP432E401Y board, what core processor is used and define its registers (purpose and number of bits). [4]
4. Create a flow chart showing the steps to configure GPIO port M on the MSP432E401Y. For each step of configuring a GPIO port, define the relevant register's purpose. [6 marks]

R While your textbook is a valuable resource, the manufacturer of the device(s) always provides official technical references and normally these are free. Similar to books you buy on software, the textbook is a guide written *from* these technical references. The textbook can never cover the same depth as the original references and you will certainly see this with microcontrollers. It is imperative you become familiar with the technical manuals from the beginning of this course.

R Pre-lab work is designed to ensure you come prepared to lab and also to help you complete the lab. This is to be done with your lab partner. If you do not have a lab partner or your partner is away, please note you are still responsible to complete this work. Ensure both partners have a copy of the lab and pre-lab after submission.

3.3 Integrated Circuits & Components

The following integrated circuits are to be used in this laboratory:

Table 3.1: Integrated Circuits and Components for Lab 1

Identification	Description
ARM Cortex-M4F	TI MSP432E401Y Microcontroller
Normally open push button	generic
LED	onboard

Obtain the data sheets for each of the above digital devices. Familiarize yourself with their logical and electrical characteristics and bring a copy to your lab session.

3.4 Laboratory Exercises and Milestones

Read the following experiment and study the circuits as shown. Pre-filling your summary with the necessary truth tables and structuring your submission such that you only need to record experimental output will allow you to focus on the milestones.

R **ENSURE YOUR MILESTONES ARE VISUALLY CHECKED AND RECORDED BY A TA – THERE ARE NO MARKS AFTERWARDS.**

3.4.1 Hello World Assembly

Regardless of the language (C or assembler) the steps to configuring and programming a general purpose input and output (GPIO) port remain the same. Review steps to programming a port and review the assembly code from prior Studio(s).

Milestone 3.1 Create a new project, this time for assembly code, and write a program to blink one of the onboard LEDs from Port N. To do this you will need to look up the relevant Port N addresses.

40 marks ■

3.4.2 Pushing your button(s) Assembly

The momentary switch appears deceptively simple. While schematically shown with only two connections, they physically have 4 pins, as shown in figure 3.1. The connection of these pins is device dependent – never assume the connections! The datasheet should be consulted even for something as simple as a momentary switch, or a quick set of tests done to see (at the very least) if the device is “normally open" (n.o.) or “normally closed" (n.c.) when not being pressed.

Figure 3.1: Image of a generic momentary switch:

The next question is how to connect a momentary switch? With our own equipment we do not have the benefit of a pre-wired board. Figure 3.2 illustrates four methods to connect such a switch, but unfortunately only one will work correctly. The first time connecting a momentary switch, most will wire it as shown in figure 3.2-(c) and this might appear to work; however with a normally open momentary switch, when the button is not pressed the input to the microcontroller is not connected to any voltage input (referred to as *floating*). When an input to a microcontroller floats it cannot be assumed to be any value. The proper wiring requires the input to be *pulled-up* or *pulled-down* when the button is not being pressed. Figure 3.2-(b) properly implements a “pull-up resistor" whereby the input to the microntroller is pulled to a logic high when the button is NOT pressed and then forces the input to the micro to logic low when the button is pressed. Ensure you understand this arrangement as it may be counter-intutive: button unpressed = 1, button pressed = 0.

Now that we can program a GPIO pin as an output to turn an LED on (logic high) and off (logic low), the next step is to extend our code to capture a GPIO pin’s input.

Milestone 3.2 Return to reviewing the configuration steps of the GPIO, but in this case, use Port M bit 0 to capture if a push button is open (not pressed) or closed (pressed). When the button is pressed, turn on the designated LED. When the button is not pressed, turn off the designated LED. This milestone is to be done in assembly code.

If your student number’s least significant digit is:

1. 0, 4, or 8 then use LED 2

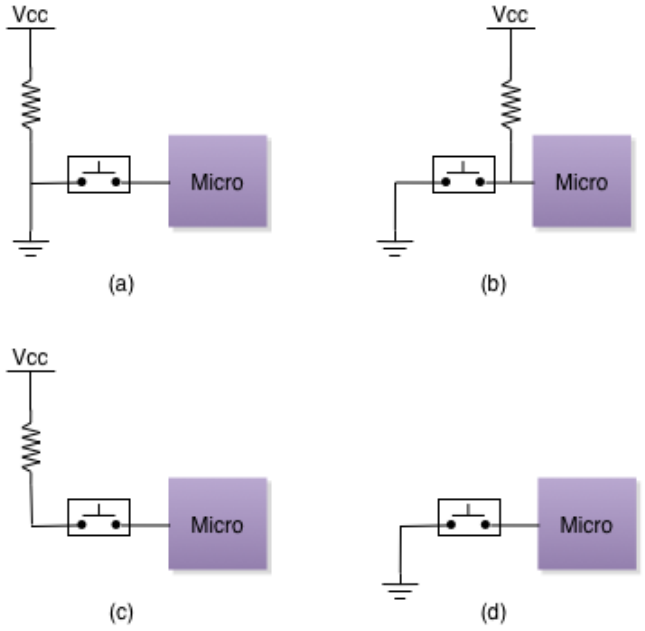


Figure 3.2: Connecting a momentary switch: (a), (c) and (d) incorrectly wired momentary switch input to a microcontroller, (b) a correctly wired pull-up resistor for momentary switch input to a microcontroller

2. 1, 5, or 9 then use LED 3
3. 2, or 6 then used LED 4
4. 3, or 7 then use LED 1

40 marks ■

3.5 Summary of Milestones & Evaluation Rubric

TA must visually verify and record your successful completion of:

1. Milestone 1
2. Milestone 2

Be aware, if the TA does not have a record of the execution then a mark of zero is assessed.

Table 3.2: Evaluation Rubric

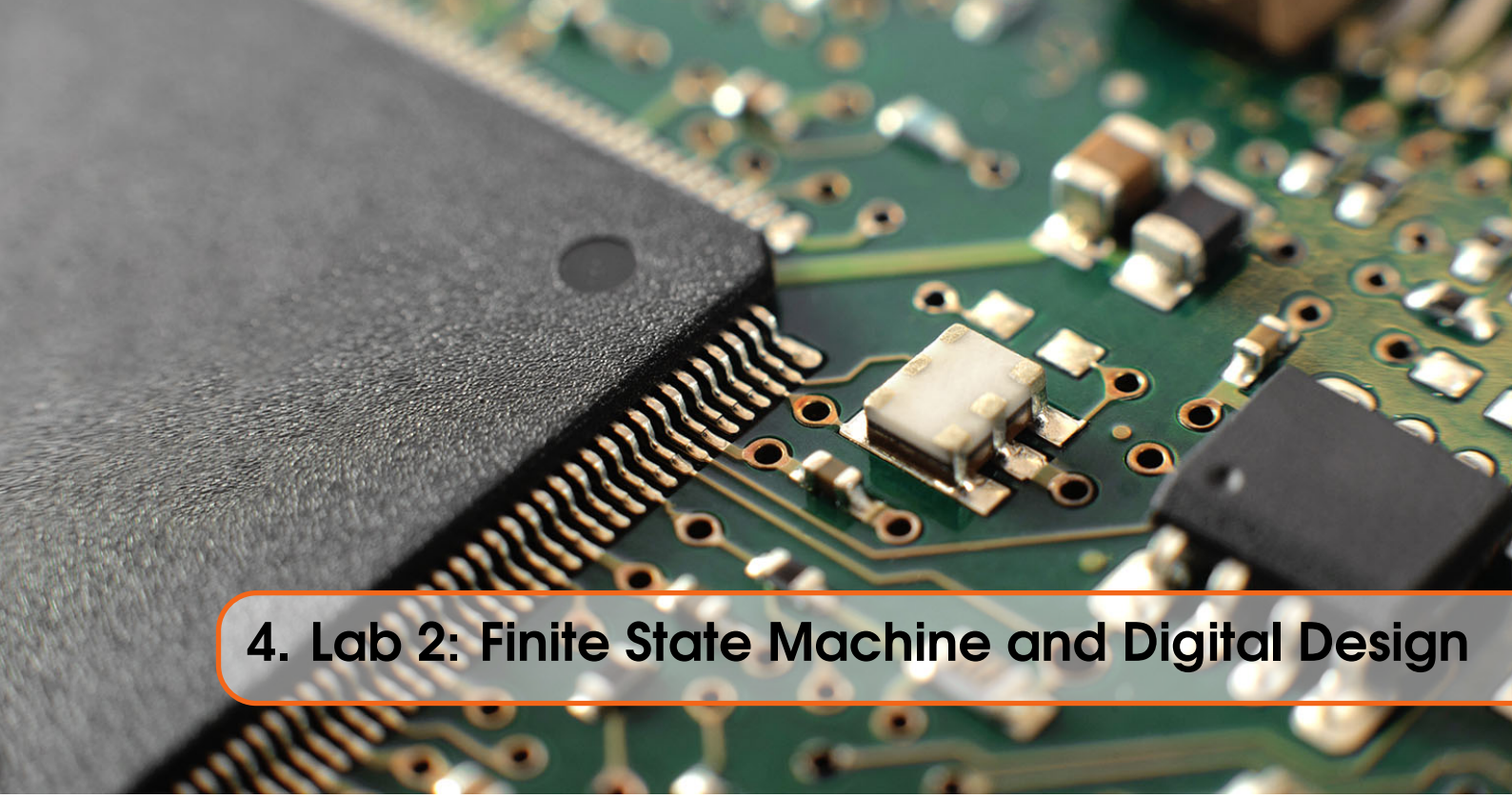
Criteria	Mark
Successfully demonstrate and explain to TA a fully correctly working milestone	100%
Demonstrate and explain a coherent attempt at milestone, but incorrect result	50%
No demonstration, cannot explain, or non-coherent attempt/demonstration	0

3.6 Submission Requirements

Refer to section 1.3.

3.7 Grading Summary Table

Component	Weight	Grade
Pre-Lab Questions	20	
Milestone 1	40	
Milestone 2	40	
Total	**100**	
Deductions		
Final Score		



4. Lab 2: Finite State Machine and Digital Design

4.1 Objective

The objective of this lab is to incorporate digital design methodology for the development of embedded systems. Using a Finite-State-Machine (FSM) approach, and extending our knowledge of GPIO in this lab you will gain a better understanding of the embedded hardware architecture and how to program it to detect and manipulate digital signals using assembly language.

4.1.1 Resources

You should review the following resources to prepare yourself for this lab and the future labs in this course:

1. Technical: MSP432E401Y - Microcontroller Data Sheet
2. Technical: MSP432E401Y - Technical Reference Manual
3. Technical: MSP-EXP432E401Y - Schematic
4. Technical: ARM Cortex M4 Processor Technical Reference Manual
5. Technical: ARM M Architecture Reference Manual
6. Technical: The Cortex-M4 Instruction Set

4.2 Finite State Machine

When designing complex systems, it is beneficial for the designer (and any one reviewing the design) to abstract away from the implementation details and to focus on describing the solution. Similar to how flowcharts and UML diagrams are used to describe procedural and object-oriented program design, the finite-state-machine (FSM) approach is useful for digital system design.

As your designs become more complex, the FSM approach becomes more powerful.

This lab is intended to introduce the FSM concept with your early design problems so that you may use it when the system design becomes more complex. While there are other resources, you can refer to your 2DI4 notes or your textbook section 6.5.2.

4.3 Pre-Laboratory Preparation

[20 marks total]

1. Starting with a truth table, design a 4-bit combinational digital lock that opens only when the parallel inputs are 1010. State the Boolean expression and draw the schematic for this combinational circuit. When the digital combinational lock is open, the output of the circuit should be a logic 1 to turn on an LED. The LED should remain off until the lock is opened. Flowcharts are useful to describe procedural steps in a simple system, such as this one. While not required for this question, consider how a flowchart would be written for your parallel digital lock. [10]

2. A better way to describe a more complex system is to use a finite state machine (FSM). Using the FSM approach, show the design of a 4-bit digital lock that opens only when the 1-bit serial sequential input is entered as 1,0,1,1. When the digital lock is open, the output of the circuit should be a logic 1 to turn on an LED. When the input serial sequence fails, the design should reset to the initial state. The LED should remain off until the lock is opened. This question is only asking for the FSM - do not design the sequential circuit. Do not worry about microcontroller settings for this question. For simplicity you may assume a Moore implementation. [10]

4.4 Integrated Circuits & Components

The following integrated circuits are to be used in this laboratory:

Table 4.1: Integrated Circuits and Components for Lab 2

Identification	Description
ARM Cortex-M4F	TI MSP432E401Y Microcontroller
Normally open push button	generic
LED	onboard

Obtain the data sheets for each of the above digital devices. Familiarize yourself with their logical and electrical characteristics and bring a copy to your lab session.

4.5 Laboratory Exercises and Milestones

Read the following experiments and study the circuits as shown. Pre-filling your summary with the necessary truth tables and structuring your submission such that you only need to record experimental output will allow you to focus on the milestones.

ENSURE YOUR MILESTONES ARE VISUALLY CHECKED AND RECORDED BY A TA – THERE ARE NO MARKS AFTERWARDS.

4.5.1 Parallel I/O – Combinational Lock

Generally an embedded system is handling much more functionality than a single button or switch. Review steps to configuring a port for input and output. For this section we will implement a combinational lock system with a data load functionality.

Milestone 4.1 Configure Port M for bits 0-2 (PM0, PM1, PM2) to be inputs. Configure your system as:

1. Connect PM[0-2] to three push buttons or toggle switches that represents a binary value input.
2. Connect PM3 to one push button or toggle switch that will be used to load the binary value for logic testing.
3. If your student number ends in an odd number, D2 ON represents a successful code load and D1 ON represents an unsuccessful code load. If your student number ends in an even number, D1 ON represents a successful code load, D2 ON represents a unsuccessful code load.

Your specific code is based on your student number. Observing the last 3 digits, inspect each digit. If a digit is even then the bit code at that position is 1. If a digit is odd then the bit code at that position is 0. For example, if the student number was 12346078, then the last 3 digits being 078, then the combinational code for your lock would be 101.

Write the assembly code to implement this parallel I/O system as a digital lock.
40 marks ■

4.5.2 Sequential I/O – Sequential Lock

You can likely reuse the port configuration from the prior milestone.

Milestone 4.2 Configure Port M for 2-bits to be inputs (hint: your configuration code should not need to change from previous milestone). Configure your system as:

1. Connect PM0 to one push button or toggle switch that represents a binary value input.
2. Connect PM2 to one push button or toggle switch that will be used to load the binary value for logic testing. This is the synchronizing clock that we would have used in 2DI4.
3. If your student number ends in an odd number, D2 ON represents a successful code load and D1 ON represents an unsuccessful code load. If your student number ends in an even number, D1 ON represents a successful code load, D2 ON represents a unsuccessful code load.

Your specific sequential code is based on your student number. Observing the last 4 digits, inspect each digit. If a digit is even then the bit code at that position is 1. If a digit is odd then the bit code at that position is 0. For example, if the student number was 12346078, then the last 4 digits being 6078, the sequential serial code for your lock would be 1,1,0,1.

Write the assembly code to implement this synchronous sequential system as a digital lock. Your design from the pre-lab can be updated and used to represent the FSM approach to designing this system. **40 marks** ■

4.5.3 BONUS: Troubleshooting

When debugging realtime systems it can be a challenge for IDEs to help troubleshoot because pausing an embedded program can modify the values that the programmer seeks to inspect (plus it's not realtime when it's paused). The benefit of LEDs for realtime debugging should never be underestimated.

Milestone 4.3 Return to your Sequential Lock FSM. Assign a unique binary code to each state in the FSM (2 or 3 bits depending on your design). Modify your code to output the present state binary code to LEDs, thus allowing you to see exactly which state your program is in during

operation. (using the 4 onboard LEDs, set 2-3 LEDs for state value and 1 LED for successful unlock) You may pick the onboard LEDs you use. **10 marks** ■

4.6 Summary of Milestones & Evaluation Rubric

TA must visually verify and record your successful completion of:

1. Milestone 1
2. Milestone 2
3. BONUS Milestone

Be aware, if the TA does not have a record of the execution then a mark of zero is assessed.

Table 4.2: Evaluation Rubric

Criteria	Mark
Successfully demonstrate and explain to TA a fully correctly working milestone	100%
Demonstrate and explain a coherent attempt at milestone, but incorrect result	50%
No demonstration, cannot explain, or non-coherent attempt/demonstration	0

4.7 Submission Requirements

Refer to section 1.3.

4.8 Grading Summary Table

Component	Weight	Grade
Pre-Lab Questions	20	
Milestone 1	40	
Milestone 2	40	
BONUS Milestone	10	
Total	**100**	
Deductions		
Final Score		

5. Lab 3: Analog Signals

5.1 Objective

The objective of this lab is to gain a better understanding of the embedded hardware architecture and how to program it to acquire analog signals using C language that is cross-assembled for the ARM.

5.1.1 Resources

You should review the following resources to prepare yourself for this lab and the future labs in this course:

1. Text:“Embedded Systems: Introduction to ARM Cortex-M Microcontrollers", 5th edition, 6th printing, January 2019, by Valvano – Chapter 10.1, 10.4, 10.5
2. Technical: MSP432E401Y - Microcontroller Data Sheet
3. Technical: MSP432E401Y - Technical Reference Manual (Refer to chapter 10)
4. Technical: MSP-EXP432E401Y - Schematic
5. Technical: ARM Cortex M4 Processor Technical Reference Manual
6. Technical: ARM M Architecture Reference Manual
7. Technical: The Cortex-M4 Instruction Set
8. Technical: Appendix D - Keil Debugging (included analog capture)

R Review how to create a new project in Keil in C. See end of the lab exercises for example C code. From this lab forward all programming will be done in C for assembly.

R The ADC is a complex system on the MSP432E401Y. You are provided code to get you started; however, to effectively use the code you will need to understand the ADC process and the concepts. Ensure you review your textbook, lecture, studio and support videos.

5.2 Pre-Laboratory Preparation

[30 marks total]

1. What method of analog to digital conversion does the MSP432E401Y ARM microcontroller use for its ADC? How many bits? [3 marks]
2. Converting an analog signal to digital will result in quantization error - show the calculations and explain the MSP432E401Y's maximum quantization error. [3 marks]
3. Draw a flowchart outlining the steps to configure the ADC on the MSP432E401Y. Hint: there are 13 steps. [13 marks]
4. Complete the table below assuming a 12-bit ADC (same as MSP432E401Y's ADC), assuming a full-scale voltage of 3.3V [11 marks]:

Analog Voltage (V_H)	x-bit ADC (hex)
0.00	
0.33	
0.66	
1.00	
1.33	
1.66	
2.00	
2.33	
2.66	
3.00	
3.30	

5.3 Integrated Circuits & Components

The following integrated circuits are to be used in this laboratory:

Table 5.1: Integrated Circuits and Components for Lab 1

Identification	Description
ARM Cortex-M4F	TI MSP432E401Y Microcontroller
Bonus: 7 LEDs	generic

Obtain the data sheets for each of the above devices. Familiarize yourself with their logical and electrical characteristics and bring a copy to your lab session.

5.4 Laboratory Exercises and Milestones

Read the following experiment and study the circuits as shown. Pre-filling your summary with the necessary truth tables and structuring your work such that you only need record experimental output will allow you to focus on the milestones.

ENSURE YOUR MILESTONES ARE VISUALLY CHECKED AND RECORDED BY A TA – THERE ARE NO MARKS AFTERWARDS.

5.4.1 Analog to digital conversion (ADC) of DC Signal

In this milestone you will need to select and configure one of the available analog pins to engage the ADC system. Review steps to programming port and the textbook section on the ADC.

Milestone 5.1 Connect the AD2 signal generator (use Wavegen to provide DC values) and complete the table with the binary values and the percent of maximum digital value. You will need to write your code to copy the converted digital value to a register for your inspection. Your TA will be asking how the ADC value is found.

For this milestone you must use and modify the memory capture method defined in Appendix C. Acquire 5-10 samples of each voltage and provide the average.

25 marks ■

Analog Voltage (V_H) DC	x-bit ADC (hex)	Percent of Max Value
0.00		
0.33		
0.66		
1.00		
1.33		
1.66		
2.00		
2.33		
2.66		
3.00		
3.30		

R In studio you were trying to measure a much smaller signal (maximum 10mV). Consider that such a small amplitude would only produce values from 0 to 12 of a maximum of 4095 using a 12-bit ADC. We used an op amp to provide voltage gain so that our 10mV would be amplified to the ADC's V_{RHP}, thus using the full 4095 ADC digital range.

For this lab, we are using the signal generator to set the signal amplitude to the ideal value instead of using an op amp.

5.4.2 ADC of Sinusoidal Waveform

A sinusoidal waveform is a periodic signal that varies with time. As discussed in lecture, taking a measurement of this waveform (referred to as *sampling*) requires the programmer to choose a rate by which to take measurements (the sampling rate). Sampling too slow results in aliasing. Sampling too fast can consume your computational resources, such as memory and cpu. Based upon the assigned sinusoid frequency, implement Shannon's sampling theory for the reproduction of the original sinusoid waveform's frequency.

R Pay very close attention to the waveform output of the signal generator (hint, it should be displayed and confirmed on scope before connecting to microcontroller).

Before proceeding, check the following:

1. Ensure the sinusoid signal is within the voltage range of the ADC (V_{RHN} V_{RHP}).
2. Ensure the signal generator output and the microcontroller power supply are connected to the same reference.

If you're not sure, please ask your TA.

Add the student numbers of the lab group together and observe the least significant digit for the assigned sinusoid frequency.

Least Significant Digit	0-3V Sinusoid Frequency (Hz)
0, 3, 6, 9	80
1, 4, 7	160
2, 5, 8	240

Milestone 5.2 Configure signal generator (AD2 Wavegen) to output a sinusoid 3Vpp (0-3V DC) at the assigned frequency. Configure ADC to sample the sinusoid without aliasing (reproduce frequency content). Clearly state your sampling rate and explain how this was achieved in your code.

In Studio we were able to use the functional debugging approach by setting up an IDE Watch where a global array was defined and it could be inspected during execution and post-execution. Our data of interest was small enough that we could transcribe by hand the values into an excel document for plotting. Given our sampling rate, and meaningful duration of time will mean hundreds of data points which will be impossible to transcribe during your lab time. Instead we will use Keil's Command Line tools to export our data to a file for analysis – see the pre-lab video for a demonstration of using the Keil Command Line for exporting data from our microcontroller's memory. (see manual Appendix C for assistance with Keil debug commands: LOG, EVal, D, FUNC, and creating .ini files)

Plot 500ms of captured waveform data.

35 marks ■

5.4.3 ADC of Square Waveform

Repeat the previous milestone by changing the sinusoid to a square wave.

Milestone 5.3 Configure signal generator (AD2 Wavegen) to output a square wave 3Vpp (0-3V DC) at the assigned frequency. Configure ADC to sample at the same frequency you used for the previous milestone. Explain the result.

Plot 500ms of captured waveform data.

10 marks ■

5.4.4 BONUS: Frequency Counter

What assumptions can you make about a sinusoid? What additional assumptions are there when a sinusoid is input to a microcontroller's ADC?

Milestone 5.4 Write a program to calculate the frequency of a waveform which is assumed to be non-aliased upto 100Hz. Your TA will ask you to modify the signal generator frequency to a value of 100Hz or less and your program must automatically measure and update the frequency count. Use the functional debugging approach we employed an IDE Watch in Studio where a global variable was be inspected during execution and post-execution.
BONUS 10 marks ■

5.5 Summary of Milestones & Evaluation Rubric

TA must visually verify and record your successful completion of:

1. Milestone 1
2. Milestone 2
3. Milestone 3
4. BONUS Milestone

Be aware, if the TA does not have a record of the execution then a mark of zero is assessed.

Table 5.2: Evaluation Rubric

Criteria	Mark
Successfully demonstrate and explain to TA a fully correctly working milestone	100%
Demonstrate and explain a coherent attempt at milestone, but incorrect result	50%
No demonstration, cannot explain, or non-coherent attempt/demonstration	0

5.6 Submission Requirements

Refer to section 1.3.

5.7 Grading Summary Table

Component	Weight	Grade
Pre-Lab Questions	30	
Milestone 1	25	
Milestone 2	35	
Milestone 3	10	
BONUS Milestone	10	
Total	**100**	
Deductions		
Final Score		

5.8 Supplied Reference Code

```
//0. Documentation
//*****************************************************************************
// Simple output C program based on the modification of the blinkLED.c code
// provided by TI.
//
// There are 7 steps to initialize a GPIO port for general use
//    i. Activate the clock for the port by setting the corresponding bit in
//         the RCGCGPIO register and then wait for status bit in PRGPIO
//         register to be true.
//   ii. Unlock port if using PD7.
//  iii. Disable analog function of the pin. (Upon reset default = disabled)
//   iv. Clear bits in PCTL to select regular digital function (see textbook
//         tables 4.1&4.2, noting upon reset the default function is digital).
//    v. Set the data direction register (i.e., input or output).  In the DIR
//         register, a bit set to: 0 = input, 1 = output.
//   vi. Clear bits in the alternate function register (remember pins / ports
//         can have alternate functionality when configured as such.
//  vii. Enable the digtal port
//
//  Note: Step i must be done first, but remaining steps may be performed in
//    any order.  If your intention is to use a GPIO pin in its default
//    digital state, then steps iii, iv, and vi can be omitted
//    (also ii if not using PD7).
//
//  TEDoyle
//  October 26, 2019

//*****************************************************************************
//1. Preprocessor Directives
//*****************************************************************************
// There are library files that define the peripherals, ports, and pins.
// Review the msp432e401y.h file to become familiar with the symbolic labels
// that have been preassigned to help simplify your programming. These labels
// can be different than your textbook, thus library familarity is important.

#include "msp.h"    // this libabry file chooses the approproate MSP library

//2. Functions
//*****************************************************************************

// For now, no functions outside of main().

//3. Main
//*****************************************************************************

int main(void)
{
  // step i.
  // Enable GPIO clocks for N & M peripheral
  SYSCTL->RCGCGPIO |= SYSCTL_RCGCGPIO_R12;  // Port N (onboard LED)
  SYSCTL->RCGCGPIO |= SYSCTL_RCGCGPIO_R11;  // Port M (GPIO for Lab 0 EPROM)

  // Check if the peripheral access is enabled.
  while(!(SYSCTL->PRGPIO & SYSCTL_PRGPIO_R12)); //N
  while(!(SYSCTL->PRGPIO & SYSCTL_PRGPIO_R11)); //M

  //skip steps ii, iii, iv

  // step v.
  /* Enable the GPIO pins for the onboard LEDs (PN1 = out, PN0 = out). */
  GPION->DIR = 0x03;

  /* Enable the GPIO pins for the EPROM (PM7:0 = out). */
  GPIOM->DIR = 0xFF;

  //skip steps vi

  // step vii.
```

```
69    GPION->DEN = 0x03;
70    GPIOM->DEN = 0xFF;
71
72    // main code
73    uint32_t counter = 0;
74
75    GPION->DATA = 0x01;   //initialize the bit values
76
77  while(1)
78    {
79        // Toggle GPIO pin value
80        GPION->DATA ^= BIT0;
81        GPION->DATA ^= BIT1;
82        for(counter = 0; counter < 1000000; counter++) {} //orig set to 200000
83        GPIOM->DATA = 0xFF;                               //bits 0-7 of Port M
84    }
85  }
```

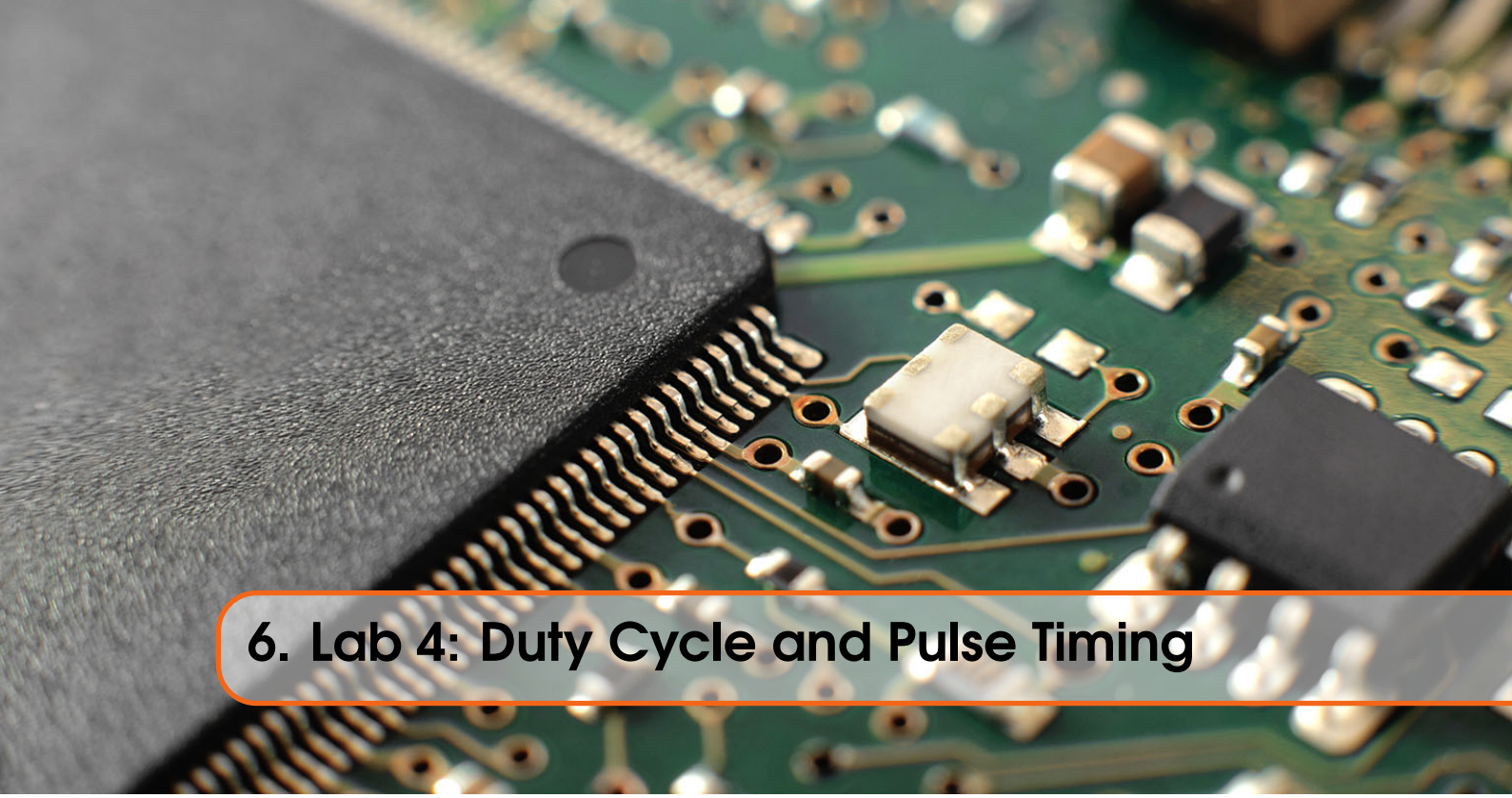
Listing 5.1: C code example for Lab 0

IV Part Four: Reason

6 Lab 4: Duty Cycle and Pulse Timing . . . 49
6.1 Objective
6.2 Pre-Laboratory Preparation
6.3 Integrated Circuits & Components
6.4 Laboratory Exercises and Milestones
6.5 Summary of Milestones & Evaluation Rubric
6.6 Submission Requirements
6.7 Grading Summary Table

7 Lab 5: Peripheral Interfacing 55
7.1 Objective
7.2 Pre-Laboratory Preparation
7.3 Integrated Circuits & Components
7.4 Using the Microcontroller's Internal Pull-up Resistors
7.5 Laboratory Exercises and Milestones
7.6 Summary of Milestones & Evaluation Rubric
7.7 Submission Requirements
7.8 Grading Summary Table

8 Lab 6: Project Deliverable 1 Early Integration 61
8.1 Objective
8.2 Instructions for Deliverable 1



6. Lab 4: Duty Cycle and Pulse Timing

6.1 Objective

The objective of this lab is to use embedded properties and timing using C language that is cross-assembled for ARM. To illustrate timing we will be using LEDs and a stepping motor.

6.1.1 Resources

You should review the following resources to prepare yourself for this lab and the future labs in this course:

1. Text:“Embedded Systems: Introduction to ARM Cortex-M Microcontrollers", 5th edition, 6th printing, January 2019, by Valvano – Chapter 8.7-8.8
2. Technical: MSP432E401Y - Microcontroller Data Sheet
3. Technical: MSP432E401Y - Technical Reference Manual
4. Technical: MSP-EXP432E401Y - Schematic
5. Technical: ARM Cortex M4 Processor Technical Reference Manual
6. Technical: ARM M Architecture Reference Manual
7. Technical: The Cortex-M4 Instruction Set
8. Technical: Velleman Stepper Motor with Driver - User Guide (`https://www.velleman.eu/downloads/29/vma401_a4v01.pdf`)
9. Technical: Velleman VMA401 - Stepper Motor Basics (`https://www.velleman.eu/downloads/29/infosheets/vma401_stepper_motor.pdf`)
10. Technical: Texas Instrument ULN2003 Datasheet - Motor Driver (`https://www.velleman.eu/downloads/29/infosheets/uln2003_datasheet.pdf`)
11. Reference: Primer on the stepper motor `https://www.youtube.com/watch?v=B86nqDRskVU`.

R Review how to create a new project in Keil in C.

6.2 Pre-Laboratory Preparation

[25 marks total]

1. Draw a flowchart for an ARM microcontroller program to flash an LED with a 50% duty-cycle. Set the period of one cycle as a variable to be later defined.[5 marks]
2. Based upon your flowchart, write a C program (and provide it in your prelab document) to flash the onboard LED. Assuming a 50% duty-cycle, the period of one cycle is defined based upon the least-significant-digit (LSD) of your student number. Include structured and commented source code (must have your name and student number) in your prelab document. [10 marks]
   LSD 0, 3, 6, 9: cycle period is 0.50 s.
   LSD 1, 4, 7: cycle period is 1.00 s.
   LSD 2, 5, 8: cycle period is 2.00 s.
3. If a stepper motor were to have 36 steps per rotation, how fast would the motor turn with a 10ms delay per step? Show calculations and answer in RPM. You may assume memory access time etc. are negligible. [5 marks]
4. If a stepper motor were to have 200 steps per rotation, calculate the delay per step for the motor to spin at 60 RPM. You may assume memory access time etc. are negligible. [5 marks]

6.3 Integrated Circuits & Components

The following integrated circuits are to be used in this laboratory:

Table 6.1: Integrated Circuits and Components for Lab 4

Identification	Description
ARM Cortex-M4F	TI MSP432E401Y Microcontroller
Stepper Motor	28BYJ-48
Motor driver	ULN2003
RGB LED	generic

Obtain the data sheets for each of the above devices. Familiarize yourself with their logical and electrical characteristics and bring a copy to your lab session.

6.4 Laboratory Exercises and Milestones

Read the following experiments and study the circuits as shown. Pre-filling your summary with the necessary truth tables and structuring your work such that you only need to record experimental output will allow you to focus on the milestones.

ENSURE YOUR MILESTONES ARE VISUALLY CHECKED AND RECORDED BY A TA – THERE ARE NO MARKS AFTERWARDS.

6.4.1 Variable Duty Cycle for PWM of an LED

Your pre-lab and prior studio work have implemented a 50% duty-cycle square wave. In this milestone, you will create a program that varies the the duty-cycle of your square wave. Varying the time high is one method of implementing pulse-width-modulation (PWM). For this milestone you can use either:

1. only the RED input to a RED-GREEN-BLUE LED (for now just ground the GREEN and BLUE), or
2. a RED LED.

Test your code at home using your hardware kit and AD2. Note that the bonus milestone will use all of the inputs for the RGB LED.

Milestone 6.1 Create a function called `IntensitySteps` that generates a square wave (frequency of 100 Hz) with a duty-cycle that steps from 0% to 100% (10% increase per step) and then decreases from 100% to 0% (10% decrease per step).

The function `IntensitySteps` should call a separate function (`DutyCycle_Percent`) which accepts an unsigned integer input indicating the percentage (out of 255) duty-cycle that should be output. For example, a value of 25 would generate ∼10% duty-cycle.

In order to see the effect of duty-cycle on the LED in the lab, each duty-cycle should be repeated ten times (e.g., 10% duty-cycle repeated ten times, 20% duty-cycle repeated ten times etc.).

Verify the duty-cycle of your square wave on the oscilloscope of your AD2.

35 marks ■

6.4.2 Stepper Motor

Stepper motors are popular for microcontroller application because you can control the position using a binary sequence. For this milestone we will be using PortM for four outputs labelled as A, B, A', and B'. For your convenience, the four bits selected from PortM should be contiguous. These four pins/bits will be used to output a control sequence to the stepper motor. Figure 6.1 illustrates the stepper motor's internal configuration. Referring to the stepper motor datasheet you will find one step pattern of interest for our unipolar stepper motor design, called "full step". Applying the bit sequence 0b1100, 0b0110, 0b0011, 0b1001 will make the rotor to turn clockwise. A really good primer on this stepper motor can be found here: `https://www.youtube.com/watch?v=B86nqDRskVU`

Milestone 6.2 Write a C program to control the rotation of the provided stepper motor. Do/determine the following:

1. Write function(s) through which you can control the direction of rotation and number of steps (assume full step mode).
2. Empirically determine the minimum time delay required between steps for the full step mode (hint - it cannot be 0). Report and demonstrate this value to the TA for your motor.
3. Determine how many steps it takes for full step mode to complete two full revolutions. Report this to the TA.
4. Demonstrate two full revolutions clockwise followed by one full revolution counter-clockwise (the same code execution should take care of both).

40 marks ■

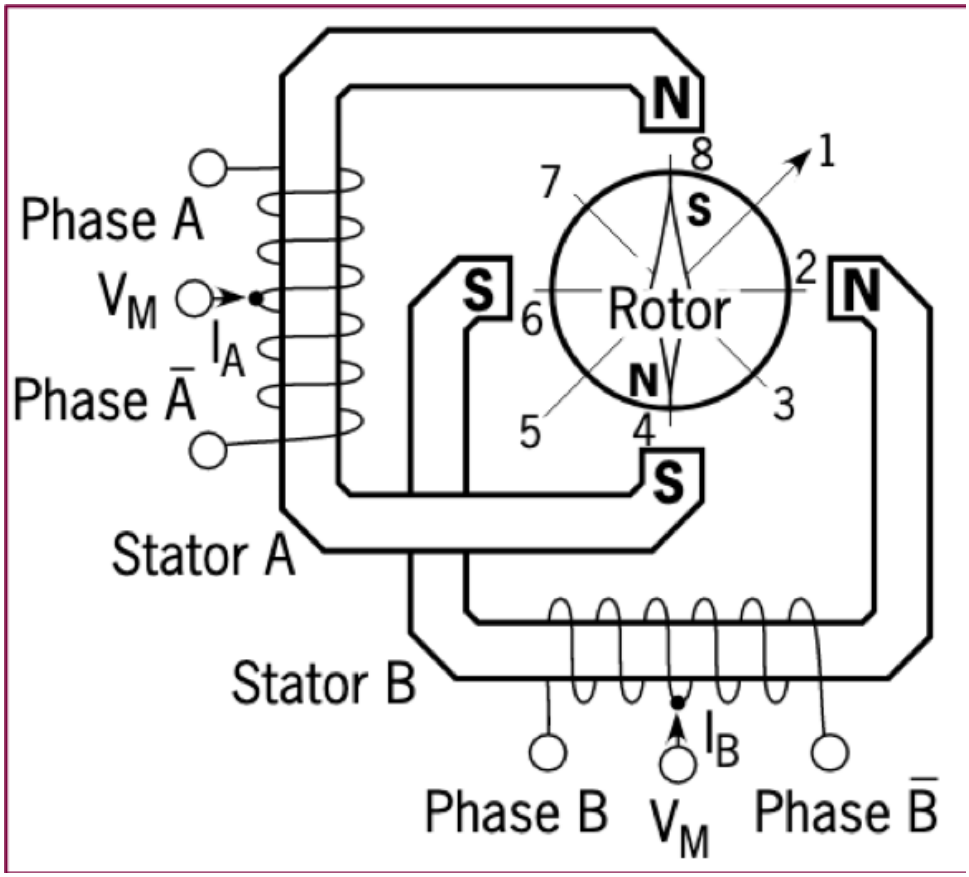


Figure 6.1: Unipolar Stepper Motor

6.4.3 BONUS: RGB LED

Milestone 6.3 Implement a duty-cycle/PWM method that implements separate duty-cycle functions (or one function accepting 3 inputs). Similar to milestone 1's `DutyCycle_Percent`, each function input shall be an unsigned 8-bit value to control the RED, GREEN, and BLUE percent duty-cycle. Three output pins must be configured and connected to the RGB LED. Demonstrate to the TA that you are able to modify the RGB colour of the LED based upon any percent duty-cycle input for each of RED, GREEN, and BLUE inputs. Your TA will provide you with new percentages for each and your code should work for those percentages.
10 marks ■

6.5 Summary of Milestones & Evaluation Rubric

TAs must visually verify and record your successful completion of:

1. Milestone 1
2. Milestone 2
3. BONUS Milestone

Be aware, if the TA does not have a record of the execution then a mark of zero is assessed.

6.6 Submission Requirements

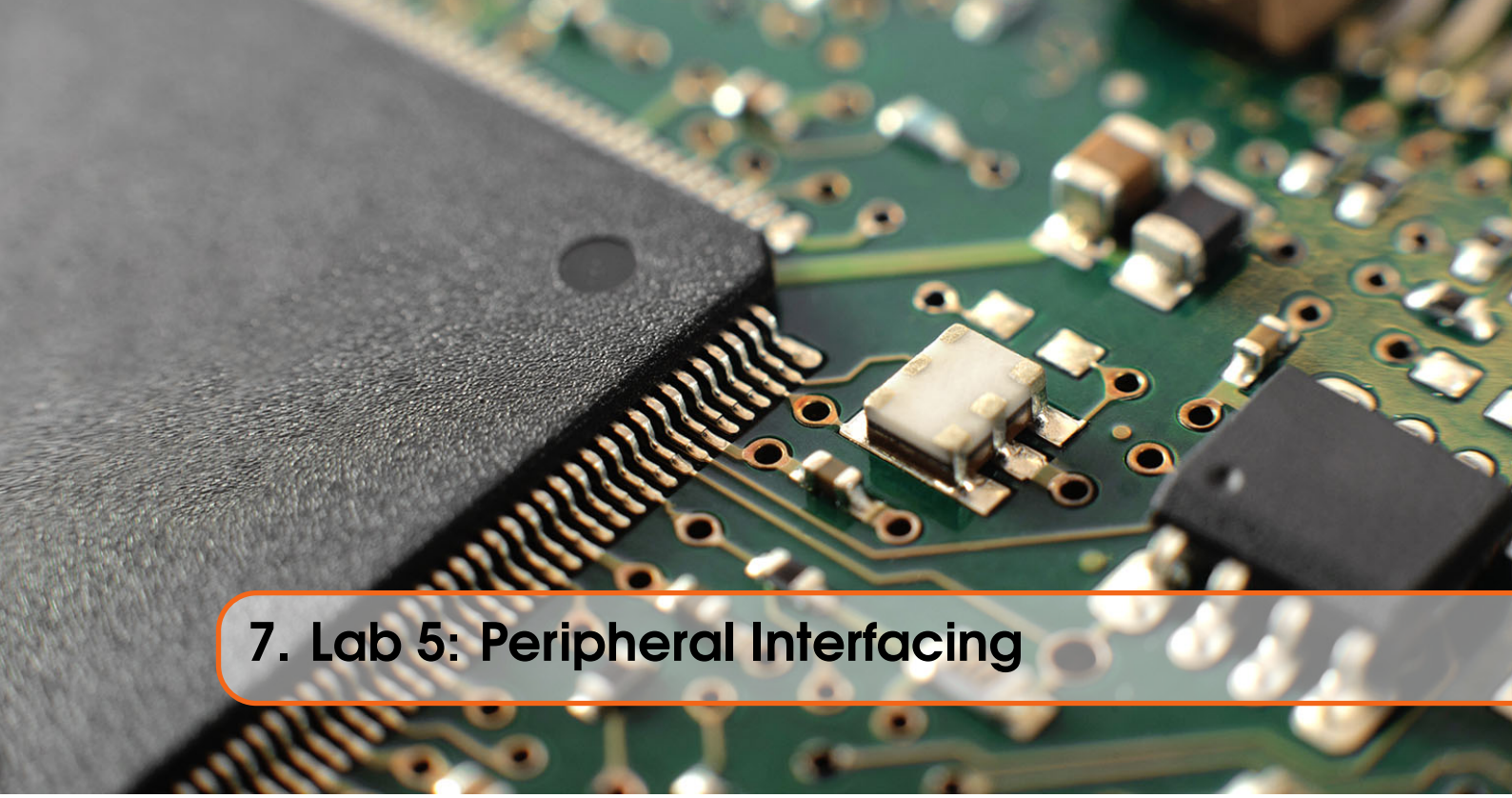
Refer to section 1.3.

Table 6.2: Evaluation Rubric

Criteria	Mark
Successfully demonstrate and explain to TA a fully correctly working milestone	100%
Demonstrate and explain a coherent attempt at milestone, but incorrect result	50%
No demonstration, cannot explain, or non-coherent attempt/demonstration	0

6.7 Grading Summary Table

Component	Weight	Grade
Pre-lab Questions	25	
Milestone 1	35	
Milestone 2	40	
BONUS Milestone	10	
Total	**100**	
Deductions		
Final Score		



7. Lab 5: Peripheral Interfacing

7.1 Objective

The objective of this lab is to start interfacing peripherals and integrating embedded concepts. In this lab we will be decoding keypad input. The keypad must be interfaced via hardware and software to create a functional user interface.

7.1.1 Resources

You should review the following resources to prepare yourself for this lab and the future labs in this course:

1. Text:“Embedded Systems: Introduction to ARM Cortex-M Microcontrollers", 5th edition, 6th printing, January 2019, by Valvano – Chapter 8.3, 8.5 (optional 8.4 uses different LCD).
2. Technical: MSP432E401Y - Microcontroller Data Sheet
3. Technical: MSP432E401Y - Technical Reference Manual
4. Technical: MSP-EXP432E401Y - Schematic
5. Technical: ARM Cortex M4 Processor Technical Reference Manual
6. Technical: ARM M Architecture Reference Manual
7. Technical: The Cortex-M4 Instruction Set
8. Technical: 4x4 Keypad (`https://www.mouser.ca/datasheet/2/626/Keypads_96-335254.pdf`)
9. Reference: American Standard Code for Information Interchange (ASCII) `http://en.wikipedia.org/wiki/ASCII`

R Review how to create a new project in Keil in C.

7.2 Pre-Laboratory Preparation

[25 marks total]

1. Referring to textbook section 4.2.2, figure 4.12, from a programmer's perspective explain how the two states of the switch (open, closed) work for negative logic, external. You may assume the port pin is fully pre-configured as an input. [5]
2. Review textbook figure 8.15. This figure illustrates an electrical structure similar to the lab's Grayhill 96BB2-006-F 4x4 keypad. From the figure, determine if the switches are configured as:
   (a) positive logic, external
   (b) negative logic, external
   [5]
3. Create a flowchart for a program that scans the 4x4 keypad using 4-output and 4-input GPIO pins and identifies which single button has been pressed (assume only one can be pressed at a time). [15]

7.3 Integrated Circuits & Components

The following integrated circuits are to be used in this laboratory:

Table 7.1: Integrated Circuits and Components for Lab 1

Identification	Description
ARM Cortex-M4F	TI MSP432E401Y Microcontroller
4x4 Keypad	Grayhill 96BB2-006-F

Obtain the data sheets for each of the above devices. Familiarize yourself with their logical and electrical characteristics and bring a copy to your lab session.

7.4 Using the Microcontroller's Internal Pull-up Resistors

In the Pre-Lab exercise, you focused on negative logic, external, with regard to connecting switches to the MCU. Remember, in this interfacing method, external pull-up resistors are used. You can use this method to do this lab but do not forget to use external pull-up resistors to connect the MCU's input pins to 3.3V. However, you can also use the MCU's internal pull-up resistors to connect to high voltage. You can do so by setting the respective port's GPIOPUR (GPIO Pullup Select) register accordingly. For example, to turn on the pull-up resistor of Port M Pin 0, you can use: GPIO_PORTM_PUR_R = 0x01; See the MCU's Technical Reference Manual (Pages 1219, 1220).

7.5 Laboratory Exercises and Milestones

Read the following experiment and study the circuits as shown. Creating a Pre-filled summary with the necessary truth tables and structuring your submission such that you only need to record experimental output will allow you to focus on the milestones.

ENSURE YOUR MILESTONES ARE VISUALLY CHECKED AND RECORDED BY A TA – THERE ARE NO MARKS AFTERWARDS.

7.5.1 4x4 Keypad Button Identification

In Studio you have already completed several keypad decoding scenarios (e.g., 1-button, 1x2 keypad, 2x2 keypad, etc.). For this milestone you will be extending that work to identify which button was pressed on the full 4x4 keypad.

Figure 7.1: Input Devices Keypad 4x4 Black Phone Legend/White

Table 7.2: Pins and Ports for Lab 3

Port	Pins	Application
Port E	3:0	Keypad scanning (output)
Port M	3:0	Key press decode (input)

Milestone 7.1 Consult the data sheet of the 96BB2-006-F 4x4 Keypad to determine the order of its pins and then use the scanning technique illustrated in your textbook (Valvano chapter 8.5) and Studio to connect the keypad to the microcontroller with Table 7.2 as your reference. Write a C program to determine which key has been pressed and store the 8-bit PM[3:0] PE[3:0] value of the scan in Keil memory according to Table 7.3 (Hint: You will need to work in the debugger). Note: The binary value will be used in a following milestone.

25 marks ■

7.5.2 Key Decode

Once a unique code for a key is identified, it must be decoded to provide meaning to the key. For example, your code may have identified that key 11101110 has been pressed, but what does that mean to the program or in the application context? For this part of the lab you will take the identified key and decode to its binary value, as shown in Table 7.3. Note this could just as easily be the ASCII character code.

Milestone 7.2 Write a function that takes the unique binary key code as input (PM[3:0] PE[3:0]) and returns the binary value from Table 7.3. This binary value should be placed in the Keil memory for TA confirmation.
25 marks ■

7.5.3 LED Display

Configure four GPIO pins as outputs to externally display the decoded key's binary value. While not required, you may choose to use the onboard LEDs because this will be required in your next lab.

Table 7.3: Key Press Decoding

Key Pressed	Binary Key Value	Scanning Input Code PM[3:0]	Scanning Output Code PE[3:0]
0	0000		
1	0001		
2	0010		
3	0011		
4	0100		
5	0101		
6	0110		
7	0111		
8	1000		
9	1001		
A	1010		
B	1011		
C	1100		
D	1101		
*	1110		
#	1111		

Milestone 7.3 Extend the above programming to take the 4-bit binary value of the decoded key press and output to the LEDs. The LEDs should be ordered such that the TA can directly read the binary value and match the value in memory. Note that the code for button 0 will not light any LED.
25 marks ■

7.5.4 BONUS: 7-Segment Display

Note for this bonus you will need a seven segment display and driver.

Milestone 7.4 Using your code from this lab's milestones, connect your 4-bit binary output to a 7-segment display such that the button pressed is displayed.
10 marks ■

7.6 Summary of Milestones & Evaluation Rubric

TA must visually verify and record your successful completion of:

1. Milestone 1
2. Milestone 2
3. Milestone 3
4. BONUS Milestone

Be aware, if the TA does not have a record of the execution then a mark of zero is assessed.

7.7 Submission Requirements

Refer to section 1.3.

Table 7.4: Evaluation Rubric

Criteria	Mark
Successfully demonstrate and explain to TA a fully correctly working milestone	100%
Demonstrate and explain a coherent attempt at milestone, but incorrect result	50%
No demonstration, cannot explain, or non-coherent attempt/demonstration	0

7.8 Grading Summary Table

Component	Weight	Grade
Pre-Lab Questions	25	
Milestone 1	25	
Milestone 2	25	
Milestone 3	25	
BONUS Milestone	10	
Total	**100**	
Deductions		
Final Score		

8. Lab 6: Project Deliverable 1 Early Integration

8.1 Objective

The objective of this lab time is for you to demonstrate the early integration of your project. You will be interviewed during this lab time on your integration of the embedded components with the goal of a functioning early system. In this lab we will be combining your previous labs and studios to create the core project platform, including stepping motor, keypad, push button inputs, and LED outputs. Remember to refer to the pin map on figure 8.1 when selecting pins from the microcontroller realestate to ensure pin functionality is not being overlapped. For this lab we have specified the pins, but they may not match prior lab code and they may not be suitable for your specific final project requirements. **This is not counted as a regular lab - this is Deliverable 1 for the project**

8.2 Instructions for Deliverable 1

You will have to do a demo and an interview according to the details below.

8.2.1 Logistics

1. The interview schedule is posted under on Avenue. Please always consult the latest version posted. The schedule tells you the time and location for your interview.
2. You should arrive 10-15 minutes before your scheduled time.
3. You should already have as much of your deliverable setup as possible.
4. You will be allowed into the lab 10 minutes before your scheduled time to complete your setup. You will not have a large area in the lab to setup so you have to have most of it already done and work efficiently and neatly to get ready to demo your work.
5. At your scheduled time, you will be asked to move to your station where a TA will be available to assess your work.
6. If you are not ready to start on time, this cuts into your own time. We will not extend the end time for your interview. Your grade will be based on how much can be done during the available time.

7. Your demo and interview together are 10 minutes long.

8.2.2 Demo

The items below are intended to clarify and provide more detail. There should be no contradiction between the items below and what is in the current project specification. If there is a contradiction, the project specification is correct.

Table 8.1: Pins and Ports for Lab 6

Port	Pins	Application
Port N	1:0	Output
Port F	4,0	Output
Port M	1:0	Input
Port J	1:0	Input
Port H	3:0	Stepper motor control

Integrated Motor Control

For early integration you will not be required to implement your specified Individualized Operational Parameters. Instead this early integration will be the same for everyone.

You will need to implement 4 push buttons as inputs (2 onboard and 2 user implemented):

1. Button 0 - onboard User Switch 1 using input PJ0
2. Button 1 - onboard User Switch 2 using input PJ1
3. Button 2 - user implemented momentary push button using PM1
4. Button 3 - user implemented momentary push button using PM0

You will need to implement outputs for the stepper motor and status LEDs:

1. Stepper control using PH3, PH2, PH1, PH0
2. Status Output 0 (LED0) - onboard User LED 1 using output PN1
3. Status Output 1 (LED1) - onboard User LED 1 using output PN0
4. Status Output 2 (LED2) - onboard User LED 1 using output PF4
5. Status Output 3 (LED3) - onboard User LED 1 using output PF0

R For troubleshooting, consider initially writing C code to have each push button control a single LED.

Adapting work from prior lab(s) and studio(s) that used the stepping motor and push buttons, use the above specified button inputs and write a C program that accepts these four momentary push buttons to implement the functionality described in Points 1 to 4 below for stepper motor control. A state-machine design approach is recommended.

Milestone 8.1

1. Button 0 - Start/Stop: When pressed, the operation of the motor will be complemented. Set (turn on) Status Output 0 (LED0) when the motor is operating. [Default: Motor Stopped and Status Output Clear (off)]
2. Button 1 - Direction: When pressed, the direction of the motor rotation will be reversed. Set Status Output 1 when the motor is rotating clockwise. Clear Status Output 1 when the motor is rotating counterclockwise [Default: Motor Rotating CW and LED 1 On]

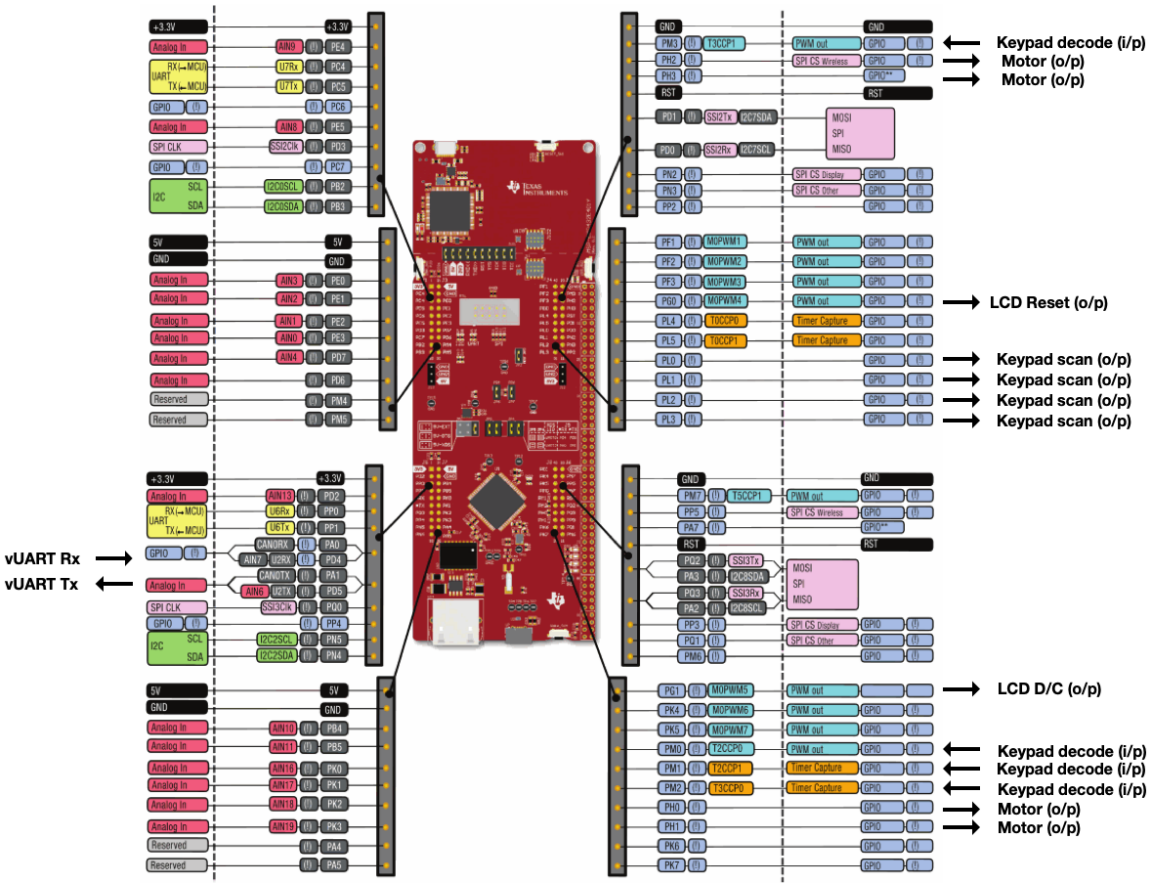


Figure 8.1: Pin Map for Integration

3. Button 2 - Angle: When pressed, the angle of motor rotation at which an onboard LED blinks should toggle between 11.25^o and 45^o. Turn on Status Output 2 when the motor angle is 11.25^o, otherwise clear. [Default: 11.25^o and LED 2 On]. Additionally, Status Output 3 blinks for each step of rotation.
4. Button 3 - Home: When pressed, your motor should return to the user defined home position (i.e., 0^o).

■

R As an example of item 3 - initially, once the motor starts rotating, LED 2 would be On, and LED 3 would blink every time the motor's outer shaft completes 11.25^o of rotation (i.e., LED 3 blinks 32 times in every full rotation). If Button 2 is pressed, LED 2 switches Off, and LED 3 only blinks whenever the motor completes 45^o of rotation (i.e., now LED 3 blinks 8 times in every full rotation). If the motor is not rotating (i.e., it is Off), both LEDs 2 and 3 should be Off.

8.2.3 Interview

You will be asked 2-3 questions regarding your project. Questions could be about how you implemented each of the above requirements, how the stepper motor works, how a pushbutton works, how you would alter the hardware or the code to operate differently (e.g. how to make your buttons active low or active high; how to change the direction of rotation; etc.), or other similar questions.

Questions could also be about thinking ahead to future project deliverables. E.g. when you implement a student specific clock speed, what parts of your current code would be impacted? What debugging might be useful to use a 2nd LED for?

Each student will be asked 2-3 questions so it will not be the same questions for each student. The grading for the questions will be on four levels:

- Below expectations: no answer; factually wrong answer; or very unclear answer (0%)
- Marginal: a basically correct but incomplete answer (50%)
- Meets expectations: the answer is correct and complete (100%)
- Exceeds expectations: a concise answer that demonstrates a deep understanding of the design, including precise use of terminology (105%)

Once your demo and interview are done, please leave the lab within 5 minutes to allow other students to continue with their interviews without delay.

V

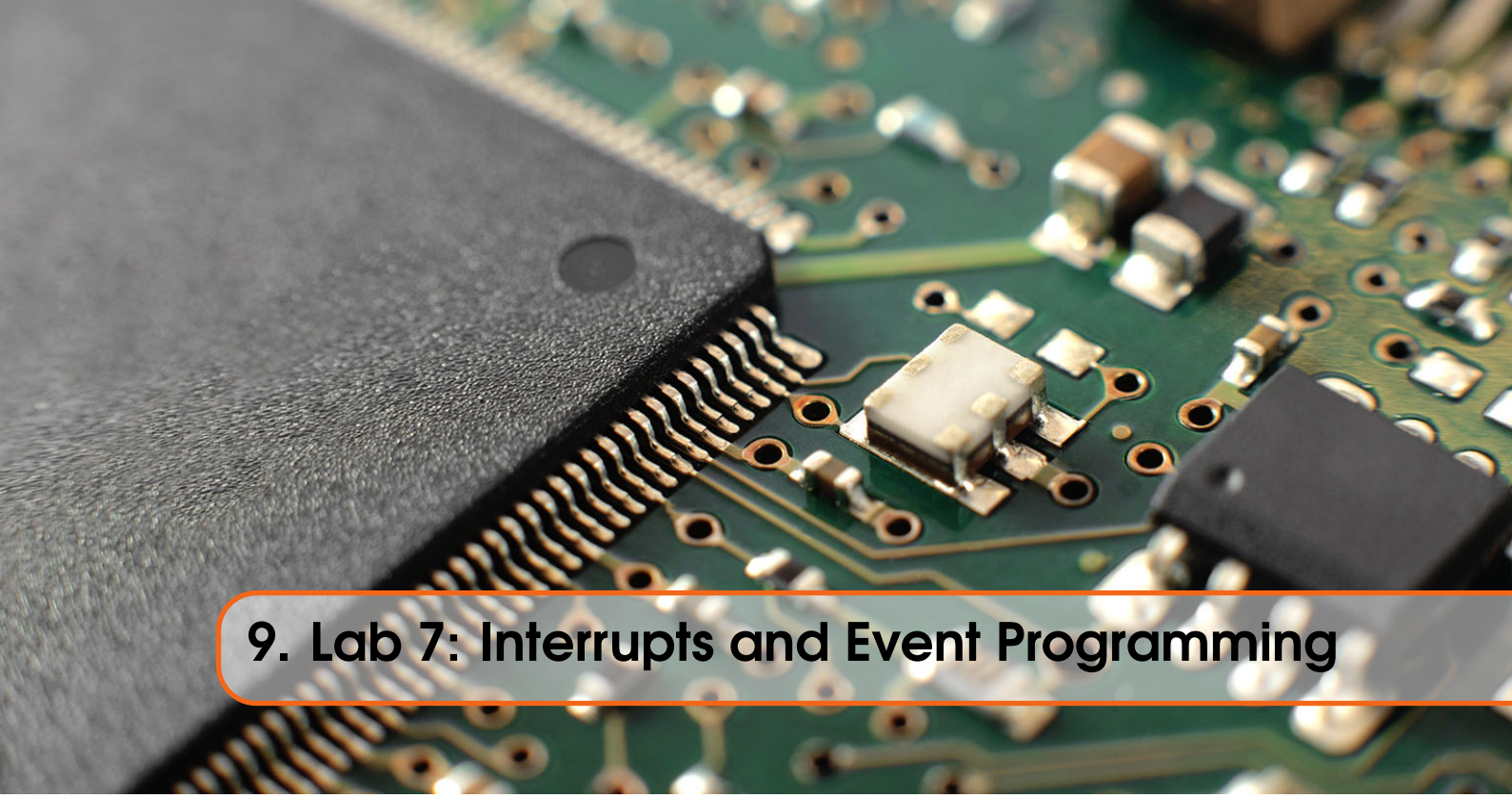
Part Five: Act

9 Lab 7: Interrupts and Event Programming 67

9.1 Objective
9.2 Pre-Laboratory Preparation
9.3 Integrated Circuits & Components
9.4 Laboratory Exercises and Milestones
9.5 Summary of Milestones & Evaluation Rubric
9.6 Submission Requirements
9.7 Grading Summary Table

10 Lab 8: Collecting Distance Data 71

10.1 Objective
10.2 Pre-Laboratory Preparation
10.3 Integrated Circuits & Components
10.4 Laboratory Exercises and Milestones
10.5 Summary of Milestones & Evaluation Rubric
10.6 Submission Requirements
10.7 Grading Summary Table



9. Lab 7: Interrupts and Event Programming

9.1 Objective

The objective of this lab is to gain experience using event based programming.

9.1.1 Resources

You should review the following resources to prepare yourself for this lab and the future labs in this course:

1. Textbook Chapter 9.
2. Studios 7A to 7D.

9.2 Pre-Laboratory Preparation

[20 marks total]

1. When referring to interrupt triggering via GPIO, explain the difference between falling edge, rising edge, low level, and high level. (Hint - see textbook section 9.5) [5 marks]
2. What five conditions must be true for an interrupt to occur? [5 marks]
3. How do you enable interrupts? [5 marks]
4. What are the steps that occur when an interrupt is processed? [5 marks]

9.3 Integrated Circuits & Components

The integrated circuits mentioned in Table 9.1 are to be used in this laboratory.

Obtain the data sheets for each of these devices. Familiarize yourself with their logical and electrical characteristics and bring a copy to your lab session.

9.4 Laboratory Exercises and Milestones

Read the following experiment and study the circuits as shown. Pre-filling your summary with the necessary truth tables and structuring your work such that you only need to record experimental

Table 9.1: Integrated Circuits and Components for Lab 7

Identification	Description
ARM Cortex-M4F	TI MSP432E401Y Microcontroller
ToF Sensor	Polo 3415 (VL53L1X Time-of-Flight Distance Sensor)
Software	Realterm or similar

output will allow you to focus on the milestones.

ENSURE YOUR MILESTONES ARE VISUALLY CHECKED AND RECORDED BY A TA – THERE ARE NO MARKS AFTERWARDS.

Table 9.2: Pins and Ports for Lab 7

Port	Pins	Application
Port B	3:2	I2C

9.4.1 Periodic Interrupt

In Studio you completed periodic interrupt triggering for an onboard LED. Reuse this code to flash an LED and output the same square wave to a GPIO pin of your choice. Using the Analog Discovery 2 scope, verify the timing.

Milestone 9.1 Instead of a generating a square wave using the polling method (Lab 4), we can use interrupts. Implement a periodic interrupt that triggers every 1s. Write an interrupt service routine that will flash an onboard LED for 250 ms and also output to a GPIO pin. The resulting program will produce a square wave with period of 1s and a 25% duty cycle.

Verify the timing by connecting the GPIO pin to a scope probe on your Analog Discovery 2. Ensure a photo of the AD2 scope display goes in your respective theme report, noting the pulse and interrupt periods.

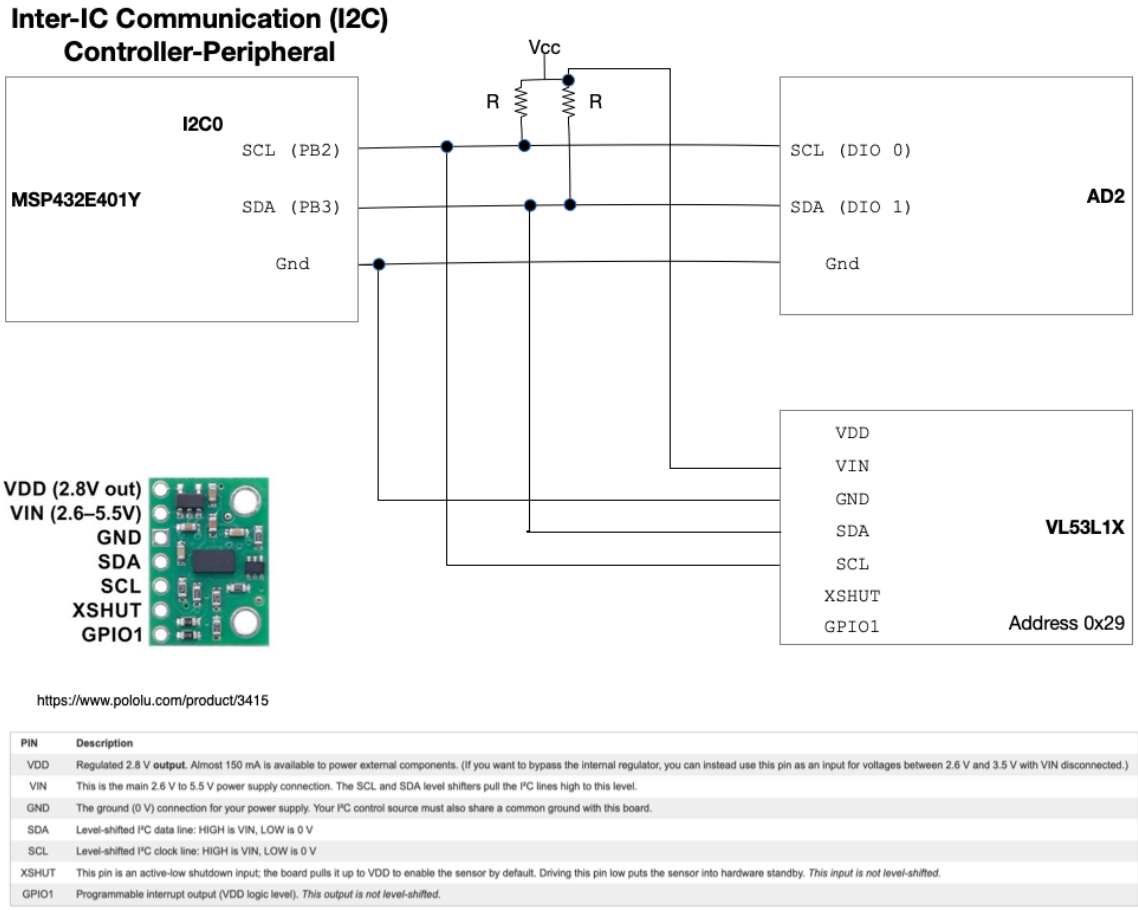
40 marks ■

9.4.2 GPIO Interrupt

In Studio you completed GPIO push-button interrupt triggering for an onboard LED. For this milestone you will need to connect the Time-of-Flight sensor the same way you did in studio (you should also connect your AD2 for debugging in Logic spy mode).

Reuse studio code to generate an interrupt when a button is pressed. In the interrupt service routine transmit (using I2C) the peripheral device (Time-of-Flight) address (0x29) and one byte of data (0x00). Using a lab oscilloscope or your AD2 scope, verify correct data transmission (you will need to decode transmission).

Milestone 9.2 Implement GPIO interrupt as noted above using the relevant studio code to transmit data to the ToF device and verify the data transmission using a lab oscilloscope or your



PIN	Description
VDD	Regulated 2.8 V **output**. Almost 150 mA is available to power external components. (If you want to bypass the internal regulator, you can instead use this pin as an input for voltages between 2.6 V and 3.5 V with VIN disconnected.)
VIN	This is the main 2.6 V to 5.5 V power supply connection. The SCL and SDA level shifters pull the I²C lines high to this level.
GND	The ground (0 V) connection for your power supply. Your I²C control source must also share a common ground with this board.
SDA	Level-shifted I²C data line: HIGH is VIN, LOW is 0 V
SCL	Level-shifted I²C clock line: HIGH is VIN, LOW is 0 V
XSHUT	This pin is an active-low shutdown input; the board pulls it up to VDD to enable the sensor by default. Driving this pin low puts the sensor into hardware standby. *This input is not level-shifted.*
GPIO1	Programmable interrupt output (VDD logic level). *This output is not level-shifted.*

Figure 9.1: Wiring layout for Lab 7. Use R = 5 kΩ

AD2 scope.

40 marks ■

9.4.3 BONUS: Interrupt Buttons for transmission to PC

Milestone 9.3 Implement GPIO interrupts for two separate buttons (1 interrupt each). Button 1 transmits your student number via the UART (virtual com port, communication method covered in Studio) to the PC and display using an application (e.g., RealTerm, Python, etc.). Button 2 transmits (also through UART) the four characters: 2DX3.

10 marks ■

9.5 Summary of Milestones & Evaluation Rubric

TA must visually verify and record your successful completion of:

1. Milestone 1
2. Milestone 2
3. BONUS Milestone

Be aware, if the TA does not have a record of the execution then a mark of zero is assessed.

Table 9.3: Evaluation Rubric

Criteria	Mark
Successfully demonstrate and explain to TA a fully correctly working milestone	100%
Demonstrate and explain a coherent attempt at milestone, but incorrect result	50%
No demonstration, cannot explain, or non-coherent attempt/demonstration	0

9.6 Submission Requirements

Refer to section 1.3.

9.7 Grading Summary Table

Component	Weight	Grade
Lab Questions	20	
Milestone 1	40	
Milestone 2	40	
BONUS Milestone	10	
Total	**100**	
Deductions		
Final Score		

10. Lab 8: Collecting Distance Data

10.1 Objective

The objective of this lab is to communicate with, and collect data from, a digital sensor.

10.1.1 Resources

You should review the following resources to prepare yourself for this lab and any future work in this course:

1. You will need to develop a method of mounting your ToF sensor to your stepper motor's rotational shaft (lego bricks have been successfully used, but you are free to use any apparatus).
2. Review the provided code from Studio. It was written to support data collection from VL53L1X using the Ultra Light Driver.
3. I2C based upon MSP432E4 Reference Manual Chapter 19.
4. VL53L1X user manual.
5. Code implementation was based upon format specified in vl531X.pdf pg19-21
6. Prior lecture, labs, studio on I2C, UART, and programming paradigm (polling vs event).
7. Textbook, Chapter 11.6 for examples of system dataflow diagrams and modular design.

10.2 Pre-Laboratory Preparation

[30 marks total]

1. Briefly describe how the VL53L1X ToF sensor works to obtain a distance measurement. [10 marks]
2. What units does the VL53L1X ToF sensor return? [5 marks]
3. How many bits is one distance measurement for the VL53L1X ToF sensor? [5 marks]
4. How many data bits can be transmitted at once using UART serial communication? [5 marks]
5. How does the UART transmitter/receiver send/receive data that is bigger than one serial package will allow? [5 marks]

10.3 Integrated Circuits & Components

The following integrated circuits are to be used in this laboratory:

Table 10.1: Integrated Circuits and Components for Lab 8

Identification	Description
ARM Cortex-M4F	TI MSP432E401Y Microcontroller
ToF Sensor	Polo 3415 (VL53L1X Time-of-Flight Distance Sensor)
Stepper Motor	28BYJ-48
Motor driver	ULN2003
Software	Realterm or similar

Obtain the data sheets for each of the above devices. Familiarize yourself with their logical and electrical characteristics and bring a copy to your lab session.

10.4 Laboratory Exercises and Milestones

Read the following experiment and study the circuits as shown. Pre-filling your summary with the necessary designs (i.e., flow charts) and structuring your work such that you only need to record experimental output will allow you to focus on the milestones.

ENSURE YOUR MILESTONES ARE VISUALLY CHECKED AND RECORDED BY A TA – THERE ARE NO MARKS AFTERWARDS.

Table 10.2: Pins and Ports for Lab 8

Port	Pins	Application
Port B	3:2	I2C
Port H	3:0	Stepper motor control
Port A	1:0	Virtual com port UART

10.4.1 Read Time-of-Flight Sensor ID and Module Type

In Studio you communicated with the distance sensor to collect its ID via I2C. In this milestone you will extend that code to serially transmit the sensor ID to a PC over UART. Obtaining the device ID is often the first test done with any I2C device to confirm it is on the bus and functioning.

Milestone 10.1 Implement a serial UART communication to transmit the VL53L1X model ID and module type. You can start by using your relevant studio code as base and then use the following three API function calls:

1. `status = VL53L1_RdByte(dev, 0x010F, &byteData);` //for model ID (0xEA)
2. `status = VL53L1_RdByte(dev, 0x0110, &byteData);` //for module type (0xCC)
3. `status = VL53L1_RdWord(dev, 0x010F, &wordData);` //for both model ID and type

The PC should receive 1-byte model ID, 1-byte module type, and both bytes from `wordData`.

Refer to the wiring diagram for this lab (Figure 10.1). Note the AD2 has been removed and the pull-up resistors are no longer needed because the VL53L1X has pull up resistors on board its PCB.
25 marks ■

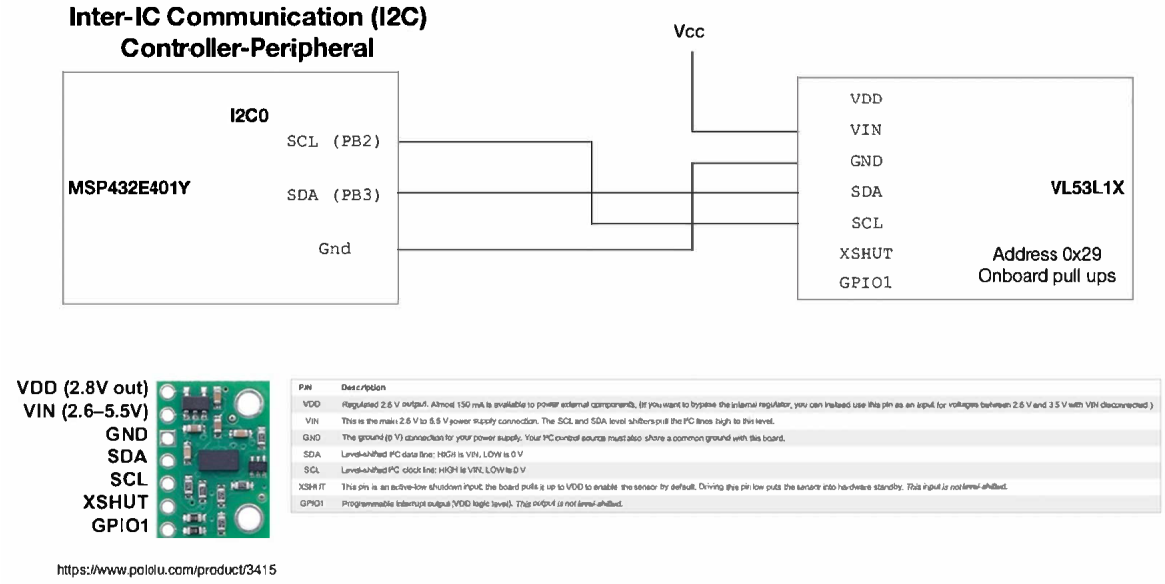


Figure 10.1: Wiring layout for Lab 8

10.4.2 Time-of-Flight Sensor Measurement via I2C

You have been supplied code for communicating with the Time-of-Flight (ToF) VL53L1X sensor in studio. This is a digital module that we will require the use of I2C for communication and control. The code provided is based upon the manufacturer API for use of the ultra light drivers[1].

The API functions provided to you are:

1. `VL53L1X_BootState`
2. `VL53L1X_SensorInit`
3. `VL53L1X_StartRanging`
4. `VL53L1X_CheckForDataReady`

[1] See: UM2510 A guide to using the VL53L1X ultra lite driver `https://www.st.com/content/ccc/resource/technical/document/user_manual/group1/63/e0/7d/f1/0e/52/4d/cd/DM00562924/files/DM00562924.pdf/jcr:content/translations/en.DM00562924.pdf`

5. `VL53L1X_GetDistance`
6. `VL53L1X_ClearInterrupt`
7. `VL53L1X_Stop`

The Studio sample code provided uses a polling approach to obtain measurement. In Studio you captured data using the ToF VL53L1X sensor. In prior labs you have implemented both UART and stepper motor code.

Milestone 10.2 Mount the ToF VL53L1X sensor onto your stepper motor and connect both to the MCU. Combine the ToF VL53L1X sensor code with prior stepper motor code to capture 8 (45^o intervals) or more distance measurements from a single 360^o horizontal scan.

Convert these 8 measurements to an x,y coordinate system and plot using a software program like Excel.

45 marks ■

10.4.3 BONUS 1: Time-of-Flight Sensor Measurement to Python

In Studio you used PySerial to communicate between the microcontroller and the PC. In this lab milestone you will be extending last lab by communicating ToF data to the PC using PySerial.

By now you have combined the ToF VL53L1X sensor with the stepper motor to capture 8 (45^o intervals) or more distance measurements from a single 360^o horizontal scan. You then converted these 8 measurements to an x,y coordinate and plotted them using a software program like Excel.

Milestone 10.3 Implement a serial UART communication to transmit one horizontal scan (8 measurements) of ToF measurement data to the PC via PySerial.

Your Python code should then translate your measurements to an x,y,z coordinate system and output to a file called `tof_radar.xyz`. This file should be in the XYZ format (example in W9 lecture).

10 marks ■

10.4.4 BONUS 2: Visualize Time-of-Flight Measurement Data

Once the xyz coordinate file has been written, we now have a small example of a point cloud. A point cloud is a spatial sampling of a space or object. Point clouds can be used to generate 3D representations of the space or object. For this milestone we will use the Open3D Python package to convert our coordinates into a horizontal slice of a 3D space. This would be similar to a radar sweep or a LIDAR scan used in navigation. By using repeated horizontal scans, a much better representation of the space would result.

Milestone 10.4 Employing the Open3D (or similar), create a plot of the xyz data with connecting lines for 3D visualization.

10 marks ■

10.5 Summary of Milestones & Evaluation Rubric

The TA must visually verify and record your successful completion of:

1. Milestone 1
2. Milestone 2
3. BONUS 1 Milestone
4. BONUS 2 Milestone

Be aware, if the TA does not have a record of the execution then a mark of zero is assessed.

Table 10.3: Evaluation Rubric

Criteria	Mark
Successfully demonstrate and explain to TA a fully correctly working milestone	100%
Demonstrate and explain a coherent attempt at milestone, but incorrect result	50%
No demonstration, cannot explain, or non-coherent attempt/demonstration	0

10.6 Submission Requirements

Refer to section 1.3.

10.7 Grading Summary Table

Component	Weight	Grade
Pre-Lab Questions	30	
Milestone 1	25	
Milestone 2	45	
BONUS 1 Milestone	10	
BONUS 2 Milestone	10	
Total	**100**	
Deductions		
Final Score		

VI

Appendix A

11 Getting to Hello 79
11.1 Getting Started...
11.2 macOS – Virtualization
11.3 macOS – Bootcamp
11.4 Windows
11.5 Say Hello
11.6 Software Version Summary

11. Getting to Hello

When starting out with any new programming platform, the first thing every developer must to do is install the new development environment and verify that it works. This is where the “Hello World!" code originated in the programming domain. This si done as a quick test to compile and run a basic set of instructions that communicate to the programmer. For embedded development, we do not (yet) have the initial luxury of an existing operating system that fully supports a display, keyboard, mouse, etc. For our first attempt at an embedded “Hello World!", we will assemble pre-tested code that will flash an LED.

Before installing any new software or making any modifications to your computer configuration you must create a restore point (on Windows) and back up all of your data. Backing up your data should be a regular and frequent habit that occurs independent of this course.

11.1 Getting Started...

To start, the development environment needs to be installed. In this course we will be developing using the Keil MDK (Microcontroller Development Kit). Similar to how you would use an IDE for software development, the Keil MDK allows you to create, build, and debug embedded applications.

To obtain Keil MDK 5 use this link `https://www2.keil.com/mdk5` and enter the following information:

- First & Last Name
- McMaster email address
- Company: McMaster Computer Engineering 2DX3 (or 2DX4)
- Device: MSP432E401Y

Keil MDK only runs on Windows and is free for development under 32KB (thus free for this course). If you are running a Windows based computer, skip to section 11.4. If you are running a macOS based computer then you will need to choose either Virtualization (refer to section 11.2) or

Bootcamp (refer to section 11.3) to install Windows.

How do you choose between Virtualization or Bootcamp?

Choose **Virtualization** if your computer has an i5 or better processor, at least 8GB RAM, and 50 GB or more of free disk space.

Choose **Bootcamp** if your computer does not meet the recommended specification for Virtualization. Each virtualization software will have a minimum recommended specification which you can research further if interested; however, the above specifications are recommended from experience.

11.2 macOS – Virtualization

There are three primary options for virtualization software (listed alphabetically):
Oracle VM VirtualBox `https://www.virtualbox.org`
Parallels Desktop `https://www.parallels.com`
VMware Fusion `https://www.vmware.com/ca/products/fusion.html`

You are free to select any one of these options, but note that only Oracle VM VirtualBox is free. The others may be available through the campus computer store with an educational license for purchase. In all cases, the choice and configuration are the student responsibility. We will help when and where we can on a best effort basis. It is imperative that you sort out your development environment install, configuration, and testing as soon as possible.

We have selected Oracle VM VirtualBox for our 2DX4 development environment[1].

Follow the manufacturer installation procedure for the virtualization software. Once the virtualization software is installed, you will need to configure a virtual computer for a Windows installation. Typically, once you configure the virtualized client you point the application to your Windows install drive/file/ISO and proceed as a normal operating system installation. For our development environment we have chosen a 64-bit Windows 10 Professional installation. Finally, once Windows is installed then install any VM support tools to improve macOS integration (in VirtualBox you will need to install Guest Additions from the Devices menu).

11.3 macOS – Bootcamp

Bootcamp is a boot loader software application. This type of application runs prior to an operating system booting. Depending upon configuration, this can either autoboot to the last OS used, or a menu can load for a choice of which OS the start. This is implemented by the boot loader choosing which disk and/or partition is considered the target. While conceptually logical, the practical consideration is that when booting into the “target", the “other" OS disk/partition becomes unavailable. This can be problematic because files and disk space that are available using one OS are not available using the other (an online drive can alleviate this, or a properly configured shared drive/partition).

[1] VMware Fusion was tested, but failed to connect to the microcontroller board. This was due to either to our incorrect configuration or VMWare’s handling of USB connectivity under macOS Catalina. Cursory research found mention of the issue possibly being related VMware/macOS having some issues around USB 2 devices.

In order to install a boot loader, like Bootcamp, a software application will need to make significant changes to your hard drive organization (i.e., modifying partitions, partition tables, and boot records). A power failure, program interruption, reset, etc. will likely result in a non-bootable computer and a **loss of all data.**

The following link describes the method of installing Bootcamp: `https://support.apple.com/en-ca/HT201468`

11.4 Windows

Install Windows, run all updates, and then plug the MSP432E401Y board. Figure 11.1 notes the correct USB port with which to connect.

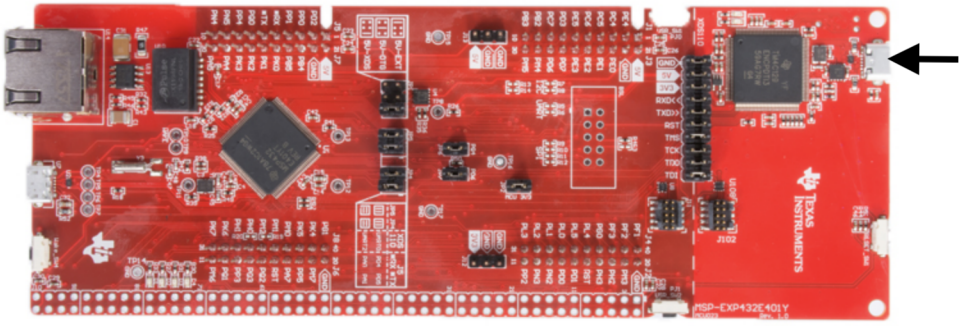


Figure 11.1: Top view of MSP432E401Y board with USB Debug location noted

Once installed, start Windows and go to `VirtualBox VM -> Devices -> USB` and select the MSP432E401Y to attach the TI board to Windows.

11.4.1 Keil MDK

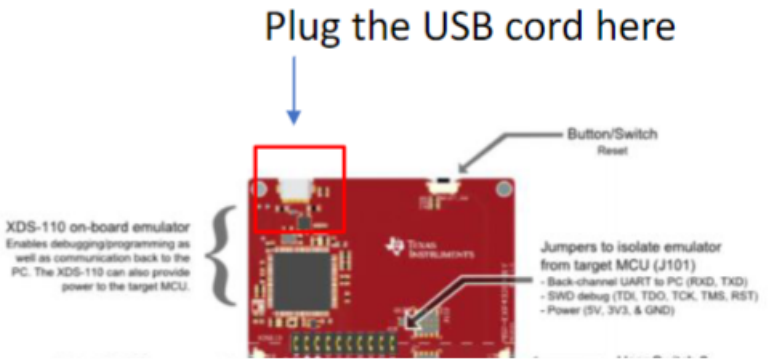
Steps to installing the MDK:

1. Install Keil MDK 5.36, check for updates of included software packs and update if necessary.
2. In this step you will install the most current Device Family Pack (DFP) for the MSP432E401Y. You will also copy the BlinkLED example to test the installation.
   (a) In Keil application, click "Pack Installer" button ( )
   (b) On the left panel, under the devices tab, search and select the device name "MSP432E401Y"
   (c) On the right panel, under packs tab, under "Device Specific" branch, install "TexasInstruments::MSP432E4_DFP" version "3.2.6"
   (d) On the right panel, under examples tab, choose `BlinkLED -> Copy`. This will copy the project from the reference to your local development directory and also give the option to start the Keil MDK.
3. Finally, the correct pronunciation of this application's name is "KYLE".

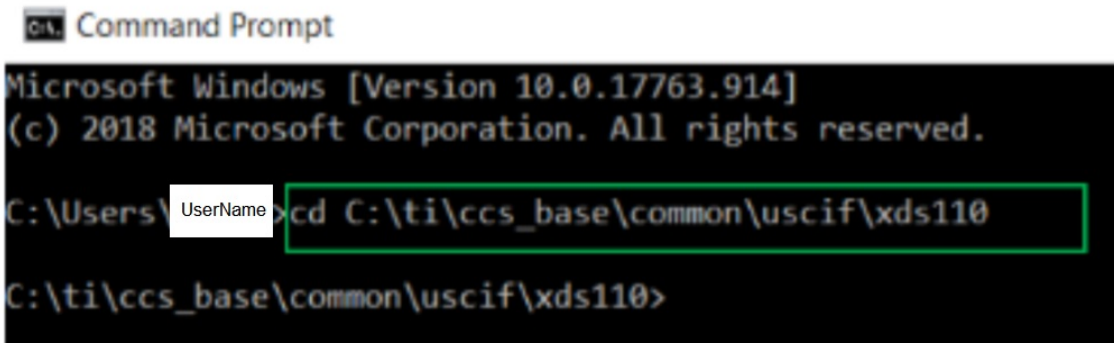
11.4.2 USB Debug Firmware

To connect the MDK to the MSP432E401Y we will need to update the debug firmware to use the onboard debugging hardware, this is the XDS110 – see `http://www.keil.com/appnotes/files/apnt_309_v1.0.pdf`.

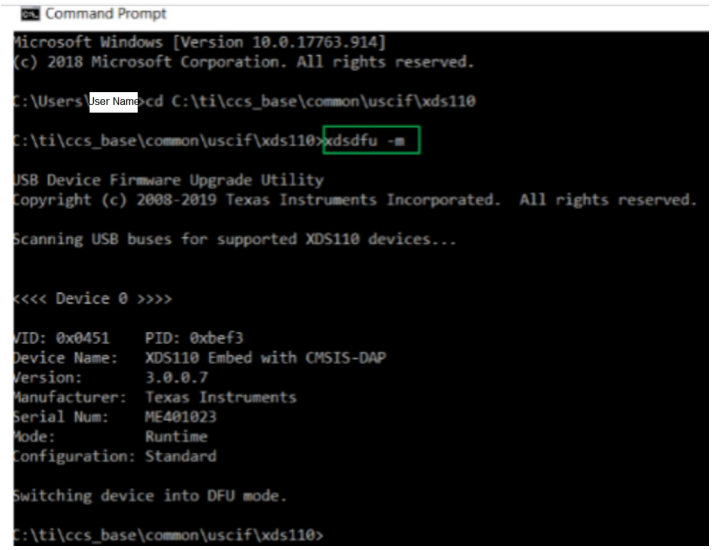
1. Visit `https://software-dl.ti.com/ccs/esd/documents/xdsdebugprobes/emu_xds_software_package_download.html` and download and install the 64-bit Windows Release 9.2.0.00002. This will extract the firmware update application.
2. Double check you have plugged in the MSP432E401Y into your computer (note there are two micro-USBs, you want to plug into the debugger at the top of the board).



3. Go into the directory where the debugger firmware is located by typing
   `d \verb C:\ti\csbase110` in Command Prompt (CMD)
   Note: Try `d \verb /d \verb C:\ti\csbase110` if `d alone does not work`



4. Type `dsdfu \verb -m (You may have to wait a bit)`



When the firmware update is complete, you may need to pause because after the firmware is updated, Windows sees a new USB device and as a result disconnects the “old" device TI board. If you are using a VM, you will now need to reconnect the board to your VM as demonstrated below:

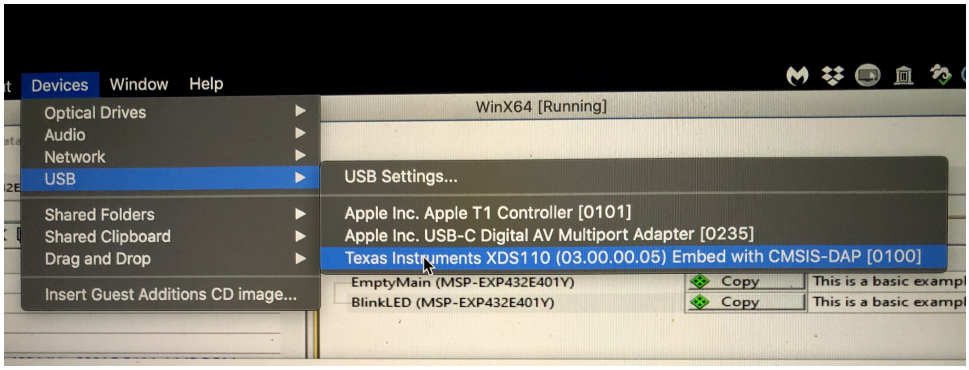
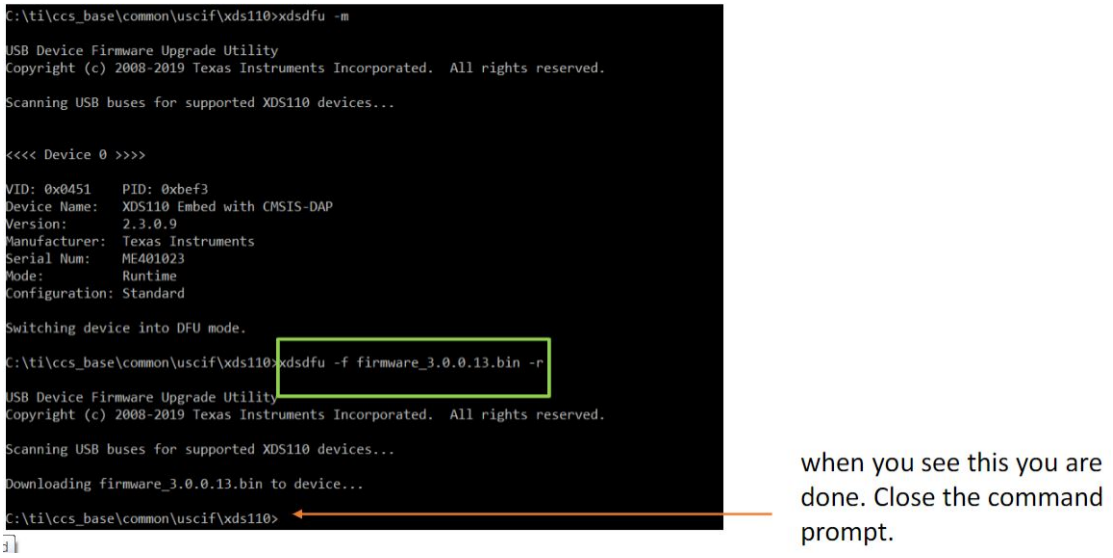


Figure 11.2: Re-Adding the MSP432E401Y’s XDS110 as a new device under VirtualBox

5. Type `dsdfu \verb -f  \verb firmware_3.0.0.13.bin  \verb -r`



11.4.3 Configure MDK Flash Tools

Select the Flash menu, then Configure Flash Tools (we are not using the ULink). We are using the MSP432E401Y's onboard XDS110, which requires the selection of CMSIS-DAP Debugger – see figure 11.3.

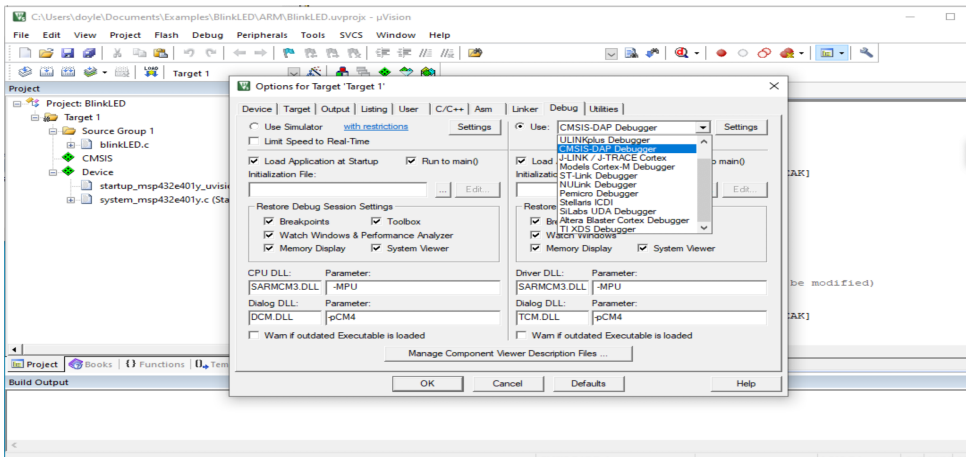


Figure 11.3: Configure Flash Tools configuration

Select Settings as shown in figure 11.4 and select OK.

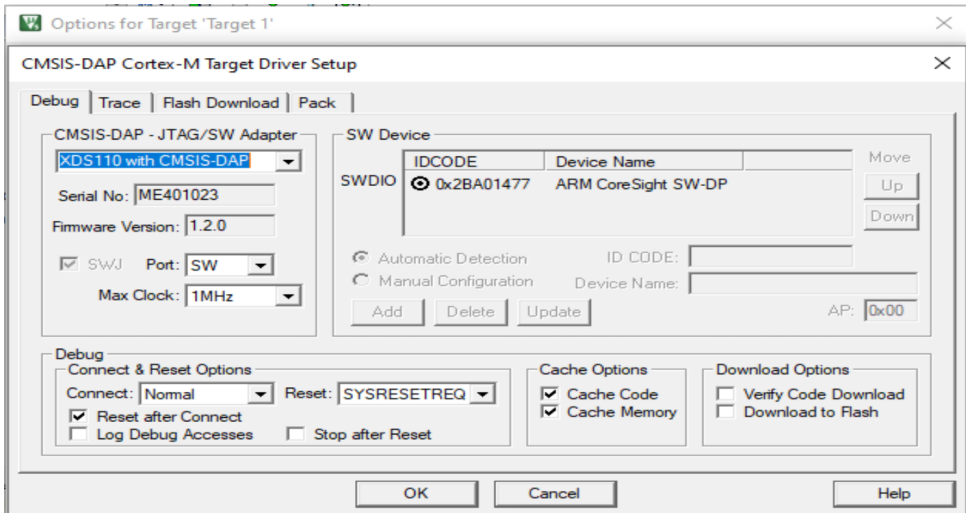


Figure 11.4: Configure Flash Tools settings

11.5 Say Hello

Return to Keil MDK and build the blinkLED.c code that you opened prior. Assuming no errors (there should be none), now select Load. The Build Output should indicate the Flash Load finished.

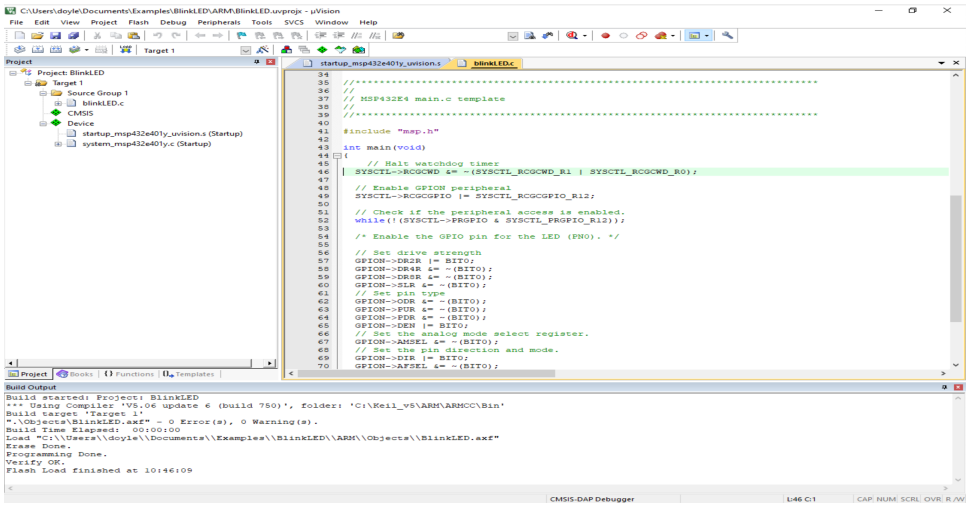


Figure 11.5: Successful build, load, and flash of blinkLED onto MSP432E401Y

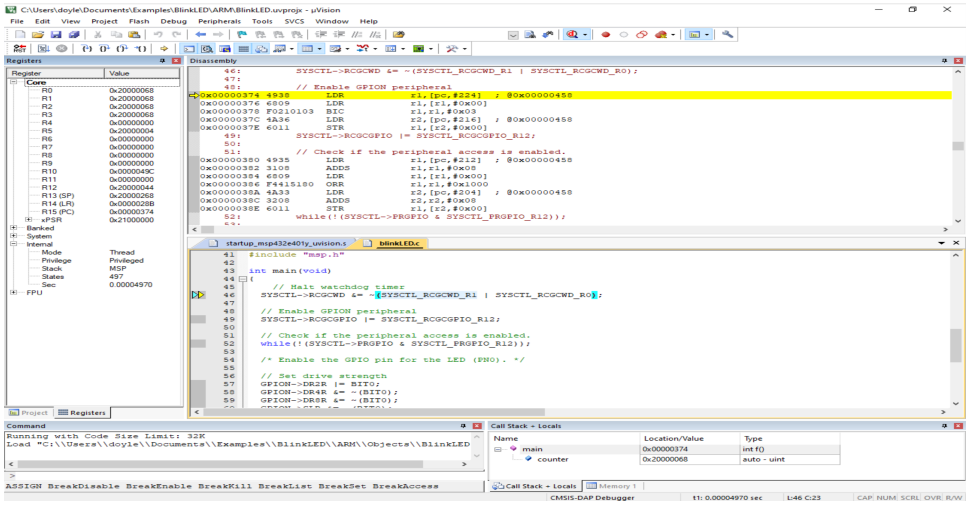


Figure 11.6: Keil MDK in hardware debug environment for blinkLED on MSP432E401Y

Press `Ctrl+F5` or select from the menu `Debug -> Start/Stop Debug Session`. The code can now be stepped/run and memory/registers inspected (`F5` to run).

Running the code should flash (a.k.a say Hello with) the onboard LED.

R Finally, to be consistent with the textbook we will modify J4 and J5 jumpers. Before modifying the jumpers, ensure the USB cable has been unplugged from the MSP432E401Y. Now, place J4 and J5 in the UART2 position so PA1 and PA0 are connected to the PC as a virtual com port. Reconnect the USB cable.

11.6 Software Version Summary

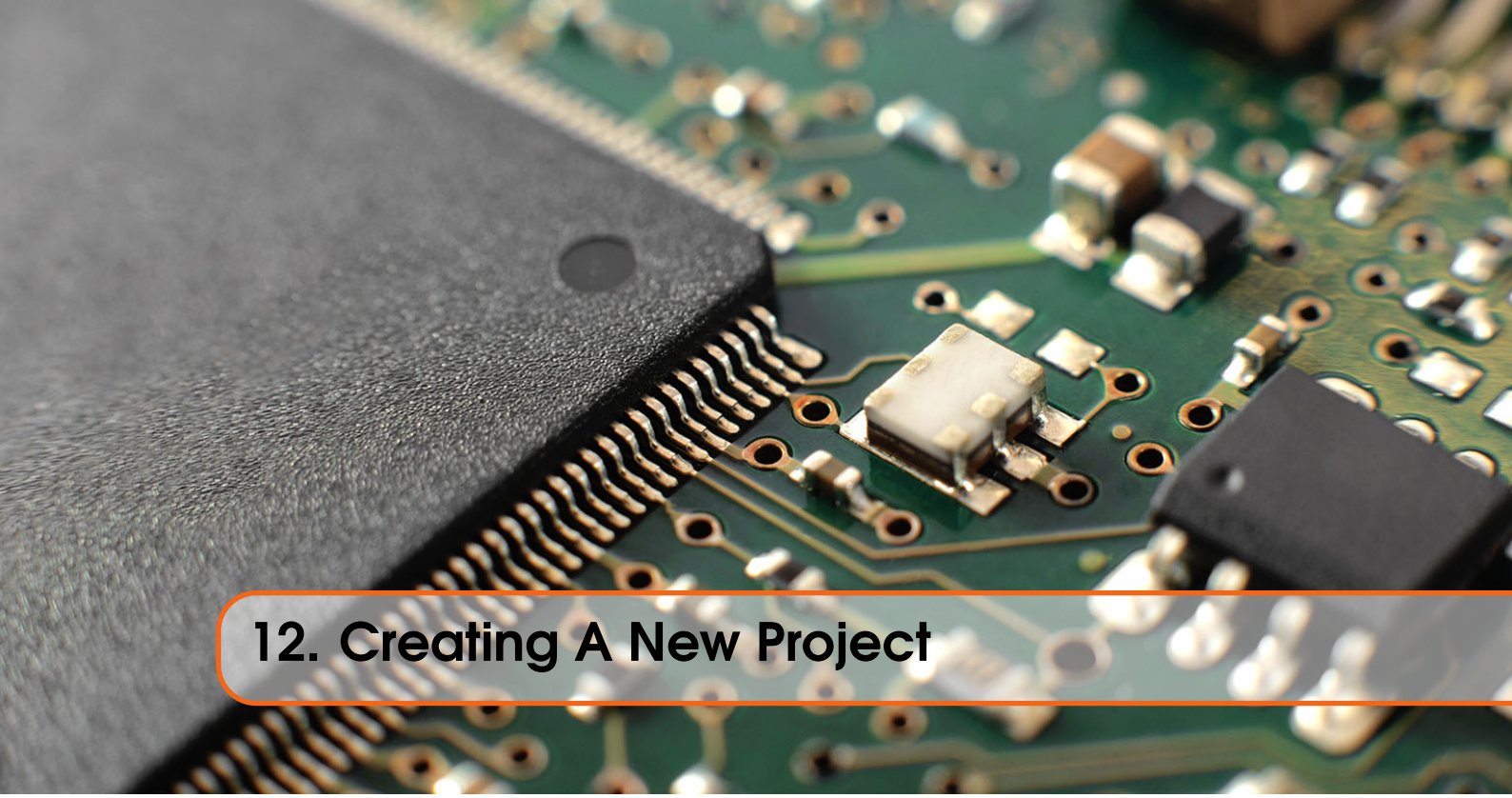
For these instructions we used the following software and version:

- Keil MDK 5.36 with `MSP432E4_DFP 3.3.6`
- macOS 10.15
- Microsoft Windows 7/10 Professional
- Oracle VM VirtualBox 6.0.14 (with Guest Additions installed after Windows)
- VMware Fusion 11.5 (with VMware Tools installed after Windows)
- MSP432E401Y firmware for XDS110 9.2.0.00002 (`ti_emupack_setup_9.2.0.00002_win_64.exe`)
- Keil sample project “blinkLED.c"

VII

Appendix B

12 Creating A New Project 89
12.1 Create a New Project from Scratch



12. Creating A New Project

Once you have confirmed your development environment is working, the next step is to create your own project.

12.1 Create a New Project from Scratch

12.1.1 Assembly Language

To create a new Keil project follow these steps:

1. Project > New uVision Project When prompts for target device select MSP432E401Y
2. In the next window go to CMSIS check core and close window
3. In the project window, click plus beside Target to reveal source group 1, right click source group 1,in the menu that opens click add existing files to source group 1
4. Add `Startup.s` and add `gpio.s` and close (in this example, `gpio.s` is your “main")
5. Build target
6. Flash to the board

The assembly files `Startup.s` and add `gpio.s` can be downloaded directly from `https://www.ece.mcmaster.ca/~doyle/2DX4_Code/L0/`.

A short video demonstrating the steps to create a new Keil project in assembly language can be found here: `https://youtu.be/wxs56Hwxxcw`.

12.1.2 C Language

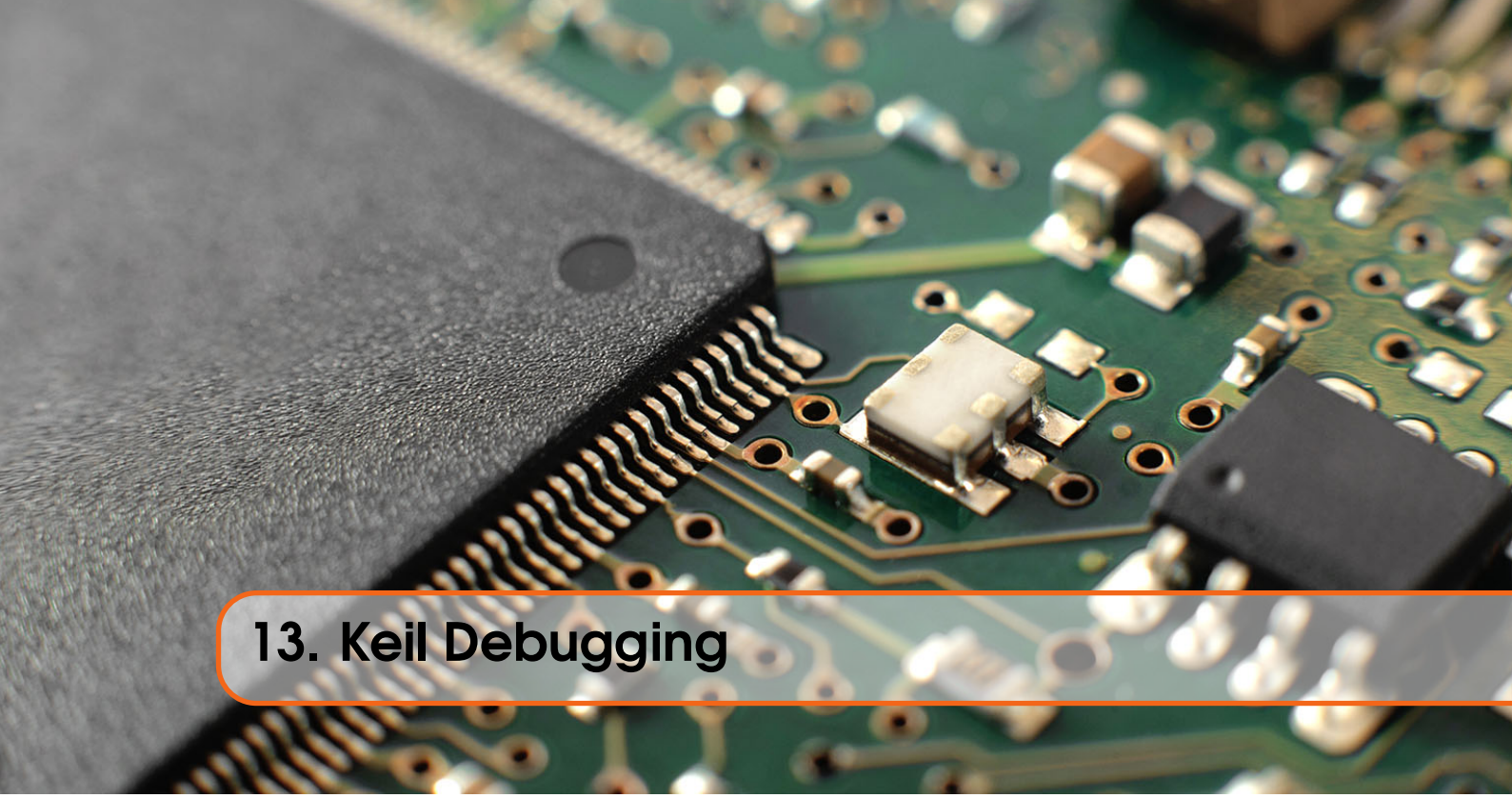
To describe the steps of creating a new Keil project in the C language, please follow the link below.

`https://youtu.be/ZLzT955KfGY`

VIII

Appendix C

13 Keil Debugging 93
13.1 Video Instruction
13.2 Exporting Data From the Keil IDE
13.3 Memory Value Capture Function



13. Keil Debugging

The Keil IDE offers a complete set of debugging tools. The following are steps for specific debugging tasks in 2DX3 labs.

13.1 Video Instruction

`https://www.youtube.com/watch?v=PgU3WirYYEU`

13.2 Exporting Data From the Keil IDE

One of the challenges when recording large amounts of data into microcontroller memory is verification. This relates directly to functional and performance dubugging.

The Keil Debug IDE offers numerous tools for inspection and the ability to customize inspection through setting Watches on variable and data structures, but also by using the Command Line. The following steps will allow you to export debug data:

1. Connect the microcontroller to computer
2. Load project
3. Configure Options for Target (proper debug hardware configuration)
4. Ensure the variable you wish to inspect is going to be in memory upon completion of your program. The easiest way to do this is to define the variable as global scope.
5. Translate, Build, Load.
6. Start Debug Session.
7. Before pressing Run, set Watch for variable(s) of interest. To do this, using your mouse hover over the variable in the source code and right click, then select “Add [variable] to...”, then choose Watch 1. If this is an array and you want inspect all elements then modify the Watch 1 window to remove the [index]. Press + to see all array elements.
8. At the top, select View menu and then select Periodic Window Update to have the IDE update output windows (such as Watch).

9. Run.
10. Upon conclusion you will see your variable(s) of interest in the Watch window for inspection.
11. To export the variable(s) we can look into the Command window operations, like LOG, EVAL, D, and FUNC to allow easy inspection and capture of the data. Try the following:

```
Command Line:              Comment:
LOG > demo.txt             Starts a log file called demo.txt
EVAL func_debug           Evaluates variable - in this case this is a
                               pointer so just provides address of index [0]
EVAL func_debug[0]         Presents contents as hex and decimal
EVAL func_debug[1]            ... helpful but not for large quantity of data
D func_debug[0],func_debug[99]  Displays memory contents which are
                                     stored little endian
LOG OFF                    Ends a log file
```

12. These are helpful, but we can make better use of the debug capabilities. Looking at FUNC and .ini files we can more easily extract meaningful data. FUNC will permit us to write ANSI C style functions and .ini files permit us to save as a script. To write a script, at the top select the Debug menu, and then select Function Editor. An example is shown below of a FUNC that logs 100 array elements.

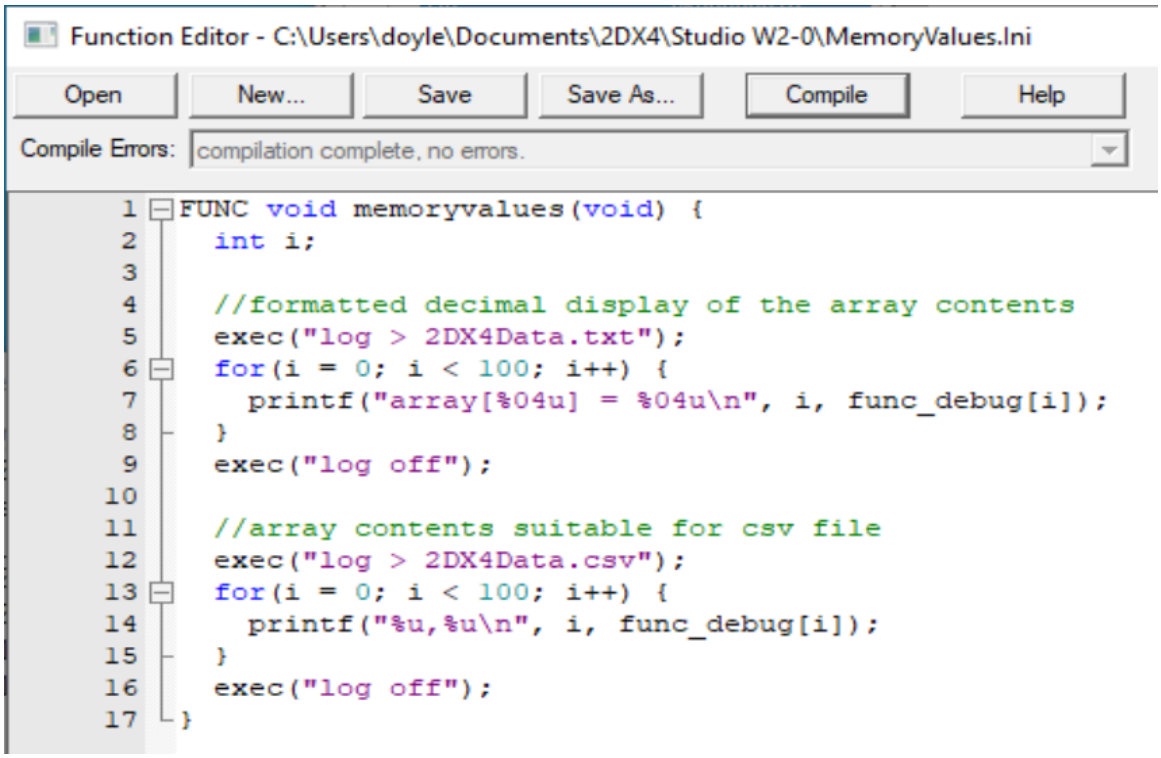


Figure 13.1: Sample .ini script illustrating ANSI C style function for Keil debugging

13. Once the program is complete and compiled, the function can be invoked in the command line:

```
memoryvalues()
```

For more information, refer to the Keil Command Line resource here: http://www.keil.com/support/man/docs/uv4/uv4_debug_commands.htm.

More details on the commands listed above:

D Display: http://www.keil.com/support/man/docs/uv4/uv4_cm_display.htm
LOG LOG: http://www.keil.com/support/man/docs/uv4/uv4_cm_log.htm
EVAL EVALuate: http://www.keil.com/support/man/docs/uv4/uv4_cm_evaluate.htm
FUNC FUNC: http://www.keil.com/support/man/docs/uv4/uv4_cm_func.htm

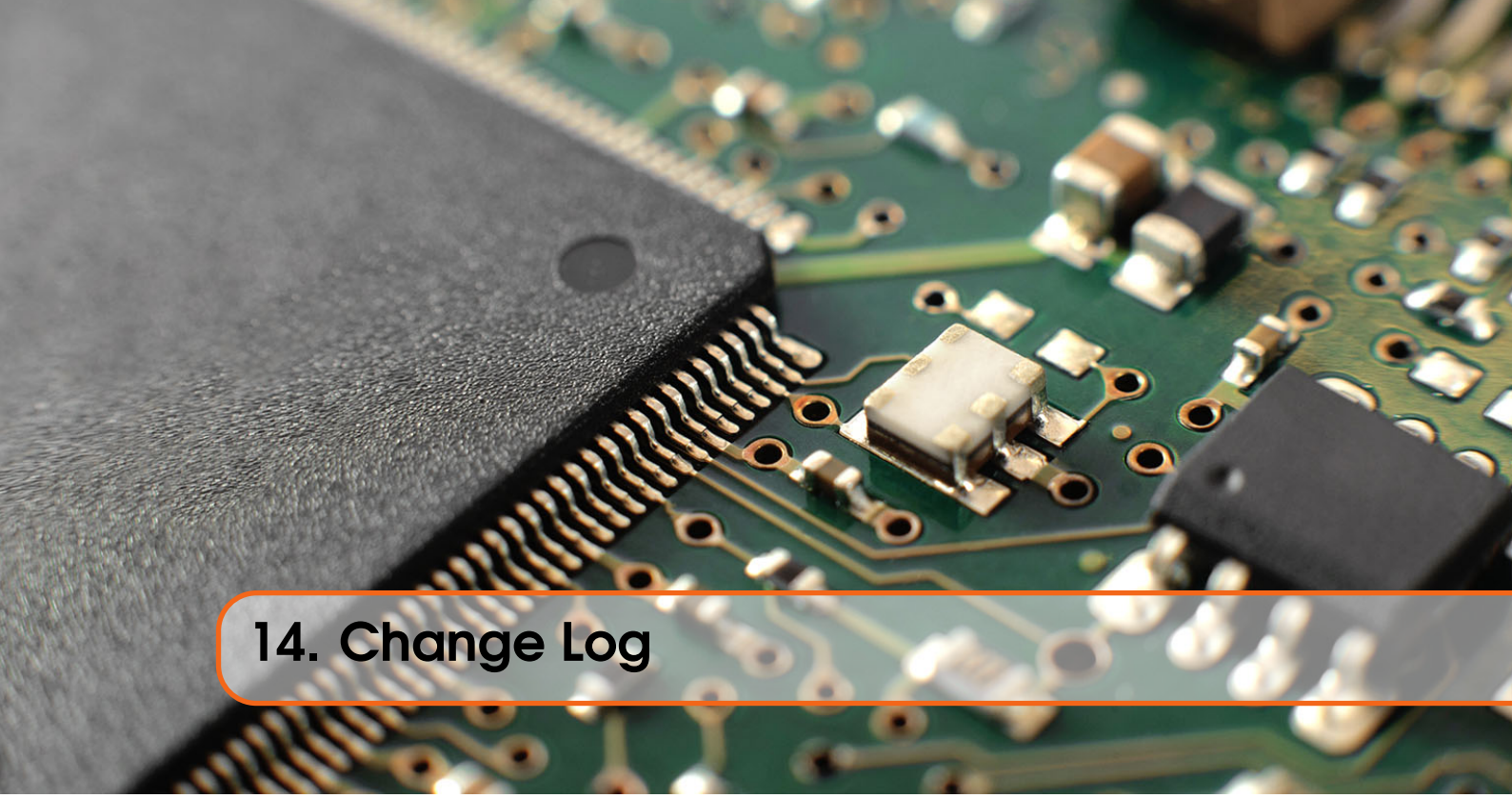
13.3 Memory Value Capture Function

```
FUNC void memoryvalues(void) {

    int i;

    //formatted decimal display of the array contents
    exec("log > 2DX4Data.txt");
    for(i = 0; i < 100; i++) {
        printf("array[%04u] = %04u\n", i, func_debug[i]);
    }
    exec("log off");

    //array contents suitable for csv file
    exec("log > 2DX4Data.csv");

    for(i = 0; i < 100; i++) {
        printf("%u,%u\n", i, func_debug[i]);
    }
    exec("log off");
}
```

IX

Appendix D

14 Change Log 99



14. Change Log

This appendix will record the changes made to this document in chronological order.

January 4, 2024 Lab manual issued with Labs 0-8.