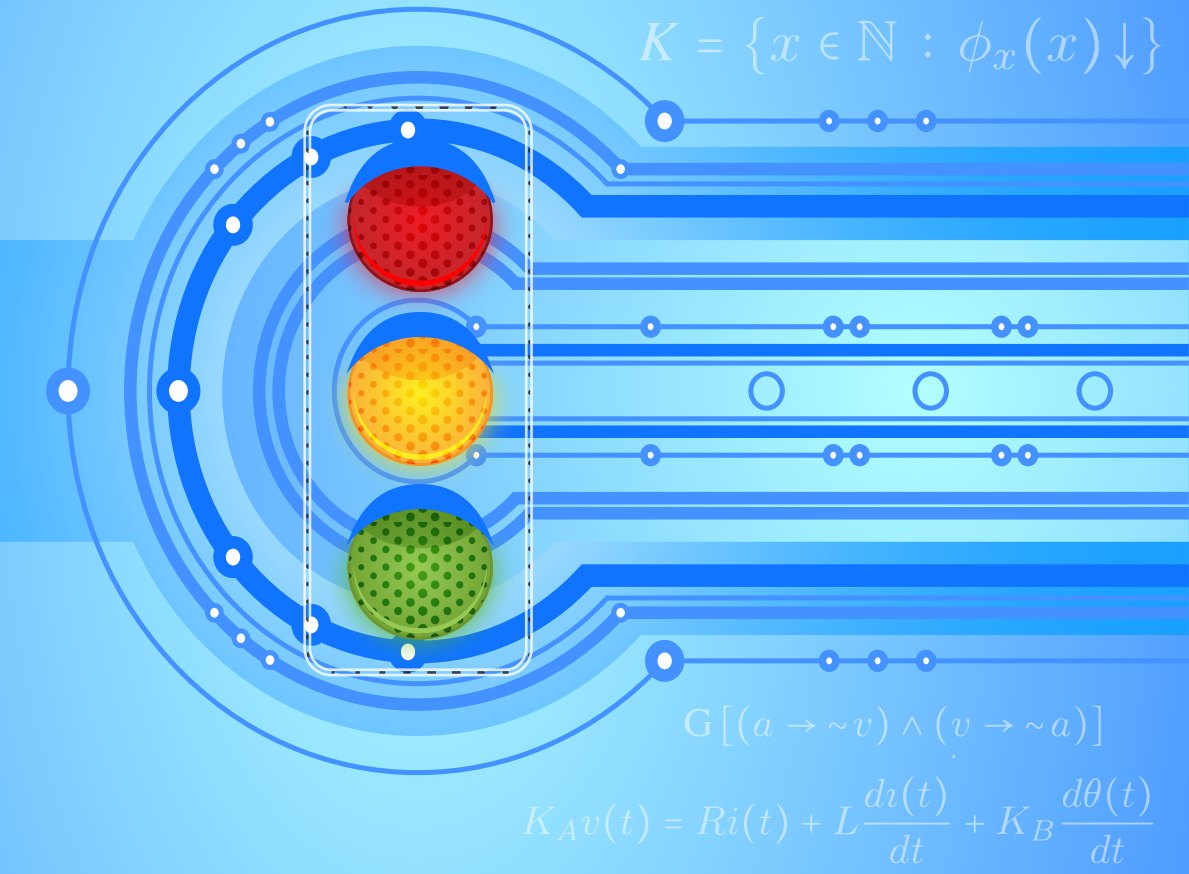
An Introductory Lab in Embedded and Cyber-Physical Systems



Jeff Cameron Jensen
Edward Ashford Lee
Sanjit Arunkumar Seshia

First Edition, Version 1.70

Please cite this book as:
J.C. Jensen, E.A. Lee, and S.A. Seshia,
An Introductory Lab in Embedded and Cyber-Physical Systems v.1.70,
http://leeseshia.org/lab, 2014.

Additional key contributors: Eric S. Kim, Trung Tran, and Matthew Weber.

This edition links to additional documentation that should accompany this PDF.

Contents

Preface

What this Book is About

The theme of this book is the exploration of embedded and **cyber-physical systems** not by resource constraints, but instead by their interactions with the physical world. While resource constraints are an important aspect of design, such constraints are part of every engineering discipline and give little insight into the interplay between computation and physical dynamics. We emphasize the basics of models, analysis tools, and design of embedded and cyber-physical systems. We guide modeling of the physical world with continuous-time differential equations and modeling of computations using logic and discrete models such as state machines. These modeling techniques are evaluated through the use of meta-modeling, illuminating the interplay of practical design with formal models of systems that incorporate both physical dynamics and computation. We introduce formal techniques to specify and verify desired behavior. A combination of structured labs and design projects solidifies these concepts when applied to the design of embedded and cyber-physical systems with real-time and concurrent behaviors.

Laboratory exercises expose the lowest levels of abstraction for programming embedded systems such as traditional imperative programming models through to the highest levels of abstraction such as graphical system design and concurrent models of computation. At the highest level, we use LabVIEW to introduce students to model-based design. One level

down from that, we use the C programming language and an RTOS. One level down from that, we use bare-metal C on a microprocessor – software that executes in the absence of an operating system.

Intended Audience

Most undergraduate students in the fields of computer science, electrical engineering, and computer engineering will take at least one course in embedded systems. We intend for this text to be a valuable addition to their junior or senior year. The topic of cyber-physical systems extends to undergraduate and graduate students in mechanical engineering who study robotics, mechatronics, or microprocessor-based systems. Many of the laboratory experiments and projects are open-ended by nature and lend to greater exploration of design methodology, an appropriate topic for graduate students in any field relating to embedded or cyber-physical systems.

This text draws from topics in physics, circuits, transducers, computer architecture, digital signal processing, digital communications, networking, operating systems, robotics, control theory, algorithms, probability, and logic – an indefensible list of prerequisites for any course, and a list that illustrates the interdisciplinary nature of embedded and cyber-physical systems. Each of these topics yields its own vast field of study far beyond the scope of this text. We instead touch on key fundamental concepts and exposure to design at multiple levels of abstraction. We suggest students first take an introductory course in signals and systems, an introductory course in computer architecture (which covers both C and assembly programming), several courses of elementary continuous calculus, and an introductory course in discrete mathematics. While these prerequisites establish a common language for the technical aspects of embedded and cyber-physical systems, the ubiquitous and interdisciplinary nature of these systems compels students to investigate topics beyond computer science.

The laboratory exercises in this book were designed and piloted for the course EECS 149, "Introduction to Embedded Systems" at the University of California, Berkeley (Lee and Seshia, 2010; Jensen et al., 2011; Lee et al., 2013; Jensen et al., 2012). The course is targeted at undergraduate juniors and seniors in Electrical Engineering and Computer Science. Students whom we have taught from this material are generally interested and engaged, celebrating their projects as well as those from other teams. They are often proud of what they accomplish, and even post project presentation videos to the internet. Such anecdotes give some insight into the impact of a course, but how do we know

for sure whether a particular change in the course or laboratory design is actually an improvement? We are pleased, at least, to witness that students surprise both themselves and their instructors, that projects demonstrate an understanding of the theoretical concepts introduced in lecture, and that students have received job offers from industry mentors.

How to Use this Book

Laboratory exercises in this text are designed to be modular so that they may be chosen according to the topics that best suit a course. Dependencies between labs are explicitly stated, and generally indicate two or more labs fit into a sequence. The National Instruments myRIO embedded controller is used for most laboratory exercises and is introduced in Chapter 1: Equipment. Other software and hardware tools are discussed in the equipment section of each chapter. Initial setup of laboratory workstations and equipment is covered in Appendix A: Lab Setup.

Each laboratory consists of a pre-laboratory assignment and in-laboratory exercises. The pre-laboratory exercises are *critical* preparation in advance of in-laboratory exercises, as they introduce documentation, tools, and concepts used. We suggest formal laboratory sessions begin with a brief lecture that covers the instruments used, their theory of operation, and the overall goal of the laboratory. Teams of two or three may then begin working on assigned laboratory exercises; in many cases, solutions are not unique, and we encourage teams to experiment and innovate.

The electronic version of this book includes a number of hyperlinks to documents that are needed or helpful for completing the labs. These documents are provided as downloads at http://LeeSeshia.org/lab. They are also cited in the references section at the end of the book, where URLs to the original document are provided.

The theoretical foundation for this text follows Introduction to Embedded Systems (Lee and Seshia, 2015).

Acknowledgements

Many people have contributed to the design and vetting of the exercises in this book. Trung N. Tran of National Instruments authored a significant portion of the software framework for myRIO, including custom VHDL, MicroBlaze architecture, C code, and

LabVIEW code. Trung has mentored a number of student design groups and has provided invaluable instructional support for laboratories. Rajesh Gupta and Seemanta Dutta at the University of California San Diego were instrumental in testing and refining labs. Godpheray Pan from National Instruments customized a physics simulator and renderer for a complete model-based design workflow using the iRobot Create. Andy Chang, also from National Instruments, made significant contributions towards the first massive online open course (MOOC) offering of this curriculum. Additional key contributions were made by the Engineering Support Group in the Electrical Engineering and Computer Science Department at the University of California Berkeley, especially Ferenc Kovac and Winthrop Williams, and the Instructional Support Group in the same department, especially Linda Huang and Kevin Mullally. We also thank the graduate student instructors who supervised offerings of the course and helped to develop and debug these lab exercises, particularly Hugo Andrade, Dai Bui, Alexander Donze, Shanna-Shaye Forbes, Garvit Juniwal, Eric Kim, Hokeun Kim, Ben Lickly, Wenchao Li, Isaac Liu, Zach Wasson, Matt Weber, Kevin Weekly, and Michael Zimmer. In particular, Eric, Matt, and Trung have contributed significantly to the new material on the Kobuki platform and associated updates to the lab content. Cover design by D. Fred Duran.

Laboratory Exercise Dependency Graph

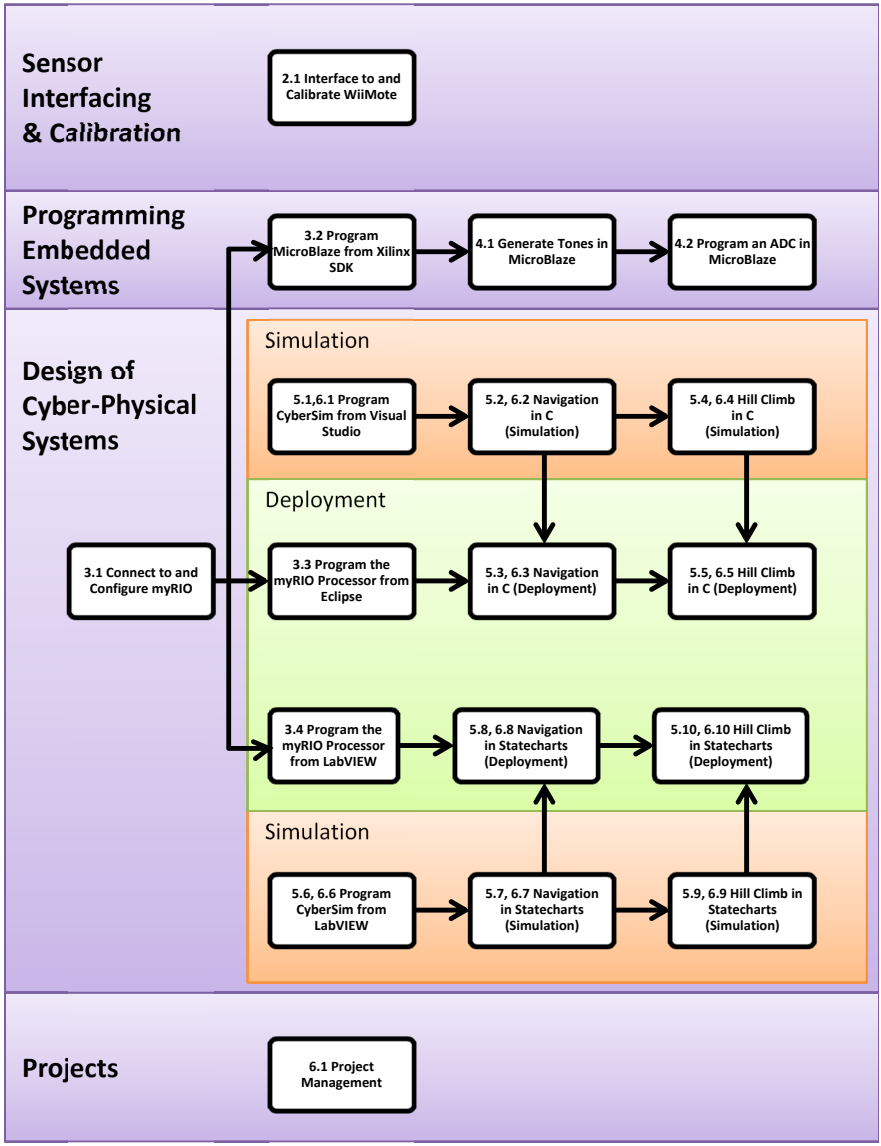


Figure 1: Laboratory exercise dependency graph.

1 Equipment

This chapter contains information about equipment used throughout laboratory exercises. Laboratory exercises begin in the next chapter.

General-purpose software, such as software that runs on a desktop computer, is typically developed on the same system in which it will execute. This enables fairly straightforward debugging as debugging tools, source code, and the executable all reside in the same system. Embedded software, however, is typically developed on a desktop computer, sent to **cross-compile** (to compile for a non-native architecture), and downloaded to an embedded system before it executes.

For design productivity, developers utilize an **integrated development environment** (**IDE**), a graphical user interface that organizes projects, hardware targets, development tools, and debuggers. An IDE may be provided by a hardware vendor, or development tools may be tied to an open-source IDE.

1.1 Equipment: WiiMote

1.1.1 Introduction to WiiMote

The Nintendo Wii Remote (Fig. 1.1), known informally as a **WiiMote**, is a handheld wireless game controller with motion sensing capability. It is an embedded system composed of sensors, actuators, a wireless radio, and a microcontroller. Its sensors consist of a three-axis analog accelerometer, an infrared camera, and momentary buttons; its actuators consist of a speaker, a DC motor (for "rumble" functionality), and LEDs. The WiiMote is a simple, popular, and widely available embedded system, and its wireless interface allows it to interact with a desktop computer.

The Bluetooth **Human Interface Device** (**HID**) profile defines a communication protocol used by the WiiMote. Each command begins with a 1-byte Transaction Header, the first nibble of which corresponds to the Transaction Type, and the second to a Parameter. The header is followed by a device-specific Report ID, and sometimes a payload. Documentation for WiiMote Report IDs has not been published, but some Report IDs have been identified and documented in WiiMote↗ (WiiBrew, 2012).

Bluetooth pairing becomes difficult when multiple devices are present, as may be the case in a laboratory. This issue is resolved by pairing according to the hardware address of the specific device to be paired. As with all networked devices, the WiiMote Bluetooth interface is identified by a unique **media access control** (**MAC**) address. A MAC address is a sequence of 6 bytes, usually written something like "00:1E:35:3B:7E:6D" using hexadecimal notation. Unfortunately, the MAC address is not visible on the WiiMote itself; it can be found by manually pairing the WiiMote using the standard Windows Bluetooth device paring. Once paired, the device properties will show the MAC address. The MAC address can then be printed on the side of the device for future reference.

1.1.2 WiiMote Useful References

Wii Remote (Wikipedia, 2013d)

WiiMote (WiiBrew, 2012)

Nintendo Controllers (Nintendo, 2013)

Bluetooth HID Profile (Bluetooth Special Interest Group, 2003)

Figure 1.1: Nintendo Wii Remote (Orlando, 2010).

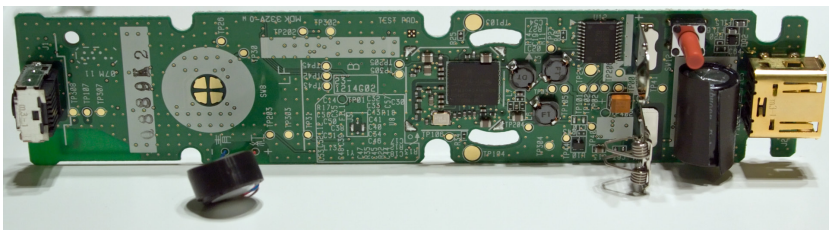


Figure 1.2: Nintendo Wii Remote exposed (Stoops, 2008).

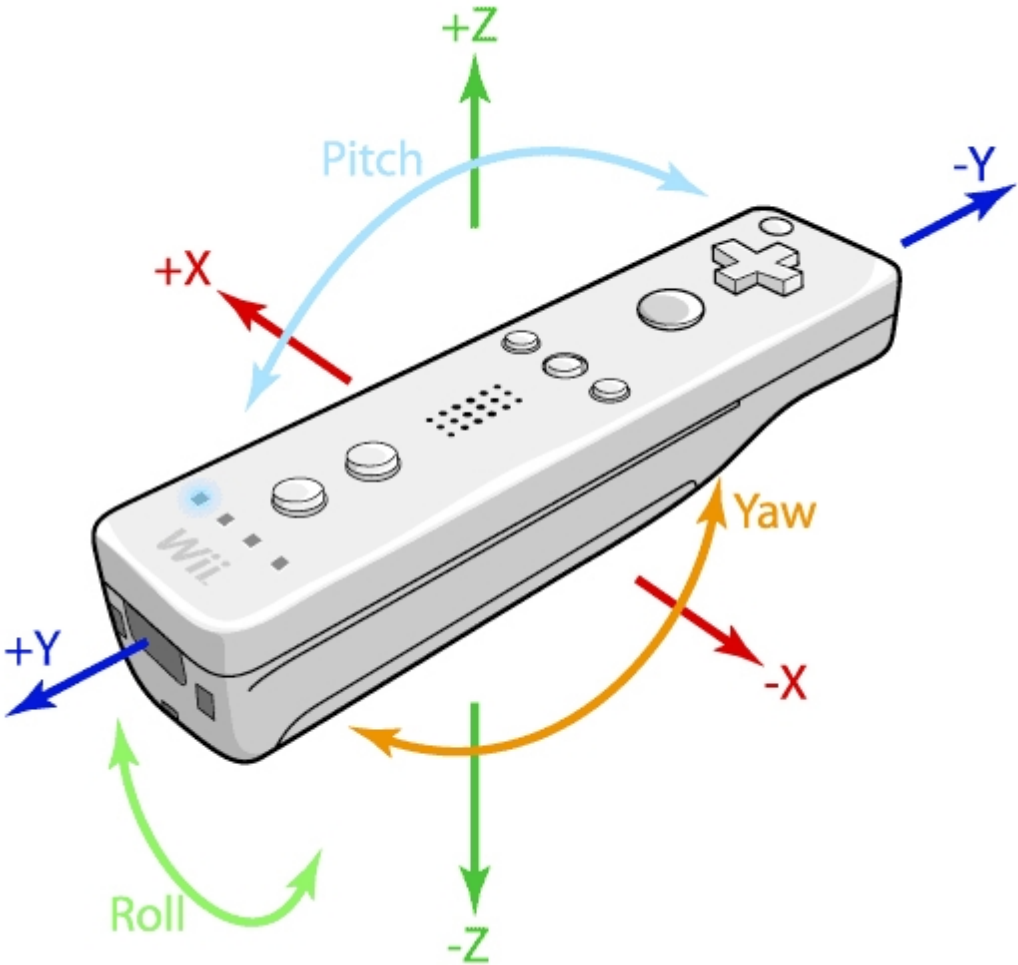


Figure 1.3: Nintendo Wii Remote accelerometer axes (Stoops, 2008).

1.2 Equipment: myRIO

1.2.1 Introduction to myRIO

The National Instruments **myRIO** (Fig. 1.4) is an embedded microcontroller with the Xilinx Zynq Z-7010 reconfigurable multiprocessor architecture (Fig. 1.5). Its two processors have their own memory and peripherals and may be programmed independently.

The fixed processor is an ARM Cortex-A9 MPCore, a dual-core processor that implements the ARMv7 **instruction set architecture** (**ISA**) and includes a fixed set of peripherals. The ARM Cortex-A9 on myRIO is preconfigured at the factory with a distribution of Linux with real-time extensions. Linux with real-time extensions is a popular **real-time operating system** (**RTOS**), an operating system with more deterministic scheduling and behavior, and used in a wide range of embedded applications. C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition is used to compile, download, execute, and debug C applications on myRIO and LabVIEW is used to compile, download, execute and debug Structured Dataflow applications on myRIO.

The reconfigurable processor on myRIO is the Xilinx Artix-7 **field-programmable gate array** (**FPGA**). An FPGA is comprised of logic units, memory, and other fundamental building blocks that may be reconfigured *at the hardware level*. An FPGA can implement hardware peripherals such as communication buses, PWM generators, quadrature encoder interfaces, signal processing algorithms, video rendering and decoding, and even other processor architectures.

An **FPGA fixed-personality** is a configuration of an FPGA that is meant to be distributed without modification. A fixed-personality is a compiled binary file known as a **bitfile** that can be distributed without source code. A multiplicity of fixed personalities can be used to rapidly reconfigure an FPGA, allowing a single hardware device to serve a broad range of applications. The FPGA fixed-personality that ships with myRIO implements a number of peripherals, such as UART, I2C, PWM, and others, over standard digital inputs and outputs. The FPGA fixed-personality is programmed in LabVIEW FPGA and released as open-source. LabVIEW FPGA can generate C code which can be imported into Eclipse to interface with the FPGA.

National Instruments **myRIO Getting Started Wizard** is a tool to connect to myRIO for the first time. It verifies connection to the hardware, displays data from onboard sensors, and links to examples and tutorials.

National Instruments **Measurement and Automation Explorer** (**MAX**) scans to discover National Instruments hardware including myRIO. MAX has built-in testing and configuration functions and is the best tool to configure the software on myRIO. MAX is launched from the Windows desktop or from the Windows Start menu under "National Instruments→NI MAX".

National Instruments **Web-Based Configuration and Monitoring** is a web-based configuration that runs on embedded targets including myRIO. Use a standard browser and navigate to the IP address of your target to launch NI Web-Based Configuration and Monitoring. The configuration is similar to that available through MAX.

1.2.2 myRIO Useful References

myRIO 1950 User Guide and Specifications (National Instruments, 2013g)

myRIO Shipping Personality Reference (National Instruments, 2013l)

myRIO Product Page (National Instruments, 2013k)

myRIO Getting Started (National Instruments, 2013i)

myRIO Community (National Instruments, 2013h)

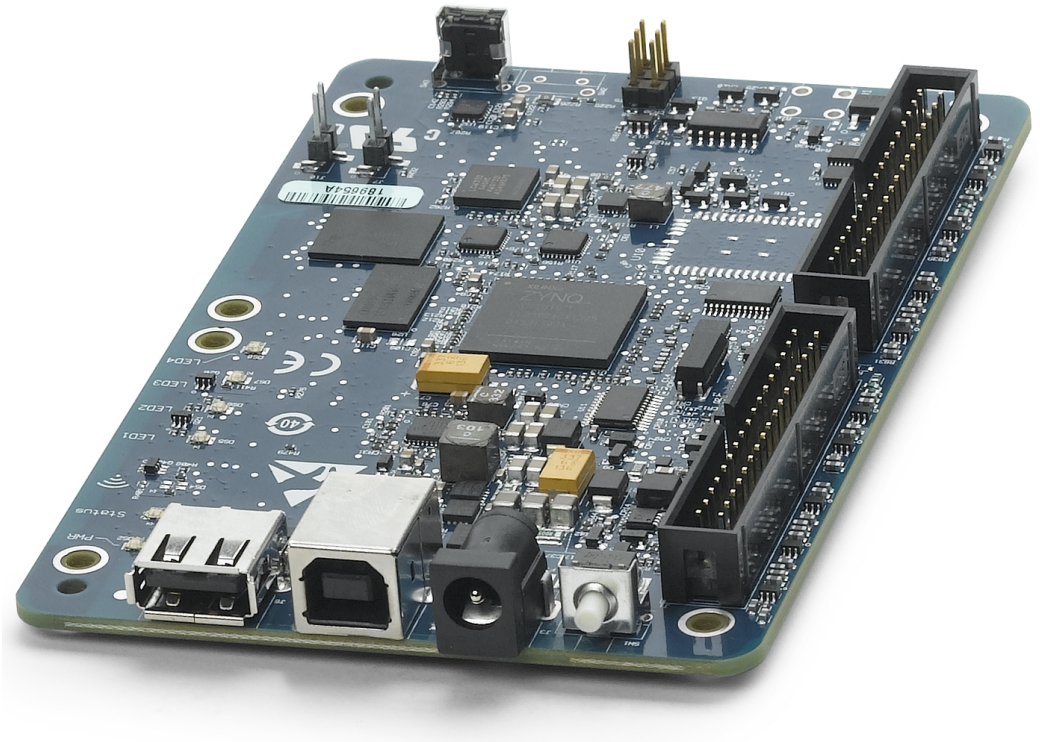


Figure 1.4: National Instruments myRIO 1950 embedded controller.

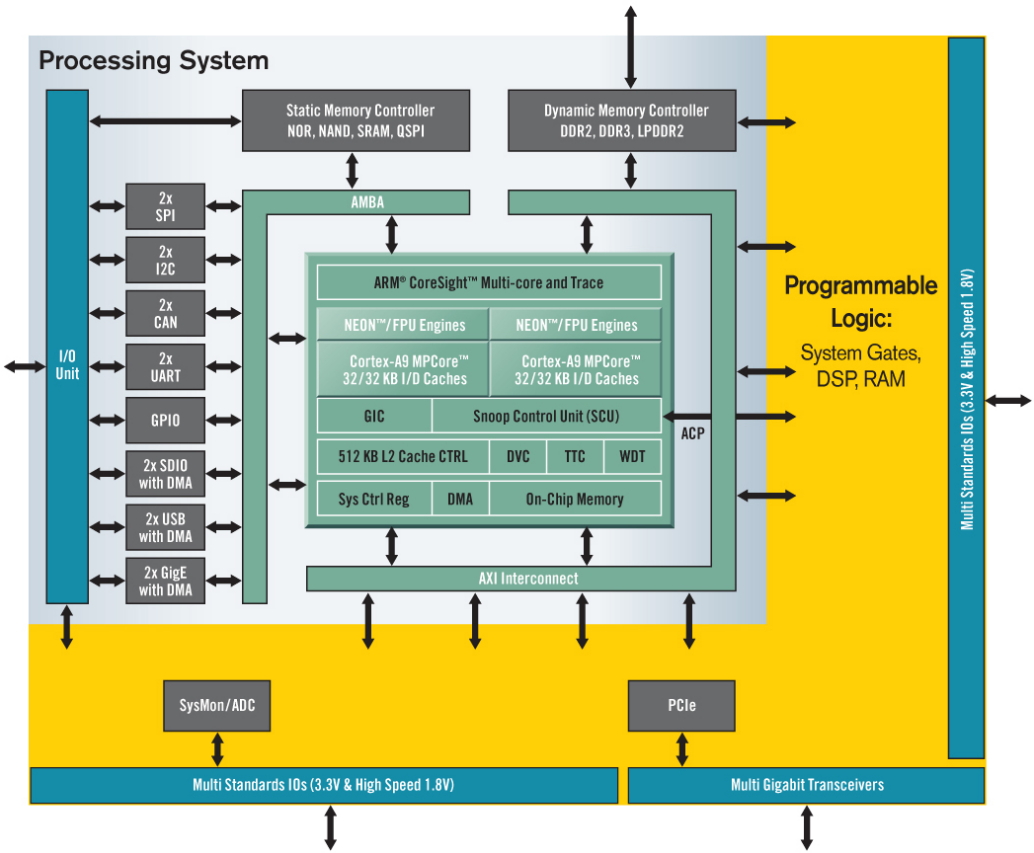


Figure 1.5: Xilinx Zynq Architecture.

1.3 Equipment: MicroBlaze

1.3.1 Introduction to MicroBlaze

FPGAs can be configured to implement an ISA and other microcontroller components that can be easily modified with new peripherals and functionality. A microprocessor implemented in an FPGA is known as a **soft-core processor**. Soft-core processors are usually slower than a custom silicon processor with equivalent functionality, but they are nonetheless attractive because they are fully customizable. Often, multiple soft-core processors may be fit onto a single FPGA to create a multicore processor.

Xilinx **MicroBlaze** is a 32-bit soft-core processor with a proprietary ISA. MicroBlaze can be programmed on an FPGA using Xilinx design tools or by deploying an FPGA fixed-personality that contains a MicroBlaze core. The MicroBlaze core in the fixed-personality used for these labs runs at a clock rate of 50MHz and has 128kB of memory. There is no floating-point processor or cache. You may use the user button on myRIO to reset the MicroBlaze processor.

Developers need a way to debug embedded applications in a similar fashion to desktop debugging but with greater insight into the processor architecture, memory, subsystems, and peripherals. A predominant solution is a standard for specialized hardware interfaces and a communication protocol. The **Joint Test Action Group** (**JTAG**) debugging standard is often implemented by special hardware on an embedded controller and interfaced to a desktop computer using a JTAG interface device. JTAG defines communication at the physical layer, and vendors often define additional protocols for downloading, programming, and debugging.

The **Xilinx Software Development Kit** (**Xilinx SDK**) is an Eclipse IDE for developing embedded applications. Together with a JTAG interface device, Xilinx SDK is used to program and debug software running on a MicroBlaze soft-core processor.

1.3.2 Microblaze and SDK Useful References

MicroBlaze Processor Reference Guide (Xilinx, 2012c)

MicroBlaze Soft Processor Core (Xilinx, 2013c)

LogiCORE IP AXI INTC (Xilinx, 2013b)

LogiCORE IP AXI Timer (Xilinx, 2012b)

Embedded System Tools Reference Manual (Xilinx, 2012a)

EDK support (Xilinx, 2013a)

Joint Test Action Group (Wikipedia, 2013a)

myRIO JTAG Instructions (National Instruments, 2013j)

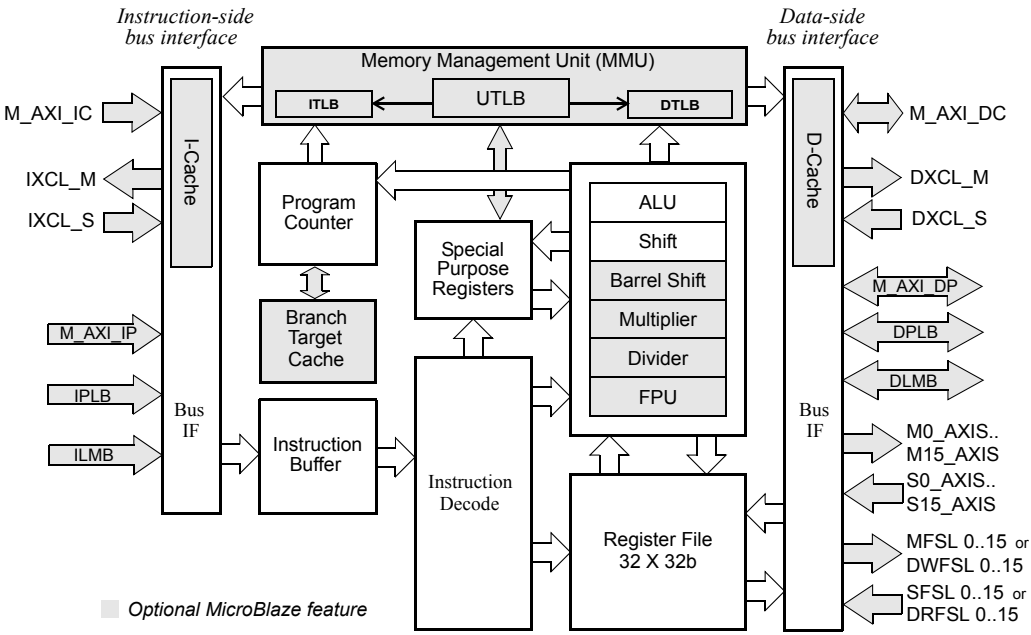


Figure 1.6: Diagram of the Xilinx MicroBlaze processor (Xilinx, 2012c).

1.3.3 Interrupts on MicroBlaze

Interrupts can be used to periodically sample sensors such as microphones for voice control or ultrasonic range finders for navigation. They can be used to drive actuators by generating a pulse-width modulation (PWM) signal for sound or motor control, or to simply flash an LED to signal a device fault. Software written for an embedded system may even reside almost entirely in timed interrupts.

An **interrupt service routine** (**ISR**) should contain a minimum number of instructions. An ISR that is invoked periodically and has an execution time nearly as long as its period will starve other processes; worse, an ISR whose execution time is *equal to or greater* than its period may cause unpredictable or unstable behavior. Interrupts step outside the paradigm of sequential code, and care must be taken to ensure no two processes attempt to simultaneously access shared resources. It is important to document any shared resources used by an ISR, including whether or not the ISR will read or write to a resource.

It is difficult to debug interrupts since their behavior may be intimately tied to the timing of the system and the occurrence of external events. A common mistake is to use a costly debugging technique like printing to a console or communication port in the body of an ISR; such operations are processor-intensive and access shared hardware resources, with the potential of contributing significant timing and logical artifacts. Any debugging instructions in an ISR will affect timing, but there are reasonably safe debugging mechanisms that have minimal impact such as toggling a digital output line or incrementing a counter, each of which may be executed in a single processor cycle.

1.3.4 MicroBlaze and AXI Bus Architecture

When programming at the register level to configure interrupts or **digital input and output** (**DIO**), pay close attention to memory locations and bit numbering. MicroBlaze can be configured for either big-endian or little-endian memory addressing; big-endian is used in this text and with the distributed MicroBlaze fixed-personality. The **Advanced eXtensible Interface** (**AXI**) bus that connects MicroBlaze to hardware peripherals uses standard bit ordering for its register architecture, shown in Fig. 1.7[1].

[1] The MicroBlaze architecture uses *bit-reversed ordering* where bit 0 is the most significant bit and bit 31 is the least significant bit. The lab exercises here do not require you to program any MicroBlaze register, but only the standard bit ordered AXI bus registers.

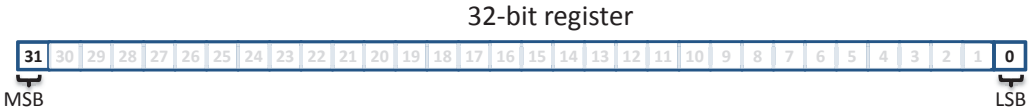


Figure 1.7: Bit numbering for an AXI bus register. The LSB is isolated by the mask `0x0000 0001`, and the MSB is isolated by the mask `0x8000 0000`.

A common practice in documenting and interfacing with memory-mapped registers is to name a register in documentation and provide reference source code that maps this name to a memory location. For example, documentation labels a register as DIOB_70OUT, a concise (and rather cryptic) name for a memory-mapped register whose value drives digital outputs on physical pins 0 through 7; example C code provided by the hardware vendor defines this name as follows:

```
#define DIOB_70OUT *((volatile int *)(XPAR_PORTDO_BASEADDR))
```

Figure 1.8: Example mapping of a documented memory-mapped register to a C macro of the same name. C statements may reference this macro as if it were a symbol, reading or assigning its value as if it were a variable (though unlike a C variable, this mechanism introduces subtle implications and dynamics). `XPAR_PORTDO_BASEADDR` refers to the memory location of the digital I/O port memory-mapped register; subsequent pins are offset from this address.

A word of caution about a **read-after-write hazard** (**RAW hazard**): depending on the hardware architecture you are using, some memory-mapped registers take a few cycles to update with new values. If a register is written and then immediately read, it's value may not have settled, causing unexpected results. It is good practice to insert `asm("nop")` instructions ("no operation" in assembly) between writes and reads of the same register. For the MicroBlaze implementation used in this lab, there should be at least four cycles between sequential writes and reads of the same register.

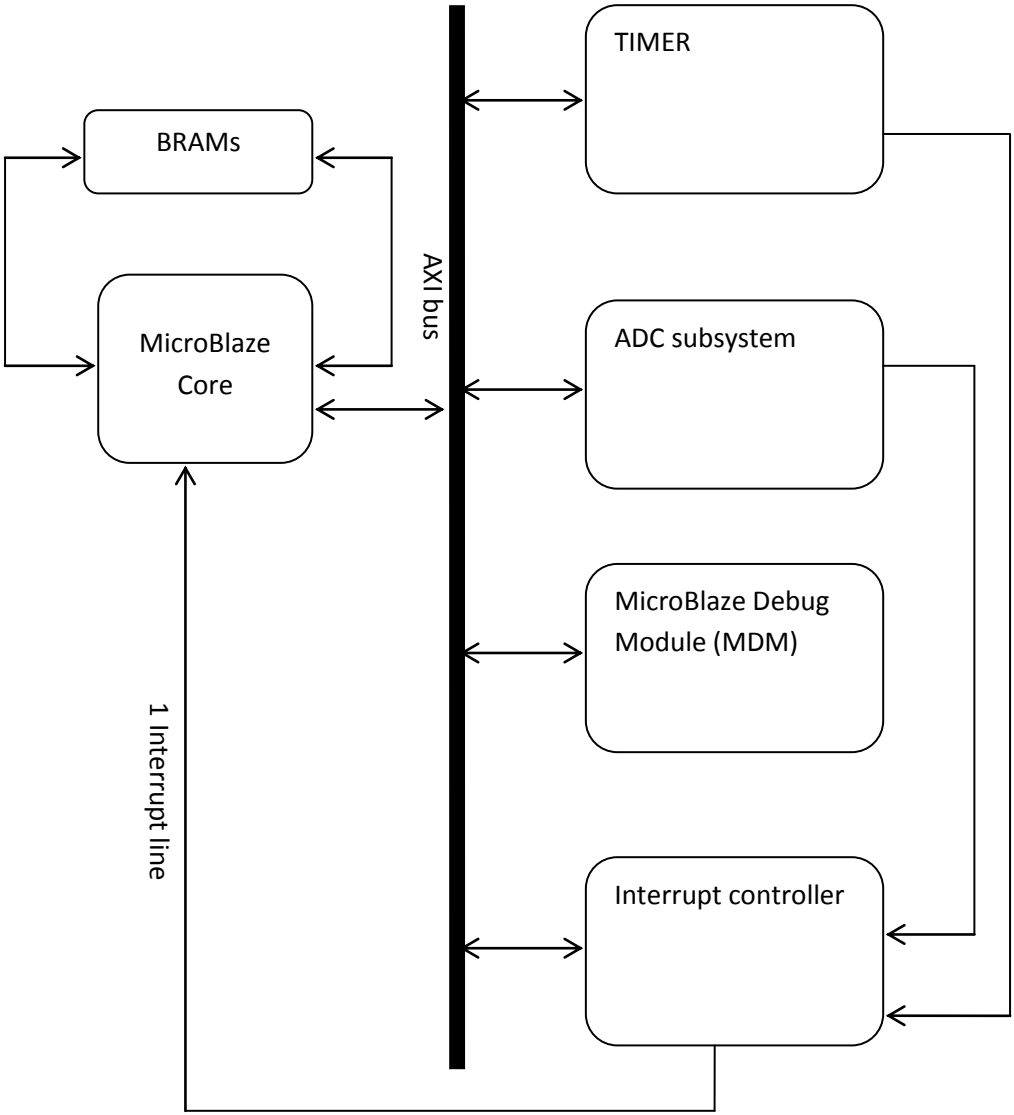
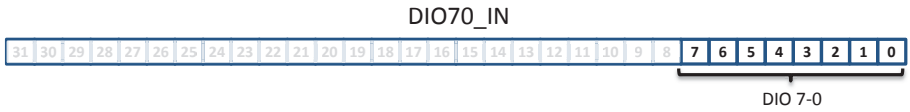


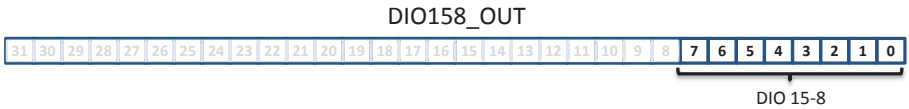
Figure 1.9: Conceptual block diagram of MicroBlaze and AXI bus architecture. Not shown: hardware peripherals such as GPIO, UART, and others.

1.3.5 Memory-Mapped Register Interface to Digital IO



Bit	Field	Access	Values
7 – 0	Digital Input	Read	myRIO DIO 7-0 input state

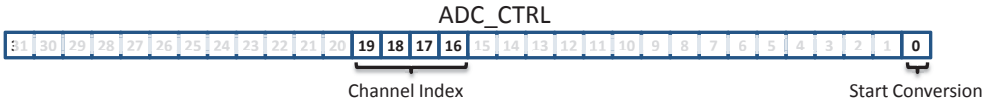
Figure 1.10: `DIO70_IN` (DIO 7-0 In) register. myRIO MXP Connector B pins 0-7. Bit 0 reads MXP Connector B DIO0, and bit 7 reads DIO7.



Bit	Field	Access	Values
7 – 0	Digital Output	Write	myRIO DIO 15-8 output state

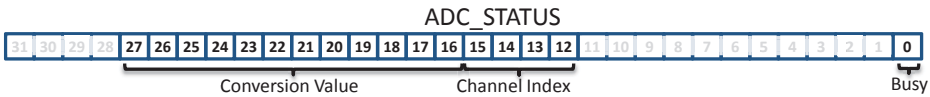
Figure 1.11: `DIO158_OUT` (DIO 15-8 Out) register. myRIO MXP Connector B pins 8-15. Bit 0 writes MXP Connector B DIO8, and bit 7 writes DIO15.

1.3.6 Memory-Mapped Register Interface to the Accelerometer ADC



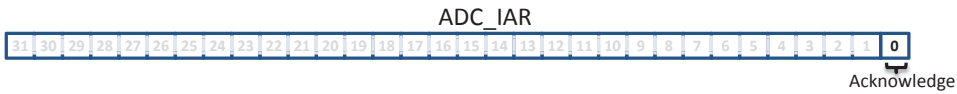
Bit	Field	Access	Values
19 – 16	Channel Index	Write	Channel 0 – 15
0	Start Conversion	Write	0: Reset 1: Start conversion

Figure 1.12: `ADC_CTRL` (ADC Control) register. The ADC begins a conversion on a rising-edge (0 to 1) transition of the Start Conversion bit.



Bit	Field	Access	Values
27 – 16	Conversion Result	Read	0 – 4095
15 – 12	Channel Index	Read	0 – 15
0	ADC Busy	Read	0: Inactive 1: Conversion in progress

Figure 1.13: `ADC_STATUS` (ADC Status) register.



Bit	Field	Access	Values
0	ADC Interrupt Acknowledge	Write	0: ADC interrupt unacknowledged. 1: Acknowledge ADC interrupt

Figure 1.14: `ADC_IAR` (ADC Interrupt Acknowledge) register.

1.4 Equipment: iRobot Create

1.4.1 Introduction to iRobot Create

The **iRobot Create** is an off-the-shelf platform capable of driving, sensing bumps and cliffs, executing simple scripts, and communicating with an external embedded controller. The iRobot Create is a complete robot development kit that can be controlled without consideration of mechanical assembly or machine code. An internal microcontroller samples its sensors, drives its actuators, and communicates with external devices over a proprietary serial interface called the **iRobot Create Open Interface** (OI) standard (iRobot, 2006a). The iRobot Create Open Interface defines the electronic and software interface for controlling the robot.

The iRobot can be finicky about its state while charging; there are some modes in which the robot will be connected to a charging source but will not actually charge. When correctly charging, the iRobot Power LED should be red, slowly fading on and off. Solid red or flashing red indicate a charging error. Solid green is, unfortunately ambiguous: it may indicate charging is complete, or it may indicate the robot is powered on and running. You may disambiguate this by removing the charging source: if the robot was powered and running, the power LED will remain solid green. If the robot is fully charged, the power LED will turn off. If you have difficulty charging your robot, first unplug any charging sources, power off the robot, and power off any external devices such as the wireless router. Then connect the charging source. Charge often and be sure to verify your robot is in charging mode when you leave the lab.

The sensors in the iRobot are at times rather inaccurate, especially the motor encoders and infrared distance sensors. Distance and angle are reported by the iRobot Create as an integer number of millimeters and degrees, respectively, and fractional remainders are truncated and lost when transmitted to an external controller. One way to mitigate wheel encoder truncation error is to increase the sensor polling period: for a fixed wheel speed, increasing the polling period allows the encoders more time to accumulate, making them numerically larger in comparison to truncation error. Another way to mitigate wheel encoder truncation error is to increase wheel speed: for a fixed sensor polling rate, increasing wheel speed causes the encoders to accumulate at a higher rate, again making them numerically larger in comparison to truncation error. Both techniques have adverse side-effects and are counter-intuitive with regard to safe and reliable operation – you may need to perform some experiments to find the polling period and wheel speeds that work best for your application. The infrared distance

sensors used for cliff detection operate more like light level sensors and are sensitive to ambient light and ground color and material. The iRobot Create reports the digital value read from this sensor, and uses a threshold to report a boolean value for whether or not this should be interpreted as a cliff. In some cases, you may need to use the digital value and determine your own threshold for cliff detection.

1.4.2 iRobot Create Useful References

Create Owner's Guide (iRobot, 2006b)

Create Open Interface Specification (iRobot, 2006a)

iRobot Create Product Page (iRobot, 2013)

Figure 1.15: iRobot Create with an external microcontroller.

1.4.3 iRobot Create Equipment Exercises

For exercises asking for a sequence of bytes, use hexidecimal (`0x__`) notation for numbers.

1. How many sensors are on the bottom of the iRobot Create? Does the placement of the sensors result in equal sensing capability when the robot is driving forward and backward?

2. What sequence of bytes should be sent to the iRobot Create UART port to command the robot to drive straight at a velocity of 300 mm/s?

3. What sequence of bytes should be sent to the iRobot Create UART port to request a single sensor packet containing all sensors?

4. What sequence of bytes should be sent to the iRobot Create to enable a continuous stream of a single packet containing all sensors?

5. After the iRobot Create has received the sequence of bytes found in Exercise 4, what sequence of bytes will the robot transmit periodically? You may represent a sensor packet as `[Packet Name (N)]` where N is the length of the packet, and checksum as `[Checksum]`.

1.5 Equipment: iClebo Kobuki

1.5.1 Introduction to Kobuki

Like the iRobot Create, the **Kobuki** is a differential drive robot capable of sensing its environment, executing scripts, and communicating with an external embedded controller via a serial port.

There are subtle hardware differences between the Kobuki and the iRobot Create. First, there are only three cliff sensors on the Kobuki instead of the iRobot's four. Second, the Kobuki measures and outputs a total distance each of its two wheels have traveled based on motor encoders.

The Kobuki also uses a different protocol to communicate with its controller. Notable differences in the protocol include a field in the packet header that contains the packet length and a move command that accepts the fastest moving wheel speed and turn radius as arguments instead of raw wheel speeds. The Kobuki continually sends feedback packets every 50 Hz containing basic sensor data over its serial extension port, as long as it is powered on. If the Kobuki stops receiving packets for a period on the order of 0.5 seconds it will shut down. The documentation for the Kobuki communication protocol can be found at (YujinRobot, 2014a).

1.5.2 Kobuki Useful References

Kobuki Documentation↗ (YujinRobot, 2014b)

Kobuki C++ Driver Specification↗ (YujinRobot, 2014a)

1.5.3 Kobuki Equipment Exercises

For exercises asking for a sequence of bytes, use hexidecimal (0x__) notation for numbers.

1. How many sensors are on the bottom of the Kobuki? In what respects does the Kobuki have sensing advantages or disadvantages compared to the iRobot?

2. What sequence of bytes should be sent to the Kobuki's serial port to make it drive straight at 300 mm/s?

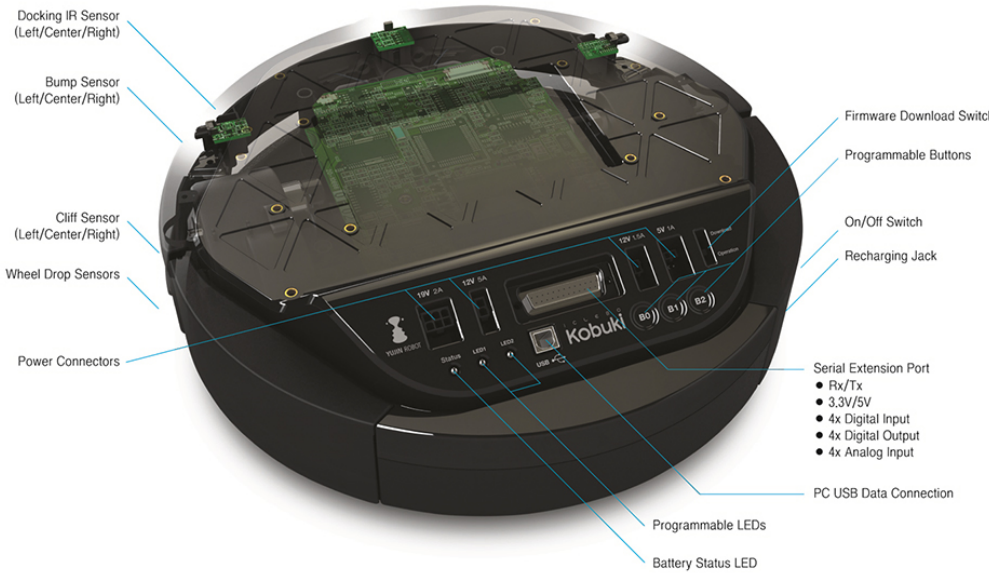


Figure 1.16: Kobuki robot with labeled sensors (YujinRobot, 2014c)

3. What sequence of bytes should be sent to the Kobuki's serial port to make it turn its right wheel at 300 mm/s and its left wheel at 250 mm/s?

4. How would the Kobuki report that the right and center bumpers were pressed simultaneously with button 2? Give the packet, writing xx for bytes that are don't cares.

5. You want the Kobuki to continue executing a single command for 5 seconds. How might you prevent it from shutting down mid-action?

1.6 Equipment: The myRIO Accelerometer

1.6.1 Introduction to the myRIO Accelerometer

The myRIO onboard accelerometer is a three axis analog accelerometer used to measure gravity and coordinate acceleration. It is a **microelectromechanical system** (**MEMS**) composed of very small mechanical and electrical components whose size is typically measured in microns. The accelerometer used is a Freescale MMA8452Q 3-axis digital accelerometer.

Special considerations come in to play on how an accelerometer interacts when it is mounted on a moving robot.

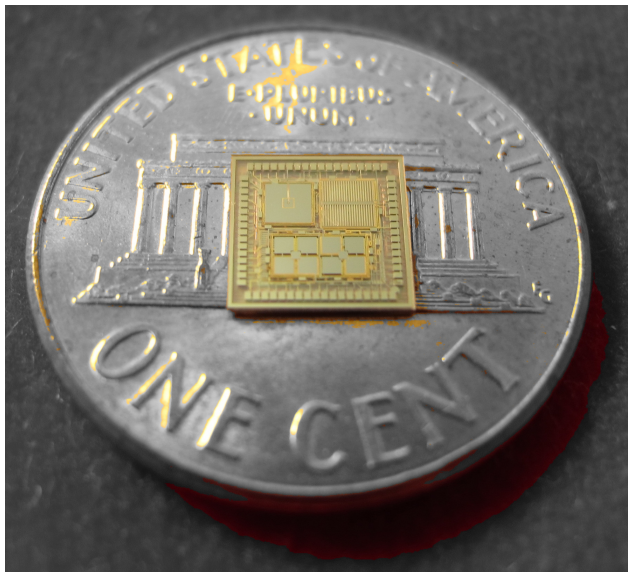


Figure 1.17: Inertial Measurement Unit (IMU) containing a three-axis accelerometer developed by DARPA researchers at the University of Michigan. Source: Defense Advanced Research Projects Agency (DARPA) (Defense Advanced Research Projects Agency, 2012).

1.6.2 myRIO Accelerometer Useful References

Xtrinsic MMA8452Q 3-Axis 12-bit/8-bit Digital Accelerometer (Freescale Semiconductor, 2013)

Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors (Ozyagcilar, 2011)

Using an Accelerometer for Inclination Sensing (Fisher, 2010)

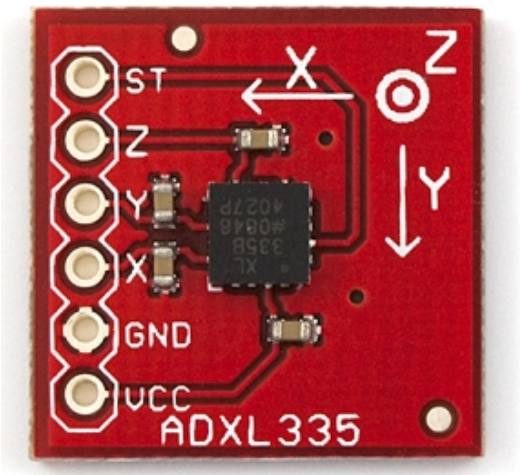


Figure 1.18: Analog Devices ADXL335 MEMS accelerometer. Credit: SparkFun (SparkFun Electronics, 2013).

1.6.3 MyRIO Accelerometer Exercises

1. Consider an accelerometer mounted according to the coordinate system in Fig. A.3. Let $\vec{a}_g = (a_{g_x}, a_{g_y}, a_{g_z})$ be the specific force of gravity as measured by an accelerometer. You may assume the accelerometer has been calibrated and is in units of g.

(a) Find $\vec{a}_g$ when the robot is on level ground. Verify $\|\vec{a}_g\| = 1g$.
(b) The robot is placed on a hill with inclination $\theta_I \in \left[0, \frac{\pi}{2}\right]$ in radians. Find $\vec{a}_g$ when the robot is oriented directly uphill. Verify $\|\vec{a}_g\| = 1g$.
(c) The robot is placed on a hill with inclination $\theta_I \in \left[0, \frac{\pi}{2}\right]$ in radians. Find $\vec{a}_g$ when the robot is rotated θ radians clockwise from uphill orientation. Verify $\|\vec{a}_g\| = 1g$.

2. This exercise highlights the joint dynamics between physical processes and embedded control. For this exercise, consider a robot with an accelerometer mounted as shown in Fig. A.3.

(a) An object on the surface of the Earth experiences a roughly constant gravitational force. If the robot is placed at rest in two distinct positions and orientations in space, will the accelerometer measurements in each position necessarily be equal?
(b) In what two ways does movement of the robot influence the accelerometer measurement? *Hint: What does an accelerometer actually measure?*
(c) One of the two components found in part 2b is undesirable when measuring tilt. What is the undesired component, and how might it be reduced or eliminated?

1.7 Equipment: Microsoft Visual Studio

Microsoft Visual Studio is the primary development platform for Microsoft Windows software, including desktop, server, tablet and mobile. Visual Studio includes support for multiple programming languages including C and C++.

1.7.1 Visual Studio Useful References

Visual Studio (Microsoft, 2014a)

Visual Studio Get Started (Microsoft, 2014b)

1.8 Equipment: Eclipse

1.8.1 Introduction to Eclipse

Eclipse from the Eclipse Project is the most widely-used open-source IDE, having been adapted as the IDE of choice for many hardware vendors. Like most IDEs, Eclipse provides a workspace to organize project files and development perspectives, a project to organize source code and dependencies, and build specifications that define communication with hardware targets.

Like most IDEs, Eclipse provides a workspace to organize project files and development perspectives. A project organizes source code, dependencies, and build specifications that define communication with hardware targets.

1.8.2 Eclipse Useful References

Eclipse Juno Documentation (Eclipse Foundation, 2013b)

Eclipse (Eclipse Foundation, 2013a)

C Support for NI myRIO User Guide (National Instruments, 2013a)

1.9 Equipment: LabVIEW

1.9.1 Introduction to LabVIEW

Model-based design emphasizes mathematical modeling to design, analyze, verify, and validate dynamic systems. Mathematical models are used to design, simulate, synthesize, and test cyber-physical systems, and are based on system specifications and analysis of the physical context in which the system resides.

Graphical design environments provide developers with a higher level of abstraction than traditional imperative programming models. National Instruments **LabVIEW** is a graphical design environment for scientists and engineers that emphasizes model-based design of embedded and cyber-physical systems. LabVIEW applications run on desktop computers or embedded controllers, and are most commonly designed to interface with sensors, actuators, instruments, data acquisition devices, and other computers and embedded devices.

LabVIEW supports heterogeneous compositions of several models of computation: continuous systems are expressed as ordinary differential equations or differential algebraic equations, and discrete systems are expressed as difference equations, in the Signal Flow model of computation; concurrent state machines are expressed in the Statecharts model of computation; imperative expressions are expressed as Formula Nodes (a subset of ANSI C) or MathScript Nodes (compatible with scripts created by developers using The Mathworks, Inc. MATLAB software and others); data acquisition and program flow are expressed in **Structured Dataflow** (Kodosky et al., 1991), the predominant language in LabVIEW. Structured Dataflow is often referred to as the **graphical language** (**G**).

A LabVIEW application is called a **virtual instrument** (**VI**). A VI consists of a **front panel** and a **block diagram**. The front panel provides an interface for setting parameters, executing a VI, and viewing results. The block diagram defines the behavior of the application. A front panel and its corresponding block diagram are shown in Fig. 1.19-1.20.

LabVIEW applications targeting the processor on myRIO are compiled into machine code and executed within an RTOS.

Figure 1.19: LabVIEW front panel for the Microsoft XBox Kinect.

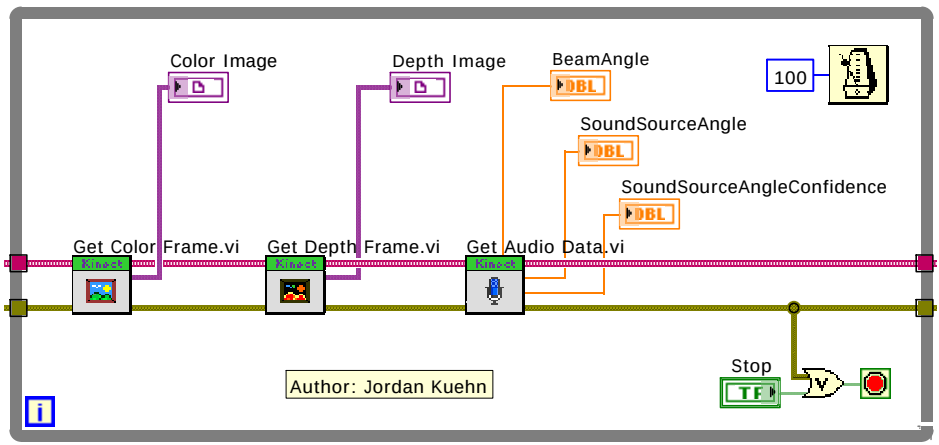


Figure 1.20: LabVIEW block diagram for the Microsoft XBox Kinect.

1.9.2 LabVIEW Useful References

Learn LabVIEW (National Instruments, 2014)

Getting Started with LabVIEW (National Instruments, 2013b)

LabVIEW Quick Reference Card (National Instruments, 2010)

LabVIEW 2013 Help (National Instruments, 2013c)

LabVIEW 2013 Real-Time Module Help (National Instruments, 2013d)

LabVIEW Product Page (National Instruments, 2013e)

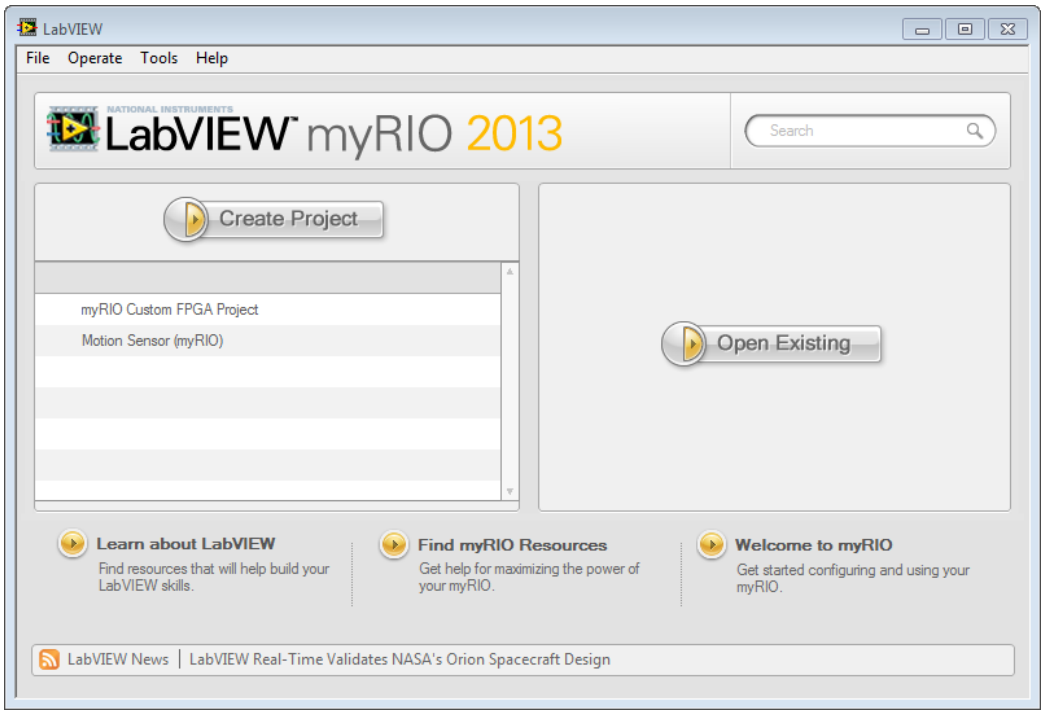


Figure 1.21: LabVIEW myRIO Start Screen.

1.10 Equipment: CyberSim

1.10.1 Introduction to CyberSim

CyberSim is software used to simulate cyber-physical systems using models of both cyber (software) and physical systems. CyberSim is built especially for the laboratory exercises in this book and is the product of a collaboration between the University of California, Berkeley and National Instruments. CyberSim is written in LabVIEW using the **LabVIEW Robotics Simulator**, and its source code is released open source at http://LeeSeshia.org/lab.

Discrete simulations of continuous systems are approximations and are subject to numerical error. These simulations are based on ordinary differential equations (or differential algebraic equations), that are computationally expensive to solve, and numerical accuracy is balanced against real-time performance. The scope of any simulator is limited: one simulation tool may focus on kinematics, another on electricity and magnetism, another on fluid mechanics, another on energy, another on quantum and nuclear dynamics, and so on. Some tools are dedicated to modeling the behavior of computers themselves!

There are currently two versions of CyberSim: one that has a physical model of the iRobot Create, and another that has a physical model of the Kobuki. In this section, we mainly focus on describing the iRobot Create version of CyberSim, noting that the user interface of CyberSim and basic physical modeling of both robots in CyberSim are similar. Physical models are based on the the Open Dynamics Engine (Smith, 2012) rigid body dynamics software that can simulate robots in a virtual environment and render them in 3D. iRobot Create sensor packets are constructed from the state of the simulation and passed into a controller that outputs desired wheel speeds. Effects such as control period, quantization, and noise are built into the simulator and are meant to closely reflect control software running on an embedded controller interfaced with the iRobot Create.

LabVIEW Robotics Simulator provides a graphical interface for customizing environments, including adding, removing, and moving obstacles, changing the initial position of the robot, adjusting sensor locations, and adding multiple robots to the environment. Robots modeled in CAD software such as SolidWorks can be imported and customized with sensors and actuators. The configuration of the environment, including dimensions

and placement of objects and robots, is stored in an XML file. The resulting XML file may be read in by CyberSim and used to simulate a new environment.

1.10.2 CyberSim Useful References

LabVIEW Robotics Simulator Wizard (National Instruments, 2013f)

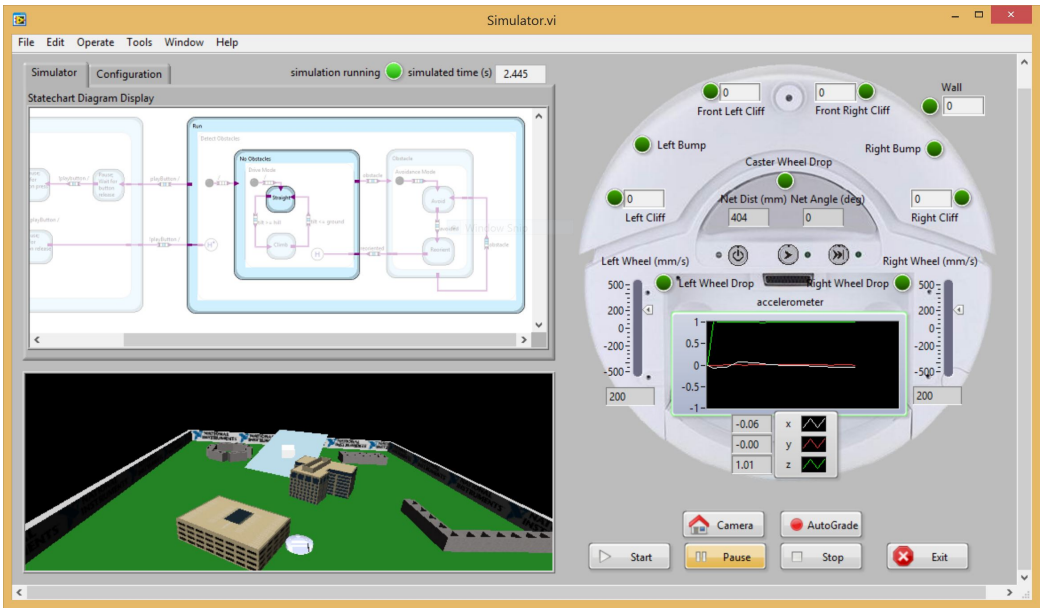


Figure 1.22: iRobot Create in CyberSim.

1.10.3 CyberSim User Interface

CyberSim is launched as a standalone executable, `CyberSim.exe` . Upon launching, the user interface (Fig. 1.22) appears. Before a simulation can run, the simulator needs to be configured with an environment and a statechart controller. The environment is an XML file generated by the LabVIEW Robotics Environment Simulator, and several such environments are included with CyberSim. The statechart controller may be either a LabVIEW VI or a Windows Dynamic-Link Library (DLL) that receives sensor inputs and produces actuator outputs. The paths to the environment and the statechart controller are set under the Configuration tab.

When running a simulation, a 3D rendering of the simulation will appear. If using a LabVIEW Statechart as the controller, its execution will also be shown alongside the simulation.

Controlling and Navigating a Simulation

Once a simulation environment and statechart controller have been specified, use the following buttons to control a simulation:

- Start begins a simulation.
- Pause pauses the simulator. Simulation time does not advance and the statechart controller is not executed while paused. Press again to resume the simulation.
- Stop stops a simulation and unloads the environment and statechart controller.
- Camera resets the simultaion camera to its default vantage.
- Exit stops an executing simulation and closes CyberSim.

To navigate around the 3D rendering of the simulation, mouse over the rendering and the cursor will change, indicating the navigation mode when clicking:

- Click and drag to navigate spherically around the clicked point.
- Ctrl + click and drag to strafe along the visible plane of the simulation.
- Shift + click and drag to zoom in and out.

Command-Line Arguments

The default configuration of CyberSim may be overridden using the following command-line arguments:

Option	Parameters	Meaning
/enableVisualization	true, false	enable visual rendering of the simulation and environment and statechart
/environment	*environment path*	Path to the environment manifest XML file. May be absolute or relative.
/maxTime	seconds, Inf	Maximum duration over which to simulate, in seconds. Double-precision floating point number.
/record	*record path*	Path to write the simulation record XML file. May be absolute or relative.
/statechart	*statechart path*	Path to the statechart VI or DLL controller to execute with the simulation. May be absolute or relative.

Table 1.1: CyberSim.exe command-line arguments.

Sample usage:

```
CyberSim.exe /environment:HillClimb.xml /statechart:libstatechart.dll /maxTime:10.5
```

1.10.4 How the Simulation Works

The CyberSim is built atop a combination of technologies. The iRobot Create was modeled in SolidWorks and then imported into the LabVIEW Robotics Environment Simulator. The simulator (based on the Open Dynamics Engine) numerically solves the ordinary differential equations for the kinematics and dynamics of the differential drive robot in a simulated environment. LabVIEW renders the environment and robot into a 3D Picture Control for viewing the simulation.

The ODE solver uses a variable step size to advance time, allowing for higher precision (smaller step size in time) when objects are moving quickly or colliding, and faster

performance (larger step size in time) otherwise. The solver uses several random number generators in predicting collisions, noise generators for introducing sensor noise, and the engine uses nondeterministic threads – as a result the simulator is nondeterministic. You may not see the same results across two simulations with the same initial conditions. While this may be undesirable from a testing standpoint, this comes with the side effect of being closer to real life, where initial conditions and an environment are never exactly replicable.

The internal microcontroller on the iRobot Create may be configured to periodically sample its sensors and transmit a sensor packet via serial port to a connected embedded controller such as myRIO. Due to design flaws with the iRobot Create, sampling less frequently produces more accurate results, and CyberSim uses the sampling period of 60ms. CyberSim models this by advancing simulation time by 60ms between each execution of the control algorithm. At each control period, the simulated world is sampled to create a sensor packet equivalent to that sent by the real robot. Single-precision floating point quantities of the simulated world are quantized and converted into the format used by each sensor.

The inputs to the control algorithm include a simulated iRobot Create sensor packet and values of the three-axis accelerometer on myRIO. The outputs of the control algorithm are the desired left and right wheel speeds. The clearly defined interface between the simulator and the control algorithm allows the same control algorithm to be used when interfaced with the real robot. The ability to design and tune a control algorithm in simulation and then deploy and verify on the real robot is an example of model-based design, shown in Fig. 1.23.

The control algorithm is provided by the user. Two interfaces are supported, a Windows DLL or a LabVIEW VI. When creating a Windows DLL, the controller must match the prototype shown in Fig. 1.24. The sensor stream parameter refers to the contents of a sensor stream packet sent by the iRobot Create containing SensorGroup6, as defined in the iRobot Create Open Interface. The return value is an error or success code, where zero indicates no error.

When using a LabVIEW VI, the VI front panel must match the template shown in Fig. 1.25.

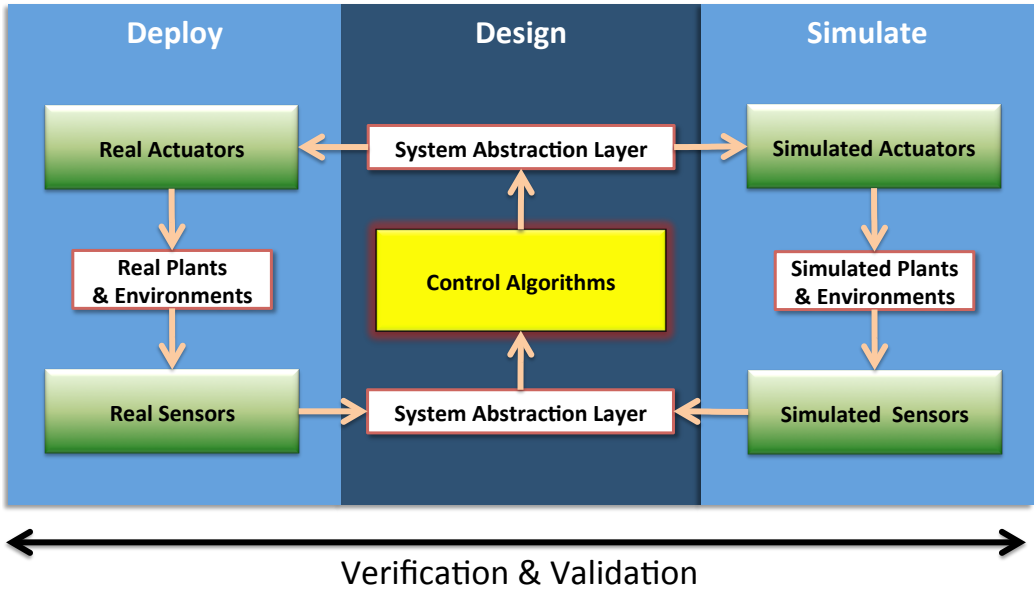


Figure 1.23: An example model-based design workflow.

```
int32_t __cdecl irobotNavigationStatechartSimulation(
    const int32_t netDistance,
    const int32_t netAngle,
    const uint8_t * sensorStream,
    const int32_t sensorStreamSize,
    const double * accelAxes,
    const int32_t accelAxesSize,
    int16_t * pRightWheelSpeed,
    int16_t * pLeftWheelSpeed);
```

Figure 1.24: C function signature for a control algorithm used by CyberSim.

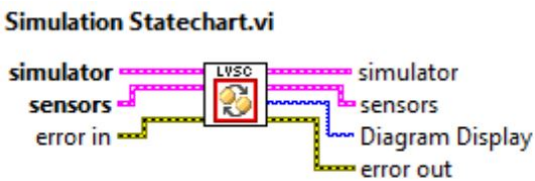


Figure 1.25: VI template for a control algorithm used in CyberSim.

1.10.5 Fidelity of the Simulation

No simulation can be 100% accurate, and you may encounter differences between the simulated and real worlds.

Distance Travelled and Angle Rotated

The simulator calculates distance traveled and rotation turned by the iRobot Create by measuring its change in position. The real robot reports distance and angle by sensing the movement of its wheels. In the simulator, distance is always unsigned. On the real robot, distance is signed. A simple way to ensure the same behavior in simulation and on the physical device is to use absolute values when calculating distance.

If the wheels on the real robot are slipping, the robot still reports this as distance traveled and angle turned, since the wheels are still moving. In the simulator, only actual movement of the robot is measured.

The simulator wraps all angles to $[-180°, +180°]$; the real robot does not.

Cliff Sensors

The infrared range finders on iRobot Create indirectly measure distance to the ground; actually, they measure light intensity. Light intensity varies significantly between different types of materials and lighting conditions. The simulator does not model light intensity, and instead the cliff sensor signal values are directly proportional to distance. On the real robot, these values may differ based on material and lighting conditions. You may need to adjust thresholds between the simulator and the real robot.

Subset of iRobot Sensors in the Simulator

Not all iRobot Create sensors are updated by the simulator, including charging sources available, battery life, battery temperature, song playing, etc. The following sensors are modeled by the CyberSim:

- Wall signal (boolean and raw)

- Cliff signals (boolean and raw)
- Bumps and wheel drops (boolean)
- Distance traveled since last sensor packet (mm) – always unsigned
- Angle traveled since last sensor packet (deg) – always signed and wrapped to $[-180°, +180°]$.

1.10.6 Troubleshooting

I receive Error 13, "File is not a Resource" when running CyberSim and a C Statechart. This is a Windows operating system message relating to an issue loading your Statechart DLL. The problem appears to be related to the compiler version used by Visual Studio. Several recommendations for resolving the issue:

- Change from Debug to Release compile mode. In Visual Studio, from the top menu bar, locate the configuration drop-down menu and select Release.
- If you are using Visual Studio Express 2013 for Desktop, change the Platform Toolset to version 120. In Visual Studio, right-click on the libstatechart project and from the context menu select Properties. In the libstatechart Property Pages dialog, navigate to Configuration Properties - General. In the right pane, locate the Platform Toolset setting and select Visual Studio 2013 (v120). Press OK to save the setting and close the dialog.
- If you are using Visual C++ Express 2010, you do not need to change your project settings. Right-click on the libstatechart project and select Rebuild. Then try running CyberSim again.

I receive Error 1003, "The VI is not executable" when running CyberSim and a LabVIEW Statechart. This error may occur for any of the following reasons:

- The Statechart source file has an error and is not executable. Check that the Statechart can generate code and does not generate errors.
- The Statechart wrapper VI is not executable. This can occur if the Statechart is not executable (see above) or if the VI has not been saved before executing in CyberSim.
- The Statechart uses advanced libraries, such as PID, Advanced Analysis Library, MathScript or others that are not supported. Use only base LabVIEW functions in your Statechart.

I receive Error 1031, "VI Reference type does not match VI connector pane." This error may occur if there is a mismatch between CyberSim and the courseware you downloaded. Ensure you are using the correct version of CyberSim and the associated courseware.

I receive Error 1088, "The Parameter is Incorrect" when running CyberSim. This error may occur when CyberSim is running from a folder that is on a network drive, or whose path includes non-ASCII characters. Move the CyberSim folder to a new

location, preferably on the C:\drive and in a path that does not include any special characters.

2

Sensor Interfacing and Calibration

2.1 Interface to and Calibrate the WiiMote

These exercises guide you in calibrating and interpreting data from a digital sensor, specifically the accelerometer from a popular gaming device. You will interface your computer with the WiiMote via Bluetooth HID wireless link, obtain raw data from its accelerometer, calibrate, and display the results in LabVIEW.

There are no prerequisites for this lab.

2.1.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.9.1 (Introduction to LabVIEW)
- §1.1.1 (Introduction to WiiMote)
- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 7: Sensors and Actuators §7.1 (Models of Sensors and Actuators)
  - Chapter 7: Sensors and Actuators §7.2.1 (Measuring Tilt and Acceleration)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Developing Algorithms Using LabVIEW MathScript RT Module: Part 1 - The LabVIEW MathScript Node (National Instruments, 2012a)
- Wii Remote (Wikipedia, 2013d)
- WiiMote (WiiBrew, 2012)
- Bluetooth HID Profile (Bluetooth Special Interest Group, 2003)
  - §"Introduction to the HID Protocol", p. 18-23
  - §"BT HID Transaction Header", p. 57-64
- Using an Accelerometer for Inclination Sensing (Fisher, 2010)
- Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors (Ozyagcilar, 2011)

These documents go beyond the scope of the core concepts of these exercises, and are provided as additional resources.

- Getting Started with MathScript RT (National Instruments, 2012c)
- MathScript RT Module Functions (National Instruments, 2012e)

2.1.2 Prelab Exercises

1. This exercise requires that you read about the WiiMote hardware. Part of the goal of this exercise is to get familiar with how to find the required information. For each question, visit the unofficial Wikipedia entry on WiiMote, or the unofficial WiiBrew Wiki; where possible, verify against manufacturer specifications, though many internal specifications have not been released.

(a) What is the manufacturer and model of the primary embedded computing system?
(b) What type of processor architecture is used by this embedded computing system? You may need to perform an internet search on the system you identified in the previous question.
(c) What is the type and size of the memory chip?
(d) What is the manufacturer and model of the accelerometer?
(e) What is the communication protocol and speed of the accessory connector port?

2. This exercise requires that you read about the Bluetooth interface to the WiiMote. Part of the goal of this exercise is to get familiar with how to find the required information. For each question, first visit the unofficial WiiBrew Wiki to locate the relevant command sequence, then reference the official Bluetooth HID documentation to decode its meaning. Use hexidecimal (0x__) notation for all numeric values.

(a) What is the command to enable LEDs 1 and 3? Identify the Transaction Header (Transaction Type and Parameter), Report ID, and Payload (if present).
(b) Given the sequence to enable LEDs 1 and 3, how can it be modified to also enable the rumble motor?
(c) What is the command to enable continuous reporting of the core buttons and the three axes of the accelerometer (even if the device is at rest)? Identify the Transaction Header (Transaction Type and Parameter), Report ID, and Payload (if present).
(d) What is the Transaction Header transmitted by the WiiMote when sending a report of its sensors?

3. Apply the affine function model of a sensor to interpret sensor data returned from the WiiMote. Let a be the sensitivity of the accelerometer, b its bias, and x the acceleration in one axis. Let $f : \mathbb{R} \to \mathbb{N}$ model the accelerometer, where $f(x)$ is the output of the accelerometer in response to acceleration x.

(a) Use the affine function model of a sensor to find $f(x)$ in terms of x, a, and b.

(b) The digital accelerometer transmits $f(x)$ to a computer or embedded controller. It is often useful in calculations to use the physical quantity of acceleration. Solve the affine function model equation for x.

(c) A digital accelerometer measures acceleration and uses an internal ADC to produce a digital output. The output of an ADC is often referred to as an **ADC unit**. If x is given in units of *g-forces* (or just g), what are the units for a, b, and $f(x)$?

2.1.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\LabVIEW\WiiMote\

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments MathScript RT Module 2014
- Nintendo Wii Remote
- Bluetooth radio

1. **Unblock a Library Dependency:** When downloading files from an unsecure source (such as the internet), Microsoft Windows blocks certain libraries from being loaded on your computer. From the downloaded exercise files, navigate to

src\LabVIEW\WiiMote\Dependencies\

right-click on the file WiimoteLib.dll and select "Properties". If at the bottom of the Properties dialog you see a button Unblock, press it followed by OK to save the library. You also need to navigate to

src\LabVIEW\WiiMote\Dependencies\32feet.NET\Assemblies\

and unblock InTheHand.Net.Personal.dll. The library is now unblocked and may be loaded by LabVIEW.

2. **Pair a desktop computer with the WiiMote:** Open the VI **WiiMote Pair.vi**, which pairs your desktop computer with your WiiMote. Given the MAC address of your WiiMote, the VI searches available Bluetooth devices and will pair your device if its

MAC address is found. Though the Windows operating system provides an interface to pair Bluetooth devices, using this VI to pair by MAC address eliminates confusion that arises when multiple WiiMotes are available.

You do not need to modify or view the block diagram of **WiiMote Pair.vi**. Key in the MAC address of your WiiMote, and run the VI to pair. When pairing the WiiMote, you must repeatedly and simultaneously press the 1 and 2 user buttons for the entire pairing process. Continue to press and let go until the after VI has completed (Fig. 2.1). When correctly paired, you should see a Windows notification that a Bluetooth HID driver was installed, and the WiiMote should flash all four user LEDs continuously. If the LEDs do not flash continuously, run the pair VI again.

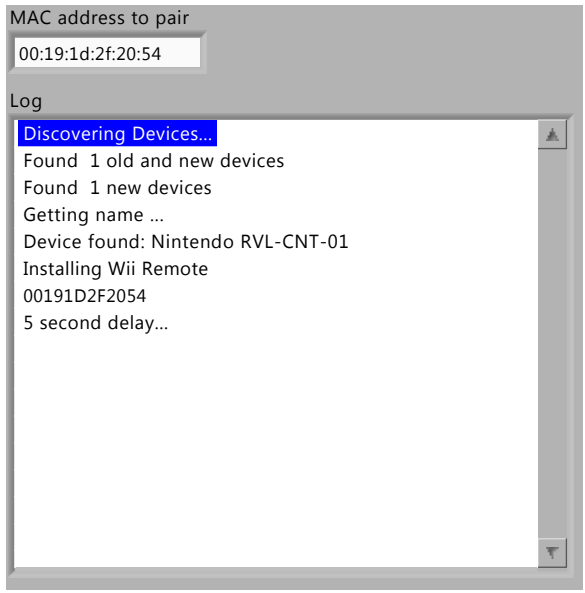


Figure 2.1: **WiiMote Pair.vi**, showing successful pairing with the WiiMote.

(a) What is the MAC address of your WiiMote?

3. **Plot the uncalibrated accelerometer signal from the WiiMote:** after pairing, open and run **WiiMote Interface.vi** (Fig. 2.2). You do not need to modify this VI for this exercise.

(a) What are the values of the x and y axes when the WiiMote is at rest on level ground?

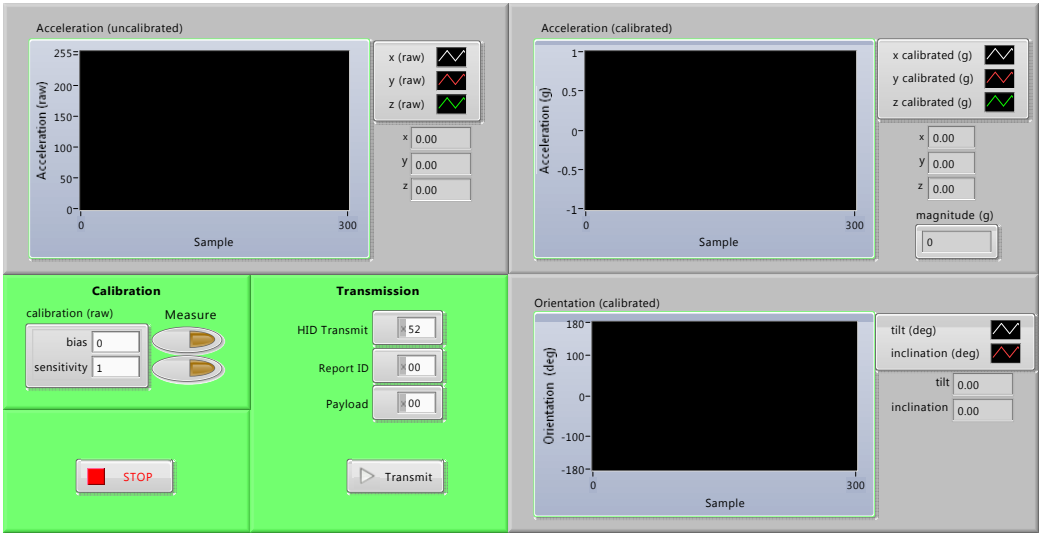


Figure 2.2: **WiiMote Interface.vi** Front Panel.

(b) What is the value of the z axis when the WiiMote is at rest on level ground? What quantity is being measured?

4. **Control the WiiMote actuators:** In the front panel of **WiiMote Interface.vi**, modify the values for **Report ID** and **Payload** to toggle user LEDs and to toggle the rumble motor.

Hint: *After an instruction has been sent to the WiiMote to turn off all four user LEDs, the device appears to be powered off even though it is still running. To preserve battery life, hold down the power button for three seconds when the device is no longer in use.*

(a) What is the Report ID and Payload to turn WiiMote LEDs 2 and 4 on, and LEDs 1 and 3 off?
(b) How can you modify the above sequence to also enable the rumble motor?

5. **Measure sensitivity and bias of the accelerometer:** The buttons on the front panel are associated with panes in the **Event Structure** within the block diagram. To view the subdiagrams for each event handled, use the scroll arrows next to the name of the event handled. Modify the subdiagrams of the **Event Structure** so that, when the **measure bias** or **measure sensitivity** button is pressed on the front panel, the corresponding measurement recorded and displayed in the **calibration (raw)** control. You may assume

the sensitivity and bias are approximately the same for all three axes. You do not need to create new buttons or events for this exercise.

Hint: *To retain your calibration parameters after LabVIEW is closed, right click on the* **calibration (raw)** *control on the front panel and select "Data Operations→Make Current Values Default".*

(a) Where are these measurements stored?
(b) What bias did you record?
(c) What sensitivity did you record?
(d) Provide screenshots of changes to the block diagram code.

6. **Calibrate the accelerometer:** modify the concents of the **MathScript Node** in **WiiMote Interface.vi** so that the **Accelerometer (calibrated)** chart plots acceleration of the x, y, and z axes, and the **magnitude** indicator displays the magnitude of acceleration, all in units of g.

(a) What equations did you use to calibrate the accelerometer and to calculate the magnitude of the acceleration it measures?
(b) Based on your calibration parameters and the number of bits of resolution of the WiiMote ADC, and an affine model of the WiiMote accelerometer, what is your estimate of the maximum acceleration that the WiiMote can measure?
(c) When the WiiMote is at rest, what should be the value of the **magnitude** indicator?
(d) Why is it better to perform calibration at runtime, rather than hard-coding values for the sensitivity and bias of the sensor?
(e) Provide the relevant MathScript code and screenshots of changes (if any) to the block diagram.

7. **Measure Pitch and Roll:** modify the contents of the **MathScript Node** in **WiiMote Interface.vi** so that the **Orientation (calibrated)** chart plots pitch and roll of the WiiMote, in units of degrees.

Hint: *The coordinate system used in Implementing a Tilt-Compensated eCompass and the coordinate system used by the WiiMote (Fig. 1.3) are related by swapping the x and y axes. Note that the definition for pitch and roll are switched in this document compared to the pitch and roll in Figure. 1.3*

(a) What equations did you use to calculate pitch and roll?
(b) Provide the relevant MathScript code and screenshots of changes (if any) to the block diagram.

8. **Filter the accelerometer signal:** Accelerometers are often used to detect high frequency signals such as noise and vibration; for a game controller, lower-frequency movements of the user should be isolated. In particular, the rumble motor will introduce a high-frequency signal. Implement a lowpass filter to smooth the signal of the calibrated accelerometer when the rumble motor is on.

Hint: *A simple lowpass filter is discussed in Implementing a Tilt-Compensated eCompass*.

(a) Describe your filter, and include an equation for its impulse response, frequency response, or its output as a function of input.
(b) Provide the relevant MathScript code and screenshots of changes (if any) to the block diagram.

9. **Share your Feedback:** what did you like about this lab, and what would you change?

2.1.4 Troubleshooting

I receive Error 1172, "WiimoteLib.WiimoteNotFoundException" when running WiiMote Interface.vi. This error occurs if your desktop is not paired with a WiiMote. Run **WiiMote Pair** to pair again.

I am able to pair the WiiMote, but WiiMote Interface.vi either does not show input or returns Error 1172 "WiimoteLib.WiimoteNotFoundException". Check the WiiMote batteries – when low, the device may stay powered on long enough to pair, only to power off shortly thereafter. When first powered on and paired, the four LEDs should remain flashing until the device is powered off, or the LED state is explicitly changed by a Bluetooth command.

I am able to pair the WiiMote and run WiiMote Interface.vi, but after a few seconds the output does not change in response to movement of the controller. Check the WiiMote batteries – when low, the device may stay powered on long enough to pair, only to power off shortly thereafter. When first powered on and paired, the four LEDs should remain flashing until the device is powered off, or the LED state is explicitly changed by a Bluetooth command.

I receive Error 1172, "Timed out waiting for status report". This may occur if you are using a WiiMote clone (i.e. not manufactured by Nintendo), and are not using the WiiMoteLib binary distributed with the lab materials. In initializing the WiiMote, the open-source WiiMoteLib verifies a vendor ID, which differs between authentic and cloned WiiMotes. We modified the WiiMoteLib source to omit this call during initialization, and provide the source as part of the downloadable lab materials.

I receive Error 1172, "Request for the permission of type... failed". This is a Microsoft .NET exception, indicating WiimoteLib.dll could not be loaded from a network drive. If possible, try moving your lab files to a local (i.e. **C:**) drive. If your administrator has placed **WiimoteLib.dll** in a system directory, it is possible that the **WiimoteLib.dll** is loading from your lab directory first; try renaming the file **WiimoteLib.dll** to **WiimoteLib.dll.bak** and relaunch LabVIEW.

My VI is broken and raises the error, "Invalid Procedure" or "Function Not Found" on the WriteStatusReport node. This may occur if you are not using the WiiMoteLib binary distributed with the lab materials. The standard library does not allow external programs to transmit arbitrary status report packets. A modified WiiMoteLib source

and binary is provided as part of the downloadable lab materials. Make sure there are no other instances of the WiiMoteLib library in the system path.

I receive the error, "Error 1386 occurred at Constructor Node in WiiMote Pair.vi: The specified .NET class is not available in LabVIEW". Ensure you have unblocked the library file InTheHand.Net.Personal.dll by right-clicking on the file, selecting Properties, and Unblock.

3 Embedded Development Tools

Writing software for embedded microcontrollers requires the use of one (or more) development tools. Tools may be specific to the hardware, the programming languages, or the models of computation used. Processor and microcontroller vendors often provide proprietary development tools such as compilers, memory programmers, and debuggers, but in many cases open-source solutions exist such as the **GNU Compiler Collection** (**gcc**) and the **GNU Debugger** (**gdb**).

Sophisticated embedded development tools can be the product of thousands – sometimes millions – of man hours, achieving complexity beyond what a single user could ever hope to master. Embedded development tools may bring together a myriad of components, including support for dozens of hardware targets or targets released over the span of more than a decade, multiple compilers and debuggers, team development features such as source control and revision control, execution timing and performance analysis, software analysis, formal verification and validation, peripheral drivers, code synthesis, and more. Technology and tools change, and while familiarity with a collection of embedded development tools is a valuable skill, of greater value is the ability to gain familiarity with a new tool or technology by quickly navigating terse technical documentation, critically analyzing its functionalities, and contrasting it with alternative tools and technologies.

The exercises in this chapter are prescriptive in nature and introduce you to common tools for programming embedded systems. Completing these exercises will not make

you an expert in any one of the tools used – each of them is in fact the entire subject matter of one or more textbooks. Focus instead on efficiently extracting useful information from technical documentation, understanding the connection between the tool and your embedded hardware, and becoming familiar with the tools used in later chapters.

Figure 3.1: GNU logo (Suvasa, 2011).

3.1 Connect to and Configure myRIO

These laboratory exercises serve as a tutorial on connecting and configuring myRIO. You will connect your myRIO to a host computer and reset it to its factory defaults.

There are no prerequisites for this lab.

3.1.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.2.1 (Introduction to myRIO)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- myRIO 1950 User Guide and Specifications (National Instruments, 2013g)

3.1.2 Prelab Exercises

For questions relating to exact specifications, be sure to reference the correct documentation for the model of myRIO you are using.

1. What is the CPU clock frequency of the ARM Cortex-A9 processor on myRIO?

2. In this exercise, you will explore the electrical characteristics of your myRIO. These are important, as exceeding tolerances may damage or destroy your controller.

   (a) What is the input voltage range for the controller power source?
   (b) What is the analog input voltage range for single-ended measurements?
   (c) What is the maximum current that may be sourced from a single DIO line?

3.1.3 Lab Exercises

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments NI-RIO 13.1
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable

1. **Connect myRIO:** Connect a power supply to myRIO. When powered, the Power LED on the controller will illuminate. Connect myRIO to your computer using a standard USB cable. Windows should indicate a new device has been connected – if this is the first time you connected the device to your computer, it may take a few moments for Windows to install the driver.

2. **Run the Getting Started Wizard:** The **myRIO USB Monitor** should automatically open when you connect myRIO to your computer. From within the myRIO USB Monitor, press Launch Getting Started Wizard. If the NI myRIO USB Monitor does not automatically open, you may launch the Getting Started Wizard from the LabVIEW myRIO start screen by selecting "Set Up and Explore→Launch the Getting Started Wizard" (Fig. 1.21). Complete the wizard as follows:

(a) In the myRIO Discovery pane, verify that your myRIO device appears as discovered. Click the device to continue to the next step.
(b) In the Rename your myRIO pane, optionally rename your myRIO device. Press Next to continue.
(c) If this is the first time the myRIO device has been configured, the wizard will install software to the device.
(d) In the Test Onboard Devices pane, verify that the onboard accelerometer, LEDs, and buttons are active and respond as expected. Press Next to continue.
(e) In the final pane, you may choose to create a new project or open LabVIEW. You may also press Close to complete the wizard.

3. **Discover myRIO using Measurement and Automation Explorer:** MAX may be launched from the Windows desktop or from the Windows Start menu under "National Instruments→NI MAX".

In the left navigation pane, expand Remote Systems to autodiscover National Instruments devices on the local network (Fig. 3.3). Autodiscovery may take some time to complete. If your myRIO appears under Remote Systems, it has been discovered

(Fig. 3.4). Click on the myRIO device to see detailed information about the target, including the IP address of the Ethernet over USB adapter.

4. **Configure myRIO using NI Web-Based Configuration and Monitoring:** myRIO supports a web-based configuration similar to NI MAX. Determine your target IP address (shown in the NI myRIO USB Monitor or in NI MAX), and navigate to this address using a standard web browser. Browse through the options to see more about the software and peripherals on myRIO.

5. **Share your Feedback:** what did you like about this lab, and what would you change?

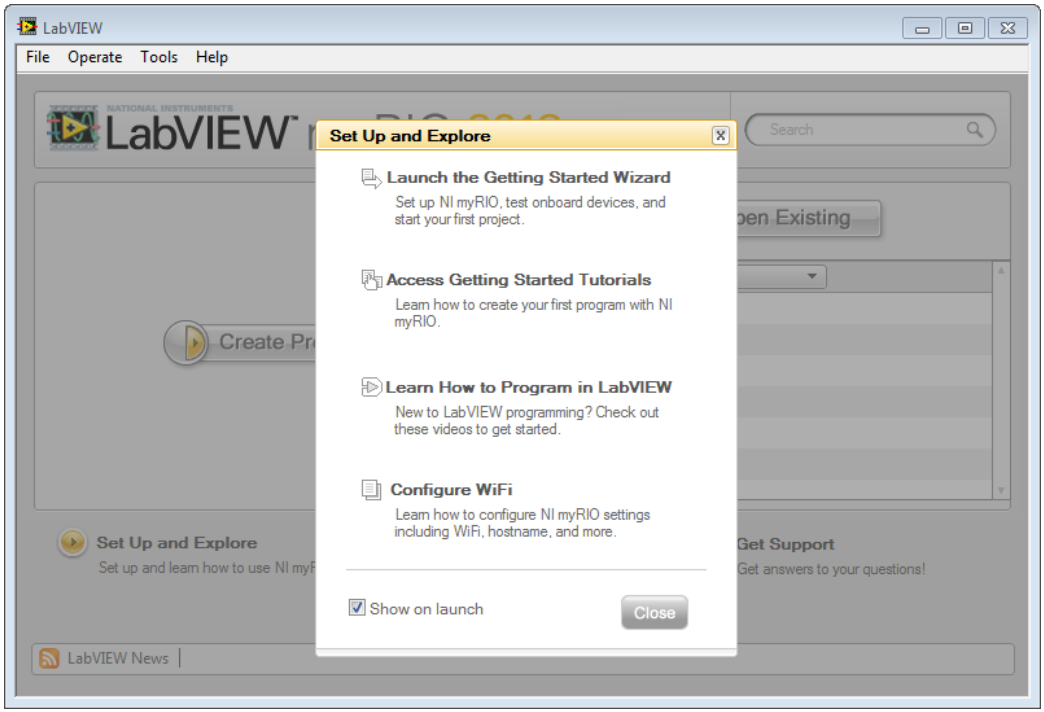


Figure 3.2: Launch the myRIO Getting Started Wizard from the LabVIEW myRIO startup screen.

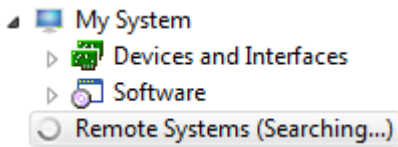


Figure 3.3: NI MAX autodiscovery pending.

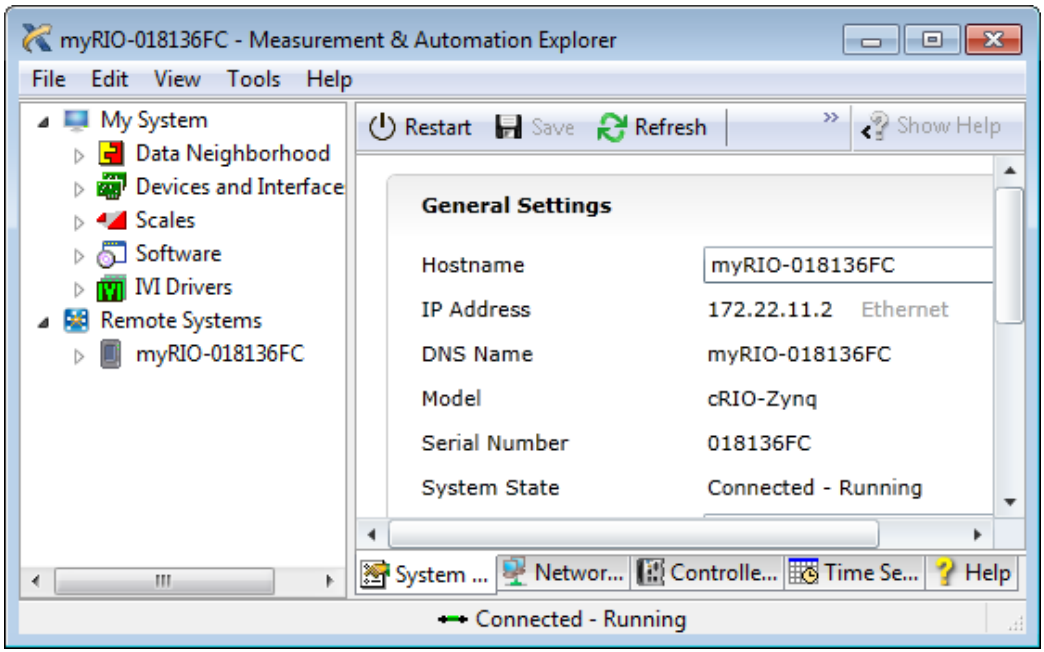


Figure 3.4: NI MAX autodiscovery showing a connected myRIO.

3.2 Program MicroBlaze from Xilinx SDK

These laboratory exercises serve as a tutorial on programming MicroBlaze on the myRIO FPGA using Xilinx SDK. You will configure your development environment, build a "Hello World" application in C from scratch, and connect to and program MicroBlaze from Xilinx SDK.

This lab builds on Lab 3.1: Connect to and Configure myRIO.

3.2.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.3.1 (Introduction to MicroBlaze)

3.2.2 Prelab Exercises

1. What are the differences between a traditional 'fixed' processor, an FPGA, and a soft-core processor?

3.2.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\XilinxSDK\

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments NI-RIO 13.1
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- Xilinx SDK 14.4
- Xilinx Platform Cable USB II

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer. See Lab 3.1: Connect to and Configure myRIO for instructions if you have not done this before.Your Xilinx Platform Cable should be connected to your desktop via USB, and to your myRIO via the JTAG header (J5). Refer to §A.5 (myRIO JTAG Wiring) for the correct pin mappings. When properly connected, the Status light on the Platform Cable will illuminate green. Connect another USB cable directly between myRIO and your computer.

2. **Deploy the MicroBlaze FPGA Fixed-Personality:** MicroBlaze is implemented on myRIO by deploying an FPGA fixed-personality. **RIO Device Setup** (Fig. 3.5) reads a bitfile to reconfigure the FPGA. RIO Device Setup erases and reprograms the FPGA of myRIO (or other NI RIO device). The tool downloads a bitfile directly to the flash memory of the FPGA, and optionally configures it to load on startup. This is the simplest way of deploying an FPGA fixed-personality that contains the MicroBlaze soft-core processor. RIO Device Setup does not program the flash memory of the ARM.

(a) Launch RIO Device Setup from the Windows Start menu, "National Instruments→ NI-RIO→RIO Device Setup".
(b) Pull-down the Resource menu and select "Browse..." to browse for the RIO0 (FPGA) target on myRIO.
(c) Press the browse button next to Bitfile to Download and navigate to the file **MicroBlaze_myRIO.lvbitx** .
(d) Press [Download Bitfile] to download the FPGA fixed-personality. This may take a minute or two.

Press the reset button on your myRIO, and confirm the LED1 is glowing. It may take a minute or two for the target to restart.

3. **Launch Xilinx SDK:** Xilinx SDK may be found in the Windows Start menu under "Xilinx SDK→SDK→Xilinx Software Development Kit". Upon launching, you may be asked to select a workspace. A workspace organizes projects, perspectives, and application preferences. Select a location to store your workspace, and Workbench will create a workspace at this location if none exists. The Xilinx SDK prohibits spaces in the workspace path. Make sure the location does not contain spaces, is writable, and if possible avoid using a network drive or fileshare – this may cause problems with the Xilinx compile toolchain.

If you are not prompted to select a workspace, then Workbench has been configured to automatically load a default workspace. If you need to change to a new location, then from the top menu bar select "File→Switch Workspace".

Standard window configurations are available for various development tasks. For most programming tasks, the Basic Device Development Perspective is sufficient. To open this perspective, from the top menu bar select "Window→Open Perspective→Other..." and select the "C/C++" perspective.

4. **Import the MicroBlaze Specification:** A hardware specification and board support package are automatically generated by the same tool that builds the MicroBlaze fixed-personality. Each is a project that will automatically generate source code needed to interface with the MicroBlaze core. To import the hardware specification and board support package, from the top menu bar select "File→Import..." to launch the Import wizard:

(a) In the Select an import source panel, select "General→Existing Projects into Workspace" and press [Next >].

(b) In the Import Projects pane, press "Select archive file" and browse to **MicroBlazeSpec_myRIO**. The projects in the archive will automatically propagate (Fig. 3.8).
(c) Press Finish to complete the import wizard.

The project will now appear in the Project Explorer view. You will see a project titled **MicroBlaze_Hardware_Specification**, a hardware description project that contains information about myRIO and how MicroBlaze is be implemented on it. **MicroBlaze_Software_Specification** is a board support package, which provides drivers to hardware peripherals on myRIO. You do not need to modify either of these projects, but feel free to explore their content.

To build the MicroBlaze specification, from the top menu bar select "Project→Clean..." and in the Clean dialog, press OK. Then from the same menu bar, select "Project→ Build All".

5. **Create a New Project:** Right-click in an empty area of the Project Explorer, and select "New→Project...". A new project wizard will open:

(a) In the Select a wizard pane, select "Xilinx→Application Project" and press Next >.
(b) In the New Project pane, set Project name to **helloWorldMicroBlaze**. Ensure the Hardware Platform is set to "MicroBlazeHardwareSpecifications" and Processor is set to "microblaze_0", OS platform is set to "standalone". Set the Board Support Package to use the existing "MicroBlazeSoftwareSpecifications". Ensure your dialog matches Fig. 3.6, and press Next >.
(c) In the Templates pane, choose "Hello World". Press Finish to complete the New Project wizard.

When the project wizard completes, your development environment should return to the C/C++ Perspective, with the **helloWorldMicroBlaze** project now appearing in the Project Explorer.

6. **Review the Template Code:** The New Project wizard automatically creates a C source file. In the Project Explorer, expand the **helloWorldMicroBlaze** project and open **src/helloworld.c**. In the `main()` function, you will see a call to the `print()` function, a function similar to `printf()` in the standard C library but specific to MicroBlaze. The standard output defaults to a serial port.

To compile the source code, from the top menu bar select "Project→Build Project". The Build Console at the bottom of your workspace will show the results of the compilation.

7. **Create and Execute a Run Configuration:** A Run Configuration is a script that will build, download, and execute your project on your target. To create a Run Configuration:

(a) From the top menu bar select "Run→Run Configurations...".
(b) In the Run Configurations wizard, right-click on Xilinx C/C++ ELF and select "New".
(c) Select the STDIO Connection tab to configure standard output for the target. Check the option Connect STDIO to Console and from the drop-down menu select "JTAG UART". This configures standard output – used by functions like `print()` – to transmit over a JTAG serial port. Xilinx SDK will automatically spawn a console that displays messages from this port.
(d) Select the Debugger Options tab and uncheck the option "Stop at main() when debugging". This enables your application to immediately execute after downloading to the target – otherwise, the debugger will place a breakpoint at the first line of the `main()` function.
(e) Lastly, to place this Run Configuration in the list of favorites, open the Common tab. Under the Display in favorites menu section, check the box next to "Run". Then click Run to save the configuration and execute the run script.

Review the Console pane to view STDIO output. If your application compiled, downloaded, and was correctly executed, you should see the text `Hello World` appear.

8. **Revise and Run Again:** Change the print message to something new. Save the file and build the project.

To stop the currently executing process on MicroBlaze, you first need to stop the processor. The **XMD Console** pane is used to issue commands to MicroBlaze and display debug messages via UART over JTAG – you must first have run your application to connect to your target over JTAG for the XMD Console to be connected to your target. From the top menu bar, select "Xilinx Tools→XMD Console". In the XMD Console pane, enter the following three commands to stop execution and reset the MicroBlaze processor: `rst`, `stop`, and `rst`.

Execute your run configuration to see the new message. Congratulations on using Xilinx SDK to build your first MicroBlaze application for myRIO!

9. **Share your Feedback:** what did you like about this lab, and what would you change?

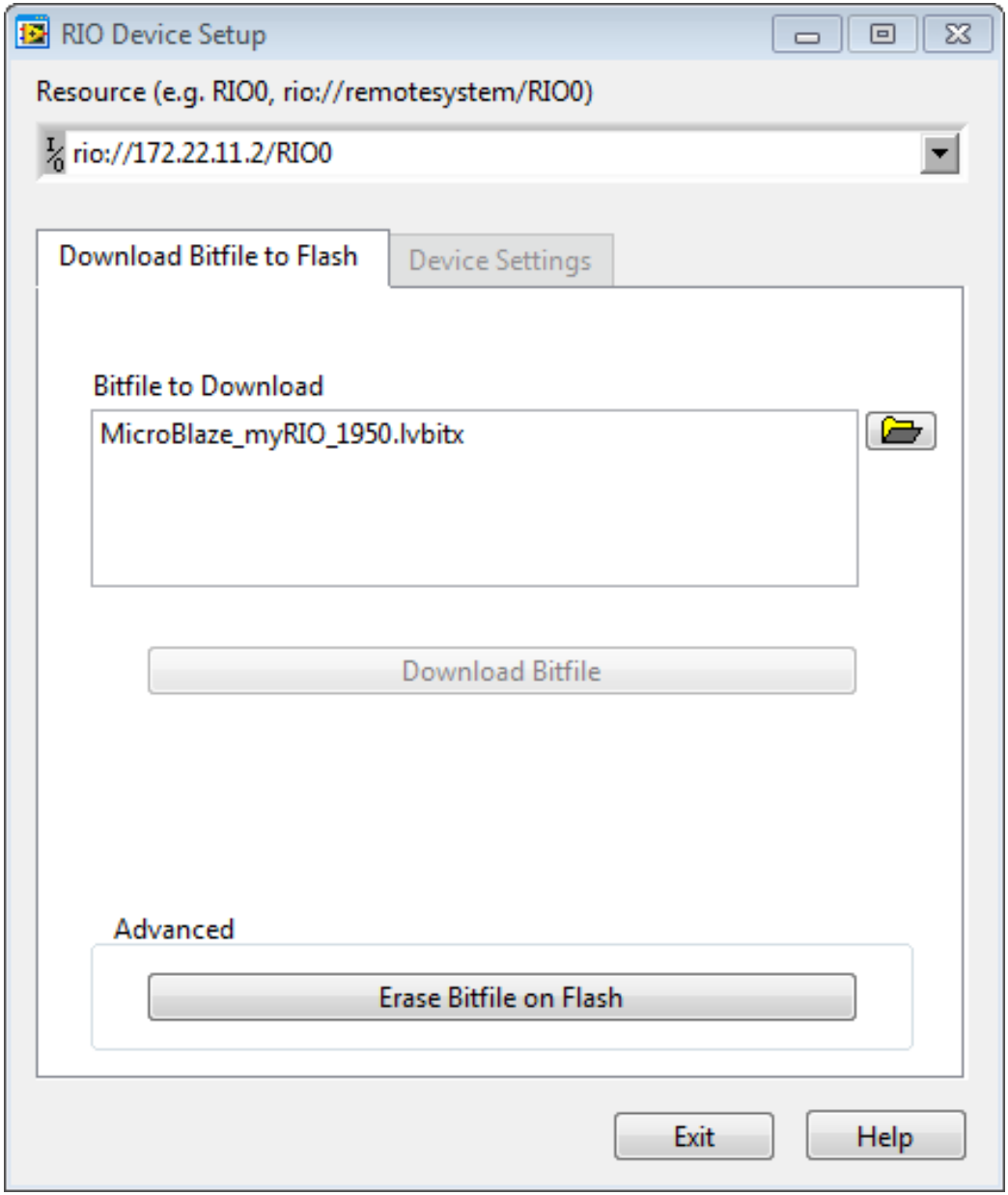


Figure 3.5: RIO Device Setup used to deploy the MicroBlaze FPGA fixed-personality to myRIO.RIO Device Setup

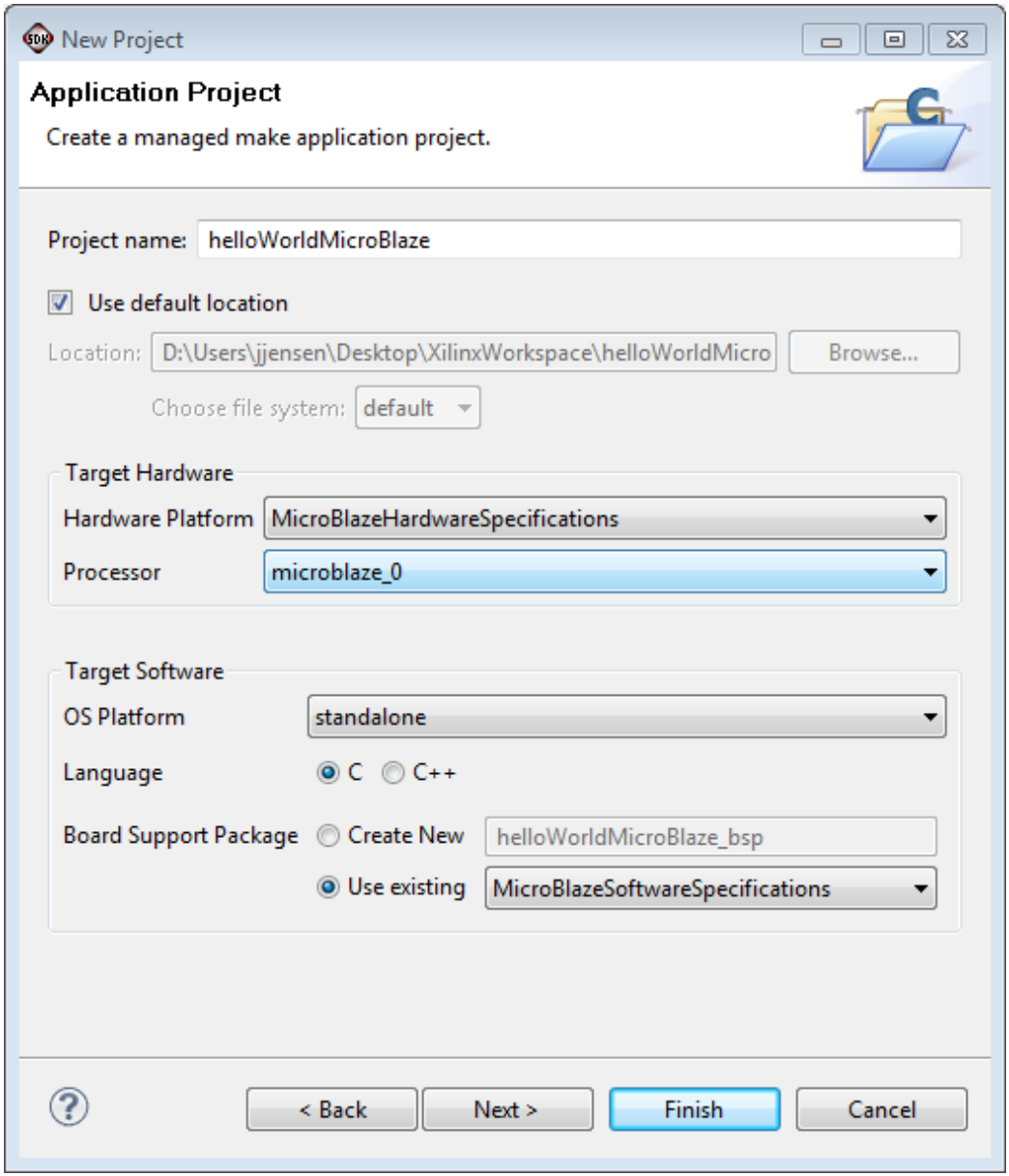


Figure 3.6: Xilinx Project Wizard.

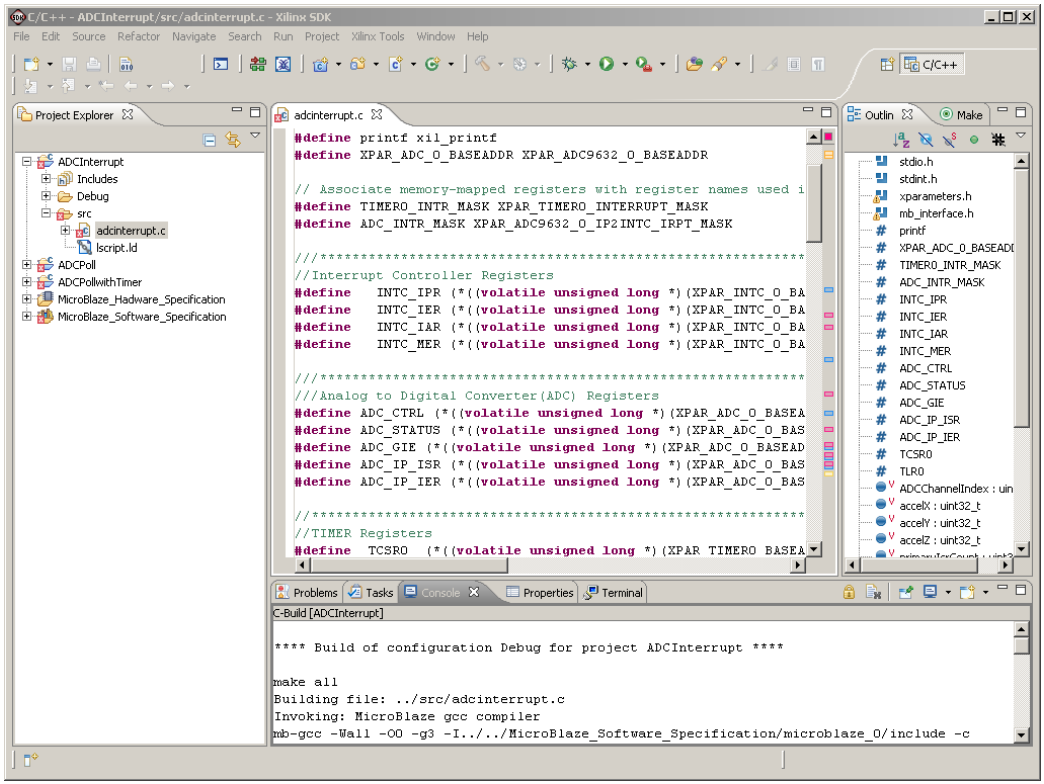


Figure 3.7: Xilinx SDK in the C/C++ perspective.

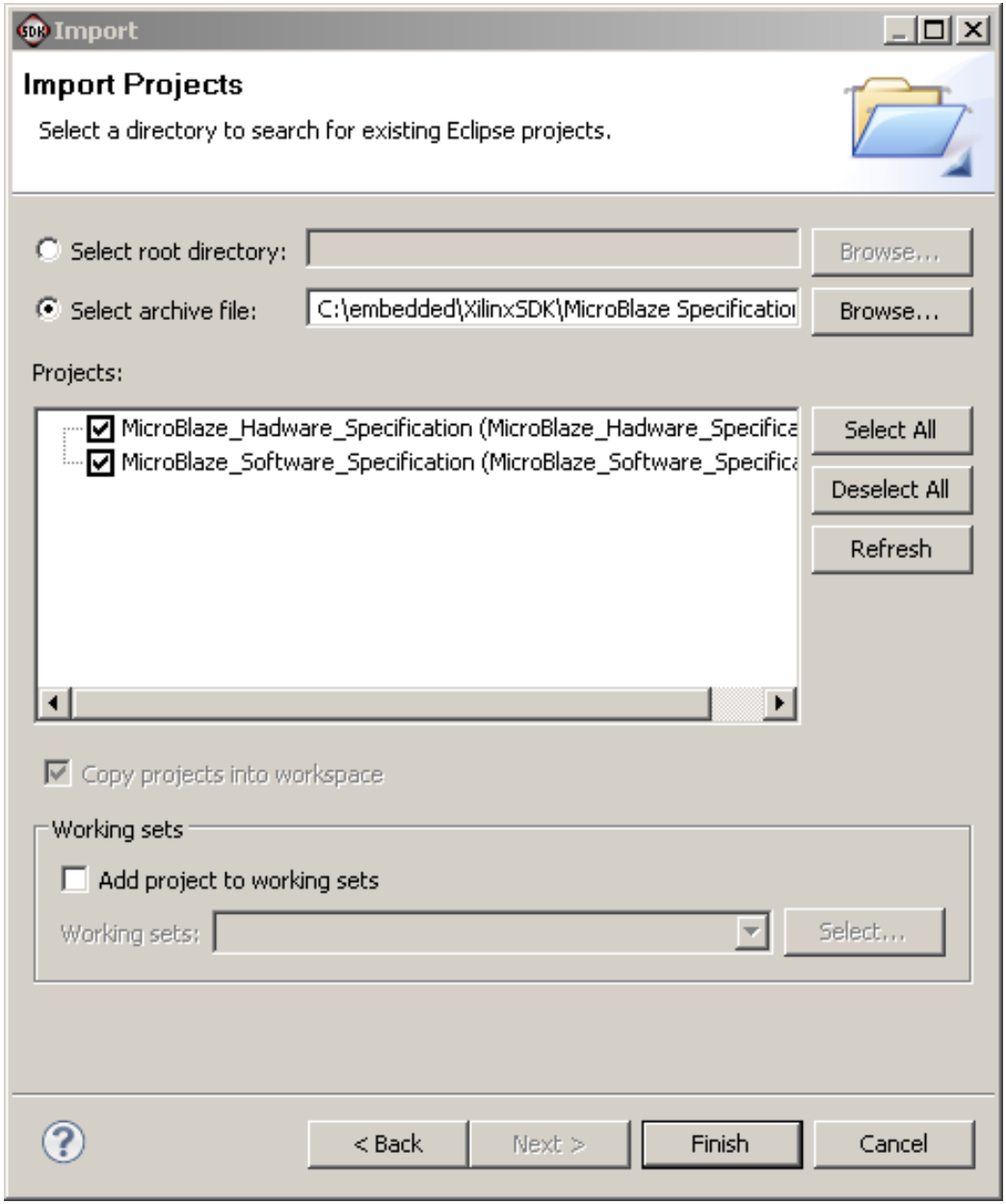


Figure 3.8: Xilinx SDK Import Wizard to import an existing project.

3.2.4 Troubleshooting

The template project builds with errors: If the template code does not build, the correct include files may not have been generated. Double-check that you have imported the MicroBlaze specification (Fig. 3.8) and have cleaned and built all projects before running the New Project wizard.

When using the New Project wizard to create a Xilinx Application Project to target MicroBlaze, nothing appears in Available Board Support Packages: Double-check that you have imported the MicroBlaze specification (Fig. 3.8) and have cleaned and built all projects before running the New Project wizard.

When running, I receive the error "Failed to Open JTAG Cable" or the console prints "Error opening JTAG UART": Double-check the wiring of the Platform USB cable (refer to §A.5 (myRIO JTAG Wiring) for the correct pin mappings), that myRIO is powered, and that the Status light on the Platform USB cable is green. Disconnect the USB cable and reconnect - Windows should indicate a recognized device has been installed.

I receive the error, "ERROR: Cannot perform the Debug Command, Current Processor State is 'Running' ". This error typically occurs immediately after the FPGA has been reprogrammed, and indicates the MicroBlaze processor is not in a state that can accept debug connections. First attempt to build and run your application, double-checking that you have set the build configuration to connect the STDIO Console to JTAG UART. In the XMD Console, enter the command `rst` to reset the processor to its initial state, followed by `stop` to stop the execution of the MicroBlaze processor. The processor should now be in its normal execution state and be able to accept debug connections.

I cannot see the XMD (debug) console: `print()` statements in software are relayed via JTAG to your desktop computer. The Xilinx XMD Console displays these messages. First, be sure that the Basic Device Development Perspective (C/C++) perspective is open (Step 3). If the console is still not visible, it can be launched manually. From the top menu bar, select "Xilinx Tools→XMD Console". The console is then connected to your target by entering the text `terminal` at the `XMD:` prompt and pressing enter. A console window should appear, and will display any `print()` messages generated by software running on MicroBlaze.

I receive the XMD Console error, "Error opening JTAG UART localhost:-1": This may be caused by trying to download and run a program without stopping the previous code. To resolve, disconnect and reconnect the JTAG USB cable, reset myRIO, and restart Xilinx SDK. To prevent this from happening, be sure to terminate your application or debugging session before downloading and running again.

The XMD Console is empty: Ensure the build configuration is correct. First try pressing the user Button on myRIO to reset MicroBlaze. If that does not work, then reset myRIO and restart Xilinx SDK.

3.3 Program the myRIO Processor from Eclipse

These laboratory exercises serve as a tutorial on programming the myRIO processor using C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition, an Eclipse-based IDE. You will configure your development environment, build a "Hello World" application in C, and connect to and program the ARM processor in Eclipse.

This lab builds on Lab 3.1: Connect to and Configure myRIO.

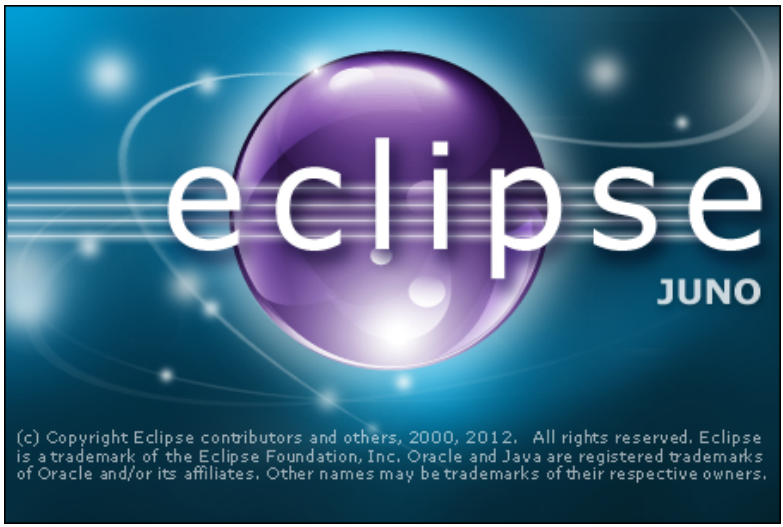


Figure 3.9: Eclipse for NI Linux Real-Time (based on Eclipse Juno). Eclipse logo is a trademark of the Eclipse Foundation.

3.3.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.8.1 (Introduction to Eclipse)

3.3.2 Prelab Exercises

There are no prelab exercises for this lab.

3.3.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\myRIO\

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments NI-RIO 13.1
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer. See Lab 3.1: Connect to and Configure myRIO for instructions if you have not done this before.

2. **Enable SSH:** Your computer communicates with myRIO using the SSH protocol. This allows you to see `printf()` messages transmitted from myRIO to your computer, and to interact with myRIO using standard Linux shell commands. You may use NI MAX to enable terminal access, or you may use the web configuration by opening your internet browser:

(a) Navigate to http://172.22.11.2 in the web browser on the host computer. If your myRIO is configured with an alternate IP address, use this address instead.
(b) The Web Configuration interface will open in your browser. In the left navigation bar, select System Configuration (the icon of a home), and under the Startup Settings section, check the open to "Enable Secure Shell Server (sshd)".
(c) Click Save to apply the new setting.
(d) Click Restart on the upper right of the page to restart myRIO.

3. **Launch Eclipse:** C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition may be found in the Windows Start menu under "National Instruments→C &

C++ Development Tools for NI Linux Real-Time, Eclipse Edition". Upon launching, you may be asked to select a workspace. A workspace organizes projects, perspectives, and application preferences. Select a location to store your workspace – Eclipse will create a workspace at this location if none exists. The location must be writable.

If you are not prompted to select a workspace, then Eclipse has been configured to automatically load a default workspace. If you need to change to a new location, then from the top menu bar select "File→Switch Workspace".

4. **Import the Template Project:** The myRIO project template specifies the compile toolchain used to compile programs for myRIO; the cross-compile toolchain executes on your desktop and compiles your program for the ARM processor on myRIO. The project further specifies a run specification to execute the successfully compiled program.

To import the myRIO project template, from the top menu bar select "File→Import..." to launch the Import wizard:

(a) In the Import panel, select "General→Existing Projects into Workspace" and press Next >.
(b) In the Import Projects pane, press "Select archive file" and browse to

C_Support_for_NI_myRIO_v1.0.zip

Several projects are available for import; the example projects are a useful resource but not needed for this lab, so deselect all example projects, leaving only **C Support for NI myRIO** and **myRIO Template**.

Press Finish to complete the import wizard.

5. **Rename the Template Project:** In the Project Explorer view, right-click on the project **myRIO Template** and select "Rename...". Rename the project **myRIO Hello World**.

6. **Set Model Number:** Set a project-level symbol to specify which model of myRIO you are using. This informs the generated C code about the pin configurations and peripherals. Right-click on the **myRIO Hello World** project and select "Properties" to open project settings. Navigate to "C/C++ General→Paths and Symbols" and open the #Symbols tab to see project-level symbols that are defined across all source files in the project. There are a number of symbols that are built-in for compiler, operating system, and Eclipse settings – you may wish to uncheck "Show built-in values" to

show only symbols you have defined. The symbol `MyRio_1900` is defined by default; delete this symbol and create a new symbol `MyRio_1950` to instruct the compiler to build for the correct hardware version.

To verify the project settings are correct and that the project can be built, from the top menu bar select "Project→Build All". Watch the Console to verify all projects build successfully.

7. **Create a Remote Target:** To view connections to remote systems, use the Remote System Explorer (RSE) perspective. To open this perspective, from the top many bar select "Window→Open Perspective→Other" and choose "Remote System Explorer".

From the top menu bar select "File→New→Other..." to launch the New Connection wizard:

(a) In the Select a wizard panel, select "Remote System Explorer→Connection" and press Next >.
(b) In the Select Remote System Type panel, select "Linux" and press Next >.
(c) In the Remote Linux System Connection panel, enter the IP address of your myRIO into the Host name field and the name "myRIO" into the Connection name field and press Next >.
(d) In the Files panel, choose the configuration "ssh.files" to indicate the filesystem connection protocol. Press Next >.
(e) In the Processes panel, choose the configuration "processes.shell.linux" to indicate the Shell process subsystem for connecting to remote processes. Press Next >.
(f) In the Shells panel, choose the configuration "ssh.shells" to indicate the shell and command protocol. Press Next >.
(g) In the Ssh Terminals panel, choose the configuration ssh.terminals to indicate the terminal protocol.
(h) Press Finish to complete the New Connection wizard.

You should now see your myRIO target appear in the Remote Systems pane (Fig. 3.10).

8. **Connect to myRIO:** From the Remote Systems pane, right-click on your myRIO target and select "Connect". Confirm the IP address, and enter "admin" and leave the password blank. Choose to save the username and password for convenient reconnecting (Fig. 3.11). Press OK to connect.

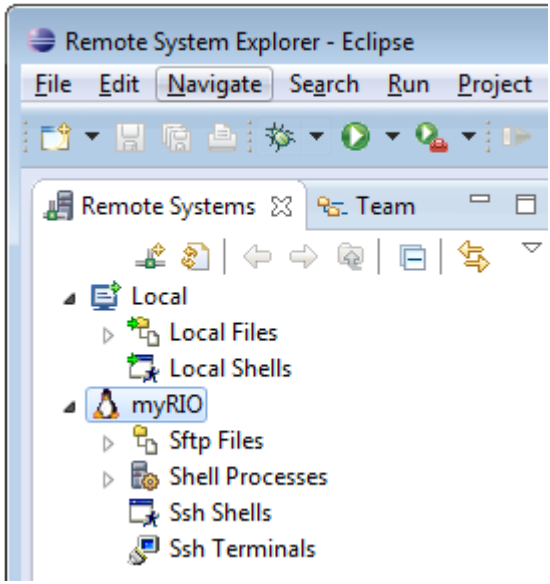


Figure 3.10: Remote System Explorer Perspective.

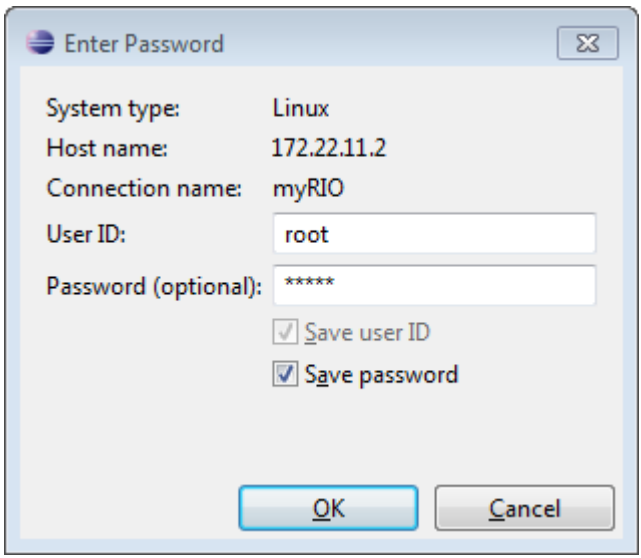


Figure 3.11: Remote system connection configuration. **NOTE:** The username 'admin' should be used instead of 'root'.

You may receive a security warning that the authenticity of the target could not be independently verified (Fig. 3.12). You may disregard this warning.

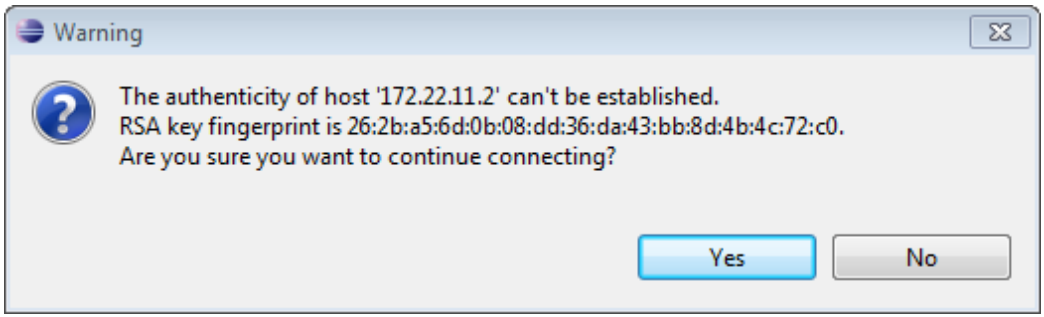


Figure 3.12: An expected security warning that may be disregarded.

When connected, a green arrow should appear over the target icon in the Remote Systems panel. You can verify the target is connected by right-clicking on the target icon – if "Disconnect" appears in place of "Connect" in the context-menu, then your target is connected.

Next, launch a terminal window which will allow you to interact with the Linux shell on myRIO. Expand the myRIO target, right-click on Ssh Terminals, and select "Launch Terminal". A terminal pane should appear in the bottom part of the perspective. Standard output messages are displayed on this terminal, and you may interact with the shell as with any Linux machine. Enter the command

```
uname --all
```

to display information about the Linux operating system on myRIO (Fig. 3.13).

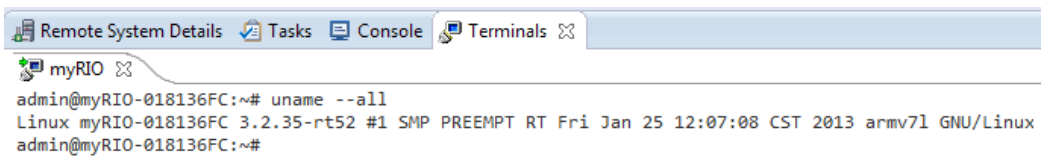


Figure 3.13: Terminal command running on myRIO that displays Linux kernel information.

9. **Transfer the FPGA Fixed-Personality to myRIO:** The FPGA on myRIO may be configured with an FPGA fixed-personality stored in a LabVIEW bitfile (**.lvbitx**) file. In the Remote Systems pane, expand "myRIO→Sftp Files" to see the filesystem on myRIO. Navigate to the folder

/var/local/natinst/

Right-click on the folder "natinst" and select "New→Folder". Name the folder "bitfiles". Right-click on the newly created "bitfiles" folder and select "Export from Project...". Navigate to the source folder

C Support for NI myRIO\source

and check to include NiFpga_myRio[model] Fpga10.lvbitx . No other files or folders should be checked. Verify that the destination folder indicates the "bitfiles" folder. Click Finish to transfer the file, and verify the file now appears in the Remote Systems pane under the "bitfiles" folder.

When your application executes on the target, the myRIO initialization code will open the correct bitfile and program it to the FPGA.

10. **Write a Hello World Message:** Most programming tasks are performed in the C/C++ perspective. To open this perspective, from the top menu bar select "Window→Open Perspective→Other" and choose "C/C++".

Expand the **myRIO Hello World** project and open the file **main.c** Add the statement `printf("Hello, World!\n");` after the comment `/* Your application code goes here.*/`.

To compile the source code, from the top menu bar select "Project→Build Project". The Problems tab at the bottom of your workspace will show the results of the compilation – if there are errors (hopefully just typos), you must resolve them before continuing to the next step.

11. **Create a Debug Configuration:** Debug configurations are used to build your application, download the executable to the remote target, and attach Eclipse to a remote debugging session. Create a new debug configuration by navigating to the top menu bar "Run→Debug Configurations...". In the Debug Configurations dialog, expand C/C++ Remote Application and open the myRIO Template configuration. Configure the debug configuration as follows:

(a) Set the Name to "myRIO Hello World".
(b) In the Main pane:
    i. Select the application to run by pressing Search Project... under C/C++ Application. If your program compiled successfully, the project **myRIO Hello World** will appear in the Program Selection dialog. Select it and press OK.
    ii. For connection profile, ensure "myRIO" is selected.
    iii. Set Remote Absolute File Path to

/home/admin/myRIOHelloWorld

(c) In the Debugger tab, ensure GDB Debugger ensure the GDB debugger is set to

arm-none-linux-gnueabi-gdb.exe

(d) Press Apply followed by Close to save settings and close the configuration window.

12. **Launch a Debug Session:** From the top menu bar, select "Run→Debug Configurations..." to open the Debug Configurations window. In the left navigation pane, select the debug configuration you created in the previous step, and press Debug to launch a debug session. Eclipse may prompt to open the Debug perspective, which contains several useful debugging views. Click Yes to open the Debug perspective.

The Console contains several views. To open the Remote Shell view, press the "Display Selected Console" button  until the console displayed reads "myRIO Hello World [C/C++ Remote Application] Remote Shell".

The debugger will stop at the first line of the `main()` function. To execute the first line, from the top menu bar select "Run→Step Over". The console should display the message, "Hello, World!". To complete execution of your application, from the top menu bar select "Run→Resume".

If the debugger exits successfully and you see the printed message on the console, then your code has been correctly cross-compiled, downloaded to the target, and remotely debugged. Next, you will configure your project to connect to the hardware peripherals on myRIO.

13. **Write an I/O Application:** Modify **main.c** to access the hardware peripherals on myRIO. At the top of the source file, include the myRIO header, and revise

the `main()` function to open turn on an LED. Add the line `extern NiFpga_Session myrio_session;` immediately following the `#include` statements, and add the following lines after your print statement:

```
    /* Set the LED register to 1, turning on LED 1 */
    status = NiFpga_WriteU8(myrio_session, DOLED30, 0x01);

    /* Print error messages (if any) from the DIO write */
    MyRio_PrintStatus(status);
```

The complete program is shown below.

```
#include <stdio.h>
#include "MyRio.h"

extern NiFpga_Session myrio_session;

int main(int argc, char **argv)
{
    NiFpga_Status status;

    status = MyRio_Open();
    if (MyRio_IsNotSuccess(status))
    {
        return status;
    }

    /*
     * Your application code goes here.
     */
    printf("Hello, World!\n");

    /* Set the LED register to 1, turning on LED 1 */
    status = NiFpga_WriteU8(myrio_session, DOLED30, 0x01);

    /* Print error messages (if any) from the DIO write */
    MyRio_PrintStatus(status);

    status = MyRio_Close();

    return status;
}
```

Compile, download, and run your application on the target. If LED 1 illuminates, and you receive no error messages on the console, the program executed correctly. Congratulations on using Eclipse to build your first C application for myRIO!

Hint: *If the Eclipse Problems pane indicates an error that the symbol* `DOLED30` *cannot be resolved, this is actually a problem with the built-in Code Analyzer. See Troubleshooting below for an explanation and simple workaround.*

14. **Share your Feedback:** what did you like about this lab, and what would you change?

3.3.4 Troubleshooting

I receive the compile error, "Cannot run program "arm-none-linux-gnueabi-gcc": Launching failed". Eclipse is unable to find the cross-compile toolchain. The compile toolchain is expected to be enumerated in the system PATH environment variable, which is typically set by your laboratory administrator. If you are the administrator of your build machine, see section §A.3 (Install C & C++ Development Tools) for instructions on how to set the environment variable. If you are unable to change the system or user PATH environment variable, one workaround is to specify the path to the cross-compile toolchain in your project settings:

In the Project Explorer, right-click on **myRIO Project**, and in the context menu select "Properties". Navigate to "C/C++ Build→Settings", and in the Tool Settings tab select "Cross Settings". Set the Prefix field to

```
arm-none-linux-gnueabi-
```

to inform the compiler that the project should be compiled for ARM architectures in the Linux operating system using the GNU open-source compiler. In the Path field, press the Browse... button to navigate to the location of the cross-compile toolchain, which is by default:

C:\Program Files (x86)\National Instruments\Eclipse\toolchain\gcc-4.4-arm\i386\bin

Remove " (x86)" from the path if you are on a 32-bit operating system. Clean and rebuild your project with the new settings. If the error still occurs, consult Eclipse documentation and verify that the compile toolchain has been installed to the National Instruments install directory.

I receive error RSEG1066, "Failed to connect sshd on 'x.x.x.x:22' ". This error occurs if Eclipse cannot connect to myRIO, or an existing session has timed out. Try connecting again in case a timeout has occurred. Double-check the IP address of myRIO and that Eclipse is not blocked by any firewall. Refer to §3.1 (Connect to and Configure myRIO) for details on connecting to myRIO.

My process terminates with error code -61024: This corresponds to a device type mismatch when loading the fixed personality on myRIO. Double-check that the correct symbol `MyRio_1950` is defined for your project.

The Eclipse Problems pane shows an unresolved symbol, but the Console output indicates a successful build: If the Eclipse Problems pane indicates an error that a symbol (such as `DOLED30`) cannot be resolved, this is actually a problem with the built-in Code Analyzer. Open the Console pane to see the output of the cross-compiler – if it indicates the program was built successfully, then you may ignore the unresolved symbol warning in the Problems pane. To prevent the Code Analyzer from falsely reporting this problem, you may disable the feature by right-clicking on your project, selecting Properties, navigating to "C/C++ General → Code Analysis", selecting to Use project settings and unchecking "Syntax and Semantic Errors → Symbol is not resolved".

3.4 Program the myRIO Processor from LabVIEW

These laboratory exercises serve as a tutorial on programming the ARM processor on myRIO using National Instruments LabVIEW. You will build a acceleration indicator application that turns on LEDs when acceleration is exceeded on each axis of the onboard accelerometer.

This lab builds on Lab 3.1: Connect to and Configure myRIO.

3.4.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.9.1 (Introduction to LabVIEW)

3.4.2 Prelab Exercises

There are no prelab exercises for this lab.

3.4.3 Lab Exercises

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW for myRIO Module 2014
- National Instruments LabVIEW Real-Time Module 2014
- National Instruments NI-RIO 13.1
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer. See Lab 3.1: Connect to and Configure myRIO for instructions if you have not done this before.

2. **Launch LabVIEW:** LabVIEW may be found in the Windows Start menu under "National Instruments→LabVIEW National Instruments LabVIEW 2014 →LabVIEW National Instruments LabVIEW 2014 ". Upon launching, you will see the Getting Started dialog which links to recent files, projects, and help documents.

3. **Create a Project:** From the top menu of the Getting Started dialog, select "File→ Create Project..." to launch the Create Project Wizard. Navigate to "Templates→ myRIO→myRIO Project" and select Next.

In the Configure the new project dialog, name your project **Acceleration Threshold** and choose a location to save your project files. Your myRIO should automatically appear under the Select Target pane – if not, first verify that you can see your myRIO in NI MAX, and re-run the wizard.

Select Finish to complete the Create Project Wizard. The Project Explorer will automatically open with your new project. The Project Explorer is used to connect to myRIO and organize VIs that will run on the target processor.

4. **Run the Template VI:** Open the top-level VI that will execute on the processor by navigating in the Project Explorer to "Project Acceleration Indicator.lvproj→ myRIO→Main.vi". A template VI has been created that reads data from the onboard accelerometer and displays it on a graph. On clicking the Run button, the VI will

be deployed to myRIO and run. Verify that the accelerometer data is shown on your screen.

5. **Modify the Template VI:** In the block diagram of **Main.vi**, replicate the code shown in Figs. 3.14 – 3.15:

(a) **LED Express VI** ("myRIO → Onboard → LED" in the palette). The Express VI will open a configuration dialog; uncheck LED3 and press OK to save the configuration.
(b) Three **Greater?** nodes that will be used to compare accelerometer values to a threshold.
(c) Wire the 'x' (top) input of the first **Greater?** node to the x output of the **Accelerometer** VI, and similarly connect the remaining two **Greater?** nodes to the y and z axes.
(d) On the front panel, create a new Numeric Control and label it **Acceleration Threshold (g)**. Set its initial value to 0.5. Right click on the control and from the context menu select "Data Operations → Make Current Value Default". On the block diagram, wire this control to the 'y' (bottom) input of each **Greater?** node.
(e) Wire the output of the x axis comperator to the LED0 input of **LED** VI, and similarly connect the output of the y and z comperators to LED1 and LED2.
(f) On the front panel, create three LED indicators. Label the first **x axis threshold**, and and similarly for the y and z axes. On the block diagram, wire these to the corresponding output of the comparators.

Run the VI to compile your application, download to myRIO, begin execution, and display debugging information to the front panel. You should see the onboard LEDs illuminate when acceleration on the corresponding axis of the onboard acclerometer exceeds the **Acceleration Threshold (g)**.

Congratulations on completing your first LabVIEW application for myRIO! To run your application again, simply press the Run button in the top toolbar.

6. **Share your Feedback:** what did you like about this lab, and what would you change?

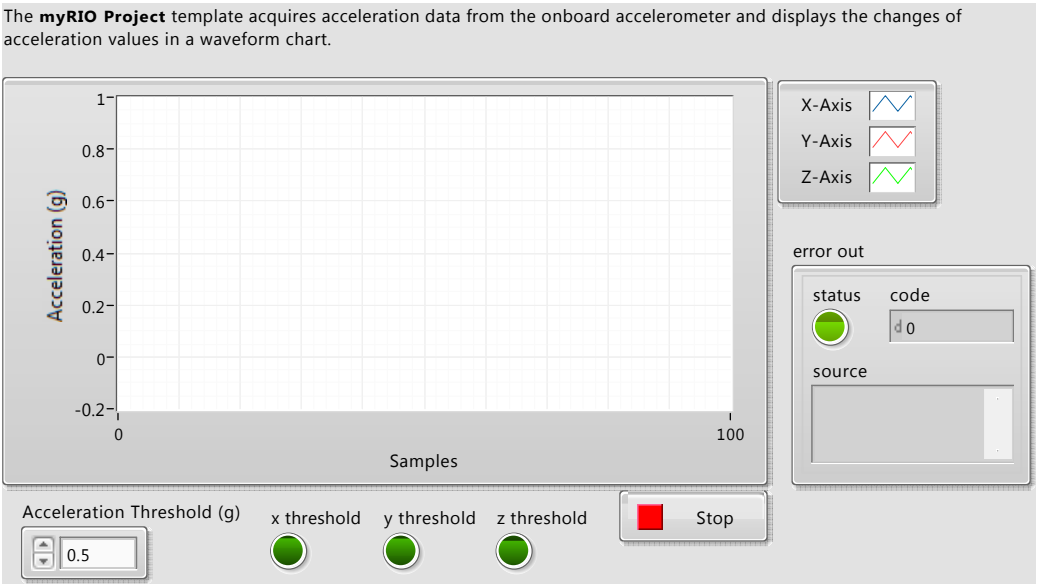


Figure 3.14: Threshold Indicator application front panel.

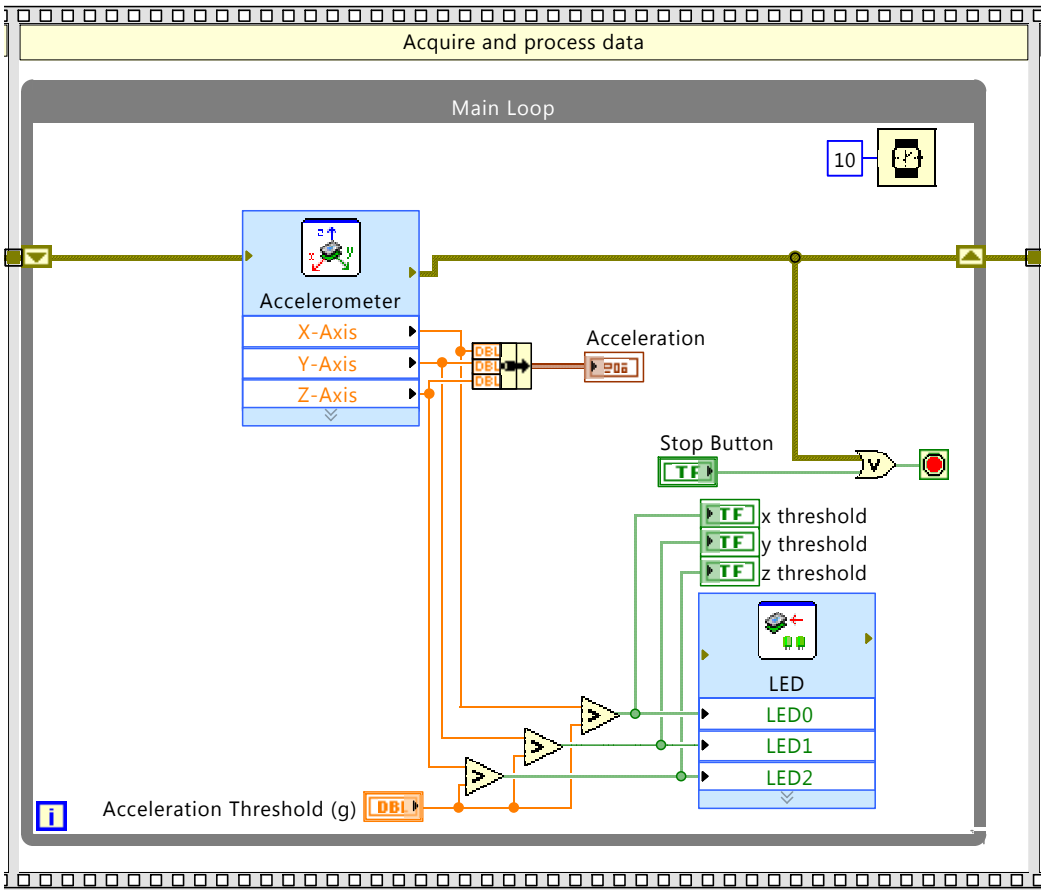


Figure 3.15: Threshold Indicator application block diagram (Main Loop only).

4

Programming Embedded Systems

4.1 Generate Tones in MicroBlaze

Contributing Author: Trung N. Tran

This lab focuses on interrupts. Using the MicroBlaze soft-core processor programmed on myRIO you will write C code to configure timers and generate tones using DIO lines. Your software executes "bare-metal", or in the absence of a scheduling kernel or operating system, and has exclusive access to the microcontroller processor, memory, and peripherals.

These exercises guide you in configuring timed interrupts using memory-mapped registers on an embedded microcontroller. A **timed interrupt** is an interrupt that is generated by a hardware clock, usually periodically. For many, the first experience programming interrupts is challenging: interrupts are documented in verbose documentation, the embedded system being programmed may not have a display or debugging interface, and breaking from the sequential model of traditional programming languages introduces new complexities (Lee, 2006). Inherent in these exercises is the importance of navigating technical documentation to solve a problem.

This lab builds on Lab 3.2: Program MicroBlaze from Xilinx SDK.

4.1.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.3.3 (Interrupts on MicroBlaze)
- §1.3.4 (MicroBlaze and AXI Bus Architecture)
- §1.3.5 (Memory-Mapped Register Interface to Digital IO)
- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 9: Memory Architectures §9.2.1 (Memory Maps)
  - Chapter 9: Memory Architectures §9.2.2 (Register Files)
  - Chapter 10: Input and Output §10.2 (Sequential Software in a Concurrent World)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Embedded System Tools Reference Manual (Xilinx, 2012a)
  - Chapter 9: GNU Compiler Tools, §"Interrupt Handlers", p.133
- LogiCORE IP AXI INTC (Xilinx, 2013b)
  - §Introduction, p.1-6
  - §Register Descriptions, p.13-19
  - §Software Considerations, p.24
- LogiCORE IP AXI Timer (Xilinx, 2012b)
  - §Introduction, p.1-5
  - §Register Data Types and Organization, p.11-15

These documents go beyond the scope of the core concepts of these exercises, and are provided as additional resources.

- MicroBlaze Interrupts and the EDK (Hickok, 2009)

4.1.2 Prelab Exercises

1. In this exercise, you will explore the memory-mapped register interface to the interrupt controller in MicroBlaze.

   (a) Which register must be configured to enable the interrupt controller to interrupt the processor in response to an interrupt request?
   (b) Which register must be configured for the interrupt controller to accept (respond) to an interrupt signal?
   (c) After the processor receives an interrupt request and executes the interrupt service routine, which register identifies enabled and active interrupts?
   (d) At the end of an interrupt service routine, which register should be written to clear the interrupt request?
   (e) Write a single C statement to clear interrupt requests. The statement should clear *only* interrupts which are enabled and active. You may assume the abbreviation of a register name is defined as its memory address.

2. What is the meaning of the `interrupt_handler` compiler attribute?

3. In this exercise, you will explore the memory-mapped register interface to the timer peripheral in MicroBlaze.

   (a) Which register must be configured to enable the Timer0? Which bit(s) must be set?
   (b) Which register holds the value used to set the period of Timer0? How does Timer0 load this value?
   (c) What register must be configured for the Timer0 peripheral to generate interrupt signals? Which bit(s) must be set?
   (d) What register must be configured for Timer0 to run continuously (when it is enabled)? Which bit(s) must be set?

4. Consider the statement which maps `TCSR0` to a memory-mapped register:

```
#define TCSR0 *((volatile unsigned long *)(XPAR_TIMER0_BASEADDR + 0x00))
```

   (a) What is the effect of the `volatile` keyword?
   (b) What is the effect of the typecast `(volatile unsigned long *)` ?
   (c) What is the effect of the outer `*` operand?

5. In this exercise, you will derive timing properties used to generate audio.

(a) An audio tone can be generated by toggling a digital pin at a fixed frequency. This is equivalent to a PWM signal with 50% duty cycle at this frequency – that is, half the time the digital pin is high, the other half, low. If you use a timed interrupt to toggle the pin, and you wish to generate a 440Hz square wave, at what frequency must your interrupt execute?
(b) The MicroBlaze processor clock executes at 50MHz. What is the period of your timed interrupt, in units of processor cycles? Can you exactly generate the desired tone?

6. In this exercise, you will evaluate a circuit for powering a speaker from myRIO.

(a) What is the typical high output voltage of a digital IO line on myRIO?
(b) If a small speaker is modeled as an 8Ω resistor and connected to a unlimited current power supply with the same output voltage you found in 6a, what current will be drawn?
(c) What is the maximum current output of a digital IO line ("sourcing current") on myRIO? Will an 8Ω speaker draw more current than this when connected between a digital IO line and ground?
(d) If the 8Ω speaker is placed in series with a 1kΩ resistor and connected between a digital IO line and ground on myRIO, will the current increase or decrease? Will this meet the electrical specifications of myRIO?

4.1.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\XilinxSDK\TimedIO

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments NI-RIO 13.1
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- Xilinx SDK 14.4
- Xilinx Platform Cable USB II
- SparkFun Electronics COM-09151 Speaker - 0.5W 8Ω
- 1kΩ 0.5W resistor

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer. See Lab 3.1: Connect to and Configure myRIO for instructions if you have not done this before.Program MicroBlaze on the FPGA by using RIO Device Setup. The MicroBlaze FPGA fixed-personality is located at

XilinxSDK\MicroBlaze_myRIO.lvbitx

See Lab 3.2: Program MicroBlaze from Xilinx SDK for instructions if you have not done this before. When the MicroBlaze fixed personality is running, myRIO LED0 will glow on and off.

Connect the USB port of the Xilinx Platform Cable to your desktop, and the JTAG cables to the digital I/O pins shown in Table A.1. When properly connected, the Status light on the Platform Cable will illuminate green.

Connect the resistor between myRIO MXP Connector B DIO8 on myRIO and the positive terminal of the speaker. Connect the negative terminal of the speaker to a myRIO ground. The resistor is a **current-limiting resistor** and ensures the sourcing current of the DIO line is not exceeded.

2. **Launch Xilinx SDK:** Open Xilinx SDK, import the MicroBlaze specification

src\XilinxSDK\microblazeSpec\myRIO.zip

Import the lab template **timedIO.zip**, right-click on the imported project to select the board support package, clean all projects, and build all projects. Use the XMD Console to stop and reset the MicroBlaze processor. See Lab 3.2: Program MicroBlaze from Xilinx SDK for instructions if you have not done this before.

Open the file **timedIO.c**. This is the only file you need to modify to complete this lab. If the program compiles and downloads successfully, you should see console output appear at the bottom of the Project Explorer view. The template code will compile and download to the target without modification, but its functionality is incomplete.

Be sure to use the XMD Console to stop and reset the MicroBlaze processor on your target between executions of your application.

3. **Configure a Periodic Timed Interrupt:** You will generate a 440Hz (fourth octave A) note by generating a square wave of this frequency[1]. Configure a timed interrupt to generate this tone. The body of the corresponding ISR should consist only of a firing counter (useful for debugging), toggling of a DIO line, and acknowledgement of the interrupt. ***Hint:*** *Interrupt enable masks are defined at the top of* ***timedIO.c****.*

   (a) Provide the content your of your main program loop and interrupt service routine.

4. **Starve the Processor:** Increasing the frequency of a timed interrupt leaves less time for the processor to attend to other tasks. In steps, increase the frequency of your timed interrupt until the system exhibits erratic behavior (as seen through your debugger) or becomes unresponsive.

   (a) At what frequency does MicroBlaze exhibit erratic behavior or become unresponsive? What behavior did you observe?

[1] A square wave is characterized by its fundamental frequency, but it contains an infinite number of harmonics. A real system (such as a speaker) will attenuate high-order harmonics.

5. **Share your Feedback:** what did you like about this lab, and what would you change?

4.1.4 Troubleshooting

Refer to Lab 3.2: Program MicroBlaze from Xilinx SDK §3.2.4 (Troubleshooting) for troubleshooting issues in the Xilinx SDK.

4.2 Program an ADC in MicroBlaze

Contributing Author: Trung N. Tran

This lab focuses on programming an ADC on myRIO. The goal is to write C code to configure a timed interrupt for periodic sampling of an analog input.

This lab builds on Lab 4.1: Generate Tones in MicroBlaze.

4.2.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.3.6 (Memory-Mapped Register Interface to the Accelerometer ADC)

4.2.2 Prelab Exercises

1. In this exercise, you will explore the memory-mapped register interface to the accelerometer ADC on myRIO.

(a) Which register enables ADC interrupts to be generated?
(b) Which register should be written to by an ISR to clear the ADC interrupt status?
(c) Which register holds the result of an ADC conversion?
(d) Write a single C statement to begin an ADC conversion on channel 2.
(e) Write a single C statement to block until an ADC conversion is complete.
(f) Write a single C statement to read the result of an ADC conversion into the variable `valueRead`, and another statement to read the index of the channel associated with the conversion into `channelRead`.

2. One way to read from an ADC is to use a loop in the `main()` function to trigger an ADC conversion, wait for completion, read its result, and print the result to the screen.

(a) Is this concurrent or sequential software?
(b) Are processor resources well-utilized by this method?
(c) Is timing explicit in this method? In what ways could you attempt to achieve a fixed sampling rate?

3. A second way to read from an ADC is to use timed interrupts. A periodic timer ISR triggers an ADC conversion, waits for completion, and stores its result in a global variable. The `main()` program loop is tasked with printing the result to the screen.

(a) Is this concurrent or sequential software?
(b) Are processor resources well-utilized by this method?
(c) Is timing explicit in this method? In what ways could you attempt to achieve a fixed sampling rate?

4. A third way to read from an ADC is to use timed interrupts and ADC interrupts. A periodic timer ISR triggers an ADC conversion and returns. When the ADC conversion is complete, an ADC interrupt ISR reads the result and returns. The `main()` program loop is tasked with printing the result to the screen.

(a) Is this concurrent or sequential software?

(b) Are processor resources well-utilized by this method?

(c) Is timing explicit in this method? In what ways could you attempt to achieve a fixed sampling rate?

5. What are some of the major differences between using polling and using interrupts to read from an ADC?

4.2.3 Lab Exercises

For the exercises below, begin with the solution you developed for Lab 4.1: Generate Tones in MicroBlaze.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments NI-RIO 13.1
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- Xilinx SDK 14.4
- Xilinx Platform Cable USB II
- SparkFun Electronics COM-09151 Speaker - 0.5W 8Ω
- 1kΩ 0.5W resistor

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer. See Lab 3.1: Connect to and Configure myRIO for instructions if you have not done this before.Program MicroBlaze on the FPGA by using RIO Device Setup. The MicroBlaze FPGA fixed-personality is located at

XilinxSDK\MicroBlaze_myRIO.lvbitx

See Lab 3.2: Program MicroBlaze from Xilinx SDK for instructions if you have not done this before. When the MicroBlaze fixed personality is running, myRIO LED0 will glow on and off.

Connect the USB port of the Xilinx Platform Cable to your desktop, and the JTAG cables to the digital I/O pins shown in Table A.1. When properly connected, the Status light on the Platform Cable will illuminate green.

2. **Launch Xilinx SDK:** Open Xilinx SDK and open the same workspace you used for Lab 4.1: Generate Tones in MicroBlaze. Clean all projects, and build all projects. See Lab 3.2: Program MicroBlaze from Xilinx SDK for instructions if you have not done this before.

Open the file **timedIO.c**. This is the only file you need to modify to complete this lab.

3. **Poll the ADC:** Configure the `main()` program loop to continuously poll the ADC and display the results to the debug console. You should not configure any interrupts for this exercise.

Hint: *Remember to execute at least four* `asm("nop")` *instructions between writing and then reading the same memory-mapped register.*

(a) Provide the content your `main()` program loop.
(b) When configuring the ADC and `main()` program loop, were there any steps that set the rate at which the ADC is polled, or does your code run as fast as possible?

4. **Use Timed Interrupts to Poll the ADC:** Configure a timed interrupt to poll the ADC every 5 milliseconds. You should not poll the ADC in the `main()` program loop, but instead in a timed interrupt. Leave all `print()` statements in the main program loop.

(a) Provide the content of `main()` that configures the timer ISR, as well as the body of the ISR routine.

5. **Use ADC and Timed Interrupts to Read the ADC:** Configure a timed interrupt to trigger an ADC conversion every 5 milliseconds and the ADC to generate an interrupt when a conversion is complete. You should read the ADC conversion result in the ADC conversion complete interrupt service routine, and not in the `main()` function or timer interrupt service routine.

(a) Provide the content your `main()` function needed to configure the timer and ADC ISRs, as well as the content of the timer and ADC ISRs.

6. **Share your Feedback:** what did you like about this lab, and what would you change?

4.2.4 Troubleshooting

Refer to Lab 3.2: Program MicroBlaze from Xilinx SDK §3.2.4 (Troubleshooting) for troubleshooting issues in the Xilinx SDK.

5 The Cal Climber: CPS Design with the iRobot Create

Cyber-physical systems continue to grow in prevalence and complexity, and their designers face the challenge of managing constraints across multiple knowledge domains such as software, transducers, signal processing, communication and networking, computer architecture, controls, simulation, and modeling of physical processes.

A cyber-physical system is perhaps best characterized by the coupling of its physical processes and embedded computations. Design of these systems requires understanding of these joint dynamics, which is the focus of this chapter and the next one.

This chapter describes a sequence of lab exercises to program the Cal Climber, a version of the iRobot Create interfaced to the NI myRIO, to climb a "hill".

5.1 Navigation Design Requirements

The following are the design requirements for navigation laboratory exercises.

1. **Startup**: When powered on or reset, the robot shall not move until the 'Play' button is pressed.
2. **Run**: The robot shall begin movement the first time the 'Play' button is pressed and continue running until paused or stopped.
   (a) **Ground Orientation**: The ground orientation of your robot is the direction the front of the robot is pointing when it first runs. The ground orientation does not change after subsequent pausing/resuming; only after a power cycle, reprogram or restart of the robot or its embedded controller.
   (b) **Drive**: Your robot shall maintain ground orientation and drive forward while on level ground and clear of obstacles.
   (c) **Obstacle Avoidance**: Your robot shall avoid obstacles.
      i. **Cliff Avoidance**: Your robot shall not fall off of cliffs or edges.
      ii. **Wheel Hazard Avoidance**: Your robot shall avoid one or more wheels losing contact with the ground. If a wheel loses contact with the ground, your robot shall attempt to recover and move around the incident hazard.
      iii. **Object Avoidance**: Your robot shall avoid objects in its path. It is acceptable for your robot to touch objects as long as it immediately changes course in an attempt to avoid.
      iv. **Avoidance Robustness**: Obstacle avoidance must always be satisfied, even if multiple obstacles are encountered simultaneously or in short succession.
      v. **Reorientaton**: After avoiding an obstacle, your robot shall eventually reorient to ground orientation.
3. **Pause/Resume**: The 'Play' button on the top of the robot shall start/resume or pause movement of the robot. At any point the robot is moving, the 'Play' button shall cause it to immediately and completely stop. Subsequently pressing the 'Play' button shall cause the robot to resume operation.
4. **E-Stop**: The 'Advance' button on the top of the robot shall immediately stop all movement of the robot and terminate execution of its controller. The robot shall not move again until it is power cycled, or its embedded controller reprogrammed or reset.

5. **Performance**: When moving, your robot must satisfy the following performance characteristics.
   (a) **Turnabout**: Your robot shall never rotate in place more than 180°.
   (b) **Chattering**: Your robot shall not move erratically or exhibit chattering.
   (c) **Abnormal Termination**: Your robot shall not abnormally terminate execution, with the exception of power or mechanical failure.
   (d) **Obstacle Hugging**: Your robot shall not repeatedly encounter ('hug') an obstacle for the purpose of navigation.
   (e) **Timeliness**: Your robot shall achieve its goals in a timely manner.

Hint: *Your robot does not need to follow a specific path or trajectory, or return to a specific path or trajectory following obstacle avoidance – it need only eventually return to and maintain its original ground orientation.*

5.2 Hill Climb Design Requirements

The following are the design requirements for the Hill Climb laboratory exercises. These requirements are in addition to the above navigation requirements.

1. **Hill Climb:** When an incline is encountered, your robot shall drive uphill towards the top of the incline.
2. **Hill Descend:** When the top of an incline is reached, as indicated by cliffs or impassible obstacles at the top of the incline, your robot shall drive downhill towards the bottom of the incline.
3. **Ground Stop:** After climbing and descending, when the bottom of an incline is reached, your robot shall stop and terminate execution – its final goal is achieved.

5.3 Program CyberSim from Visual Studio

These laboratory exercises serve as a tutorial on programming the CyberSim simulator using C and Microsoft Visual Studio. You will load a template project and verify your application is loaded by the simulator.

There are no prerequisites for this lab.

Figure 5.1: Microsoft Visual Studio.

5.3.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.7 (Equipment: Microsoft Visual Studio)
- §1.10 (Equipment: CyberSim)

5.3.2 Prelab Exercises

There are no prelab exercises for this lab.

5.3.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\C Statechart

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- Microsoft Visual Studio, one of the following:
  - Visual Studio Express 2013 for Desktop (Windows 7 and Windows 8)
  - Visual C++ Express 2010 (Windows XP SP3 and Windows Vista)
- National Instruments LabVIEW Run-Time Engine 2014

1. **Launch Visual Studio Express:** From the Windows Start Menu, launch Visual Studio Express. Depending on your version, the Start Menu shortcut may be named `VS Studio Express 2013 for Desktop` or `Visual C++ Express` .

2. **Load the Template C Statechart:** The C Statechart source code and build settings are maintained by a Visual Studio Solution. From the top menu bar, select "File→Open Project..." and navigate to the downloaded exercises,

src\C Statechart\libstatechart Visual Studio Solution\libstatechart.sln

The solution references all source files needed to compile a statechart DLL for use with CyberSim.

3. **Review the Statechart source:** The Solution Explorer pane is used to navigate files in a solution. Open the source file **C Statechart\irobotNavigationStatechart.c**, the only file you need to modify to change the controller behavior in CyberSim.

4. **Compile:** From the top menu bar, select "Build→Build Solution" to compile the solution. The output directory is automatically set to the CyberSim program folder. Ensure there are no errors in the error list, and following a successful build, check

that the output library was written to the CyberSim binary folder in the downloaded exercise files:

CyberSim\libstatechart.dll

5. **Launch CyberSim and Simulate:** Open CyberSim from the downloaded exercise files:

CyberSim\CyberSim.exe

Open the Configuration tab and in the "statechart" path field, enter **libstatechart.dll**. Press Start to begin a simulation. The simulation will progress, executing your controller at each simulation step.

6. **Stop the Simulation:** Before you can change and recompile your controller, you must stop the simulation to unload **libstatechart.dll**. Press the Stop button to stop the simulation. You do not need to exit CyberSim.

7. **Launch a Debug Session (optional):** Visual Studio can attach a debugger to CyberSim to allow you to debug your control software. Visual Studio will launch CyberSim automatically, using a command-line argument to point the simulator to the compiled library. Close all other instances of CyberSim before beginning a debug session.

In Visual Studio, from the top menu car select "Debug → Start Debugging". The solution should build and launch CyberSim. Press Start to begin a simulation. The debugger will attach when the simulation begins. You may use debugging features such as watch windows and breakpoints to step through your code.

When you are ready to terminate a debugging session, disable all breakpoints and from the Visual Studio top menu bar select "Debug→Continue" to allow CyberSim to continue executing uninterrupted. Return to CyberSim and press Exit to close CyberSim, detatch the Visual Studio Debugger, and close the debugging session.

8. **Close CyberSim** Use the Exit button to terminate CyberSim – do not close the window, as this will terminate CyberSim without unloading your statechart library or the ODE simulator.

Congratulations on using Visual Studio to write a controller for CyberSim!

9. **Share your Feedback:** what did you like about this lab, and what would you change?

5.4 Navigation in C (Simulation)

These exercises guide you in creating a state machine in C using Microsoft Visual Studio. Your state machine will read sensors and drive the iRobot Create with a basic sense of orientation while avoiding obstacles in a simulated environment.

This lab builds on Lab 5.3: Program CyberSim from Visual Studio.

5.4.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 3: Discrete Dynamics §3.3 (Finite-State Machines)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Create Owner's Guide (iRobot, 2006b)
- Create Open Interface Specification (iRobot, 2006a)

5.4.2 Prelab Exercises

1. §1.4.3 (iRobot Create Equipment Exercises) (all exercises). You may skip these if you have already completed them for another laboratory in this chapter.

2. Review the template project provided for the lab exercises. The template code implements a state machine; sketch it.

5.4.3 Lab Exercises

For the exercises below, begin with the same source files you used in Lab 5.3: Program CyberSim from Visual Studio, located in the downloaded exercise files:

src\C Statechart\

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- Microsoft Visual Studio, one of the following:
  - Visual Studio Express 2013 for Desktop (Windows 7 and Windows 8)
  - Visual C++ Express 2010 (Windows XP SP3 and Windows Vista)
- National Instruments LabVIEW Run-Time Engine 2014

1. **Launch Visual Studio Express:** Open Microsoft Visual Studio Express for Desktop and import the solution **libstatchart.sln**. Clean and build the solution. The compiled library is written to the CyberSim folder.

2. **Execute the Template Code:** Launch `CyberSim.exe` . In the Configuration tab, browse for "Statechart" and select **libstatechart.dll**. Press Start to begin a simulation.

3. **Run a Debug Sequence:** In the Configuration tab, select feedback mode "Debug Navigation" and press Debug to run through a sequence of tests that may catch common errors or bugs as you build your solution.

4. **Navigate in Simulation:** Program the virtual robot. Program the robot to avoid obstacles and maintain a basic sense of orientation. You must satisfy all design requirements as specified in §5.1 (Navigation Design Requirements).

   (a) Describe (and sketch) your obstacle avoidance algorithm.
   (b) Did you follow a state machine architecture? If so, which states did you add to the default project?

5. **Share your Feedback:** what did you like about this lab, and what would you change?

5.5 Navigation in C (Deployment)

These exercises guide you in creating a state machine in C using C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition and myRIO. Your state machine will read sensors and drive the iRobot Create with a basic sense of orientation while avoiding obstacles.

This lab builds on Lab 3.3: Program the myRIO Processor from Eclipse and Lab 5.4: Navigation in C (Simulation).

5.5.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 8: Embedded Processors "Circular Buffers" sidebar p.220
  - Chapter 10: Input and Output §10.1.3 (Serial Interfaces)

5.5.2 Prelab Exercises

1. Review the template project provided for the lab exercises.

   (a) Does the template code use polling or interrupts to read the iRobot sensors?
   (b) A software driver for a communication peripheral uses the `xqueue` data type. What is this data type, and for which peripheral is it used?
   (c) The `xqueue` data type encapsulates a buffer whose size must be a power of two. What is the benefit of such a constraint?

5.5.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\C Statechart

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments NI-RIO 13.1
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition
- iRobot Create

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer. See Lab 3.1: Connect to and Configure myRIO for instructions if you have not done this before.

2. **Verify the Robot Setup:** Your iRobot Create is interfaced with a myRIO. Verify the components are correctly wired, as shown in Fig. A.2.

3. **Launch Eclipse:** Open C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition and import the project archive **irobotNavigation.zip**. Double-check that the `MyRio_1950` symbol is set and that the target has been correctly configured. Clean and build all projects. Create a target connection, connect to the target, and launch the Target Console. Create a run configuration for the **irobotNavigation** project. See Lab 3.3: Program the myRIO Processor from Eclipse for instructions if you have not done this before.

4. **Execute the Template Code:** Launch the run configuration and observe the messages printed to the Target Console. Place your robot in a clear space and press the 'Play' button on the top of the robot to toggle between pause and execute modes.

Hint: *The easiest way to revise code and restart your process is for the original process to complete. The template code in your project is designed to finish execution when the 'Advance' button is pressed on the iRobot. When finished executing, press the 'Advance' button on the top of the robot to terminate the process. If you attempt to launch a new process before the old process terminates, you may receive an error that the executable could not be downloaded to the target.*

5. **Navigate the iRobot:** Program the real robot. Program the robot to avoid obstacles and maintain a basic sense of orientation. You must satisfy all design requirements as specified in the previous lab §5.1 (Navigation Design Requirements).

(a) Describe (and sketch) your obstacle avoidance algorithm.
(b) Did you follow a state machine architecture? If so, which states did you add to the default project?

6. **Share your Feedback:** what did you like about this lab, and what would you change?

5.5.4 Troubleshooting

Refer to §3.3.4 (Troubleshooting) for troubleshooting issues with Eclipse for NI Real-Time Linux Targets.

5.6 Hill Climb in C (Simulation)

This lab builds on programming cyber-physical systems, specifically feedback-control. Using Microsoft Visual Studio Express, you will implement a state machine in C that instructs the iRobot Create to navigate to the top of an incline while avoiding cliffs and obstacles in a simulated environment.

This lab builds on Lab 5.4: Navigation in C (Simulation).

5.6.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.6.1 (Introduction to the myRIO Accelerometer)
- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 7: Sensors and Actuators §7.1 (Models of Sensors and Actuators)
  - Chapter 7: Sensors and Actuators §7.2.1 (Measuring Tilt and Acceleration)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Create Owner's Guide (iRobot, 2006b)
- Create Open Interface Specification (iRobot, 2006a)

These documents go beyond the scope of the core concepts of these exercises, and are provided as additional resources.

- Using an Accelerometer for Inclination Sensing (Fisher, 2010)
- Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors (Ozyagcilar, 2011)
- Computational Principles of Mobile Robotics (Dudek and Jenkin, 2000)

5.6.2 Prelab Exercises

1. §1.6.3 (MyRIO Accelerometer Exercises) (all exercises). You may skip these if you have already completed them for another laboratory in this chapter.

5.6.3 Lab Exercises

For the exercises below, begin with the solution you developed for Lab 5.4: Navigation in C (Simulation), located in the downloaded exercise files:

src\C Statechart\

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- Microsoft Visual Studio, one of the following:
  - Visual Studio Express 2013 for Desktop (Windows 7 and Windows 8)
  - Visual C++ Express 2010 (Windows XP SP3 and Windows Vista)
- National Instruments LabVIEW Run-Time Engine 2014

1. **Hill Climb in Simulation:** Program the virtual robot. Using feedback from the accelerometer, your robot should differentiate between an incline and level ground, and when on an incline, orient and drive towards the top. You must satisfy all design requirements as specified in §5.2 (Hill Climb Design Requirements).

(a) Describe (and sketch) your climbing algorithm.
(b) Did you follow a state machine architecture? If so, which states did you add?
(c) How did you account for movement of the robot when reading from the accelerometer? Describe any algorithms or filters used.

2. **Share your Feedback:** what did you like about this lab, and what would you change?

5.7 Hill Climb in C (Deployment)

This lab builds on programming cyber-physical systems, specifically feedback-control. Using C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition, you will implement a state machine in C on myRIO that instructs the iRobot Create to navigate to the top of an incline while avoiding cliffs and obstacles.

This lab builds on Lab 5.5: Navigation in C (Deployment) and Lab 5.6: Hill Climb in C (Simulation).

5.7.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.6.1 (Introduction to the myRIO Accelerometer)
- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 7: Sensors and Actuators §7.1 (Models of Sensors and Actuators)
  - Chapter 7: Sensors and Actuators §7.2.1 (Measuring Tilt and Acceleration)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Create Owner's Guide (iRobot, 2006b)
- Create Open Interface Specification (iRobot, 2006a)

These documents go beyond the scope of the core concepts of these exercises, and are provided as additional resources.

- Using an Accelerometer for Inclination Sensing (Fisher, 2010)
- Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors (Ozyagcilar, 2011)
- Computational Principles of Mobile Robotics (Dudek and Jenkin, 2000)

5.7.2 Prelab Exercises

1. §1.6.3 (MyRIO Accelerometer Exercises) (all exercises). You may skip these if you have already completed them for another laboratory in this chapter.

5.7.3 Lab Exercises

For the exercises below, begin with the solution you developed for Lab 5.6: Hill Climb in C (Simulation), located in the downloaded exercise files:

src\C Statechart\

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments NI-RIO 13.1
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition
- iRobot Create

1. **Hill Climb on iRobot:** Program the real robot. Using feedback from the accelerometer, your robot should differentiate between an incline and level ground, and when on an incline, orient and drive towards the top. You must satisfy all design requirements as specified in §5.2 (Hill Climb Design Requirements).

(a) Describe (and sketch) your climbing algorithm.
(b) Did you follow a state machine architecture? If so, which states did you add?
(c) How did you account for movement of the robot when reading from the accelerometer? Describe any algorithms or filters used.

2. **Share your Feedback:** what did you like about this lab, and what would you change?

5.7.4 Troubleshooting

Refer to §3.3.4 (Troubleshooting) for troubleshooting issues with Eclipse.

5.8 Program CyberSim from LabVIEW

These laboratory exercises serve as a tutorial on programming the CyberSim simulator using Statecharts and LabVIEW. You will load a template project and verify your application is loaded by the simulator.

There are no prerequisites for this lab.

5.8.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.9 (Equipment: LabVIEW)
- §1.10 (Equipment: CyberSim)

5.8.2 Prelab Exercises

There are no prelab exercises for this lab.

5.8.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\LabVIEW\statechart

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW Statecharts Module 2014

1. **Launch LabVIEW:** From the Windows Start Menu, open National Instruments LabVIEW.

2. **Load the Template LabVIEW Statechart:** The LabVIEW Statechart source code and build settings are maintained by a LabVIEW Project. From the top menu bar, select "File→Open Project..." and navigate to the downloaded exercises,

src\LabVIEW\Statechart.lvproj

The project references all source files needed to create a LabVIEW Statechart for use with CyberSim.

3. **Review the Statechart source:** The Project Explorer pane is used to navigate files. Open the source file **Statechart.lvsc\Diagram.vi**, the only file you need to modify to change the controller behavior in CyberSim.

4. **Recompile the Statechart:** Press the "Force Code Generation for this Statechart" button to recompile the Statechart for myRIO.

5. **Verify the Statechart is Executable:** The LabVIEW Statechart file **Statechart.lvsc** is called from a wrapper VI, **Simulation Statechart.vi**. Open this VI from the Project Explorer and ensure the Run arrow is solid white – if a broken gray arrow is shown instead, press the broken arrow to open the errors dialog.

Save **Simulation Statechart.vi** after recompiling the Statechart.

6. **Launch CyberSim and Simulate:** Open CyberSim from the downloaded exercise files:

CyberSim\CyberSim.exe

Open the Configuration tab and in the "statechart" path field, enter

..\src\LabVIEW\statechart\Simulation Statechart.vi

Press Start to begin a simulation. The simulation will progress, executing your controller at each simulation step. You will see your Statechart executing in the Statechart Display tab.

7. **Stop the Simulation:** Before you can change and recompile your controller, you must stop the simulation to unload **Statechart.lvsc**. Press the Stop button to stop the simulation. You do not need to exit CyberSim.

8. **Close CyberSim** Use the Exit button to terminate CyberSim – do not close the window, as this will terminate CyberSim without unloading your statechart or the ODE simulator.

Congratulations on using Visual Studio to write a controller for CyberSim!

9. **Share your Feedback:** what did you like about this lab, and what would you change?

5.9 Navigation in Statecharts (Simulation)

You will design a control algorithm using the Statecharts model of computation. Your task is to design a Statechart that instructs the robot to maintain a basic sense of orientation while avoiding obstacles in a virtual world.

This lab builds on Lab 5.8: Program CyberSim from LabVIEW.

5.9.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.4 (Equipment: iRobot Create)
- §1.10.1 (Introduction to CyberSim)
- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 3: Discrete Dynamics §3.3 (Finite-State Machines)
  - Chapter 5: Composition of State Machines
- Designing Applications with the NI LabVIEW Statechart Module (Washington, 2009)
- Developing Applications with the NI LabVIEW Statechart Module (National Instruments, 2012b)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Create Owner's Guide (iRobot, 2006b)
- Create Open Interface Specification (iRobot, 2006a)
- LabVIEW Statechart Module Help (National Instruments, 2012d)

5.9.2 Prelab Exercises

1. §1.4.3 (iRobot Create Equipment Exercises) (all exercises). You may skip these if you have already completed them for another laboratory in this chapter.

2. How does the Statecharts model of computation differ from finite state machines?

3. In a Statechart, what is the difference between a state and a region?

4. In LabVIEW Statecharts, what is represented by an arrow containing three open or closed cells? What does each cell represent? What does this arrow connect?

5.9.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\LabVIEW\statechart

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW Statecharts Module 2014

The **LabVIEW Statecharts** editor enables developers to program a in the Statecharts model of computation. When when changing default value of a Statechart StateData control, open the Statechart diagram and press the "Force Code Generation for this Statechart" button to recompile the Statechart. 

1. **Open the Project:** From the extracted project files, open the file **Statechart.lvproj** (LabVIEW Project File). It may take a while to load the project. You may receive a warning that dependencies were loaded from a new path; this is normal and may be disregarded.

2. **Execute the Template Code:** Launch `CyberSim.exe` . In the Configuration tab, browse for "Statechart" and select **src\LabVIEW\statechart\Simulation Statechart.vi**. Press Start to begin a simulation.

3. **Run a Debug Sequence:** In the Configuration tab, select feedback mode "Debug Navigation" and press Debug to run through a sequence of tests that may catch common errors or bugs as you build your solution.

4. **Navigate in Simulation:** Program the virtual robot. Your code should reside entirely in the Statechart – you do not need to modify the block diagram of **Simulation Statechart.vi**.

Program the robot to avoid obstacles and maintain a basic sense of orientation. You must satisfy all design requirements as specified below in §5.1 (Navigation Design Requirements).

(a) Provide a description and screenshot of your obstacle avoidance algorithm.

5. **Share your Feedback:** what did you like about this lab, and what would you change?

5.10 Navigation in Statecharts (Deployment)

You will design a control algorithm using the Statecharts model of computation on myRIO, verify in simulation in the LabVIEW Robotics Simulator, and deploy the solution to the iRobot Create. Your task is to design a Statechart that instructs the robot to maintain a basic sense of orientation while avoiding obstacles.

This lab builds on Lab 3.4: Program the myRIO Processor from LabVIEW and Lab 5.9: Navigation in Statecharts (Simulation).

5.10.1 Prelab Reading

There is no prelab reading for this laboratory.

5.10.2 Prelab Exercises

There are no prelab exercises for this laboratory.

5.10.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\LabVIEW\statechart

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW for myRIO Module 2014
- National Instruments LabVIEW Real-Time Module 2014
- National Instruments NI-RIO 13.1
- National Instruments LabVIEW Statecharts Module 2014
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- iRobot Create

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer. See Lab 3.1: Connect to and Configure myRIO for instructions if you have not done this before.

2. **Verify the Robot Setup:** Your iRobot Create is interfaced with a myRIO. Verify the components are correctly wired, as shown in Fig. A.2.

3. **Open the Project:** From the extracted project files, open the file **Statechart.lvproj** (LabVIEW Project File). It may take a while to load the project. You may receive a warning that dependencies were loaded from a new path; this is normal and may be disregarded.

4. **Navigate the iRobot:** Program the real robot. The Statechart you designed in the simulator is configured to deploy on your embedded target. In the Project Explorer, navigate to "myRIO→Statechart" and open **Diagram.vi**. Press the "Force Code Generation for this Statechart" button to recompile the Statechart for myRIO.

Open **Target.vi**. This is the main VI for the myRIO and is responsible for sampling the accelerometer, reading the iRobot sensors, and transmitting drive commands. You code should reside entirely in the Statechart – you do not need to modify any others VIs. A lowpass filter has been provided to smooth the accelerometer signal.

Run **Target.vi** to deploy your Statechart to your target.

Program the robot to avoid obstacles and maintain a basic sense of orientation. You must satisfy all design requirements as specified in the previous lab §5.1 (Navigation Design Requirements).

(a) Did your Statechart require modification to work on the physical target? If so, what modifications did you make?

(b) How would you compare programming in Statecharts to programming in a traditional programming language such as C?

5. **Share your Feedback:** what did you like about this lab, and what would you change?

5.10.4 Troubleshooting

I receive the error "iRobot Navigation Target loaded with errors and was closed" when attempting to run on target: This occurs when cached generated code becomes corrupt. To resolve the issue, open the LabVIEW configuration file from the LabVIEW program folder, <LabVIEW>\LabVIEW.ini . Under the `[LabVIEW]` section add the line

```
useCacheForDeployment=False
```

Save and restart LabVIEW.

Alternatively, you may resolve the issue by renaming your Statechart.

I receive the error "LabVIEW: Resource not found. An error occurred loading VI." This occurs the custom display element of a Statechart becomes corrupt. The custom data display is not used by this lab and can be restored to resolve the issue. From the downloaded project files, open

src\LabVIEW\statechart\Restore Corrupt Statechart\Restore Corrupt Statechart.vi

Follow the instructions on the VI to restore your Statechart.

5.11 Hill Climb in Statecharts (Simulation)

This lab builds on the concept of model-based design for cyber-physical systems and focuses on feedback-control. Your goal in this lab is to implement a Statechart that instructs the iRobot Create to navigate to the top of an incline while avoiding cliffs and obstacles in a simulated environment.

This lab builds on Lab 5.9: Navigation in Statecharts (Simulation).

5.11.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.6.1 (Introduction to the myRIO Accelerometer).
- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 7: Sensors and Actuators §7.1 (Models of Sensors and Actuators)
  - Chapter 7: Sensors and Actuators §7.2.1 (Measuring Tilt and Acceleration)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Create Owner's Guide (iRobot, 2006b)
- Create Open Interface Specification (iRobot, 2006a)

These documents go beyond the scope of the core concepts of these exercises, and are provided as additional resources.

- Using an Accelerometer for Inclination Sensing (Fisher, 2010)
- Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors (Ozyagcilar, 2011)
- Computational Principles of Mobile Robotics (Dudek and Jenkin, 2000)

5.11.2 Prelab Exercises

1. §1.6.3 (MyRIO Accelerometer Exercises) (all exercises). You may skip these if you have already completed them for another laboratory in this chapter.

5.11.3 Lab Exercises

For the exercises below, begin with the solution you developed for Lab 5.9: Navigation in Statecharts (Simulation), located in the downloaded exercise files:

src\LabVIEW\statechart

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW Statecharts Module 2014

1. **Hill Climb in Simulation:** Program the virtual robot. Using feedback from the accelerometer, your robot should differentiate between an incline and level ground, and when on an incline, orient and drive towards the top. You must satisfy all design requirements as specified in §5.2 (Hill Climb Design Requirements).

(a) Provide a screenshot of your climbing algorithm.
(b) Which states did you add from the previous lab?

2. **Share your Feedback:** what did you like about this lab, and what would you change?

5.12 Hill Climb in Statecharts (Deployment)

This lab builds on the concept of model-based design for cyber-physical systems and focuses on feedback-control. Your goal in this lab is to implement a Statechart on myRIO that instructs the iRobot Create to navigate to the top of an incline while avoiding cliffs and obstacles.

This lab builds on Lab 5.10: Navigation in Statecharts (Deployment) and Lab 5.11: Hill Climb in Statecharts (Simulation).

5.12.1 Prelab Reading

There is no prelab reading for this laboratory.

5.12.2 Prelab Exercises

There are no prelab exercises for this laboratory.

5.12.3 Lab Exercises

For the exercises below, begin with the solution you developed for Lab 5.10: Navigation in Statecharts (Deployment); be sure to review the background from this lab, as it underscores important differences between the simulator and the real robot.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW for myRIO Module 2014
- National Instruments LabVIEW Real-Time Module 2014
- National Instruments NI-RIO 13.1
- National Instruments LabVIEW Statecharts Module 2014
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- iRobot Create

1. **Hill Climb on the iRobot:** Program the real robot. Using feedback from the accelerometer, your robot should differentiate between an incline and level ground, and when on an incline, orient and drive towards the top. You must satisfy all design requirements as specified in §5.2 (Hill Climb Design Requirements).

(a) Did your Statechart require modification to work on the physical target? If so, what modifications did you make?
(b) How did you account for movement of the robot when reading from the accelerometer? Describe any algorithms or filters used.

2. **Share your Feedback:** what did you like about this lab, and what would you change?

5.12.4 Troubleshooting

Refer to Lab 5.10: Navigation in Statecharts (Deployment) §5.10.4 (Troubleshooting) for troubleshooting issues with the robot configuration.

6 The Cal Klimber: CPS Design with the Kobuki

This chapter describes a sequence of lab exercises to program the *Cal Klimber*, a version of the iClebo Kobuki interfaced to the NI myRIO, to climb a "hill". It mirrors the exercises described in Chapter 5 but with a different robotic platform.

6.1 Navigation Design Requirements

We refer to the button "B0" on the Kobuki as the "Play" button. For CyberSim, a separate "Play" button has been added on top of the Kobuki top view which maps to the "B0" button.

The following are the design requirements for navigation laboratory exercises.

1. **Startup**: When powered on or reset, the robot shall not move until the 'Play' button is pressed.
2. **Run**: The robot shall begin movement the first time the 'Play' button is pressed and continue running until paused or stopped.
   (a) **Ground Orientation**: The ground orientation of your robot is the direction the front of the robot is pointing when it first runs. The ground orientation does not change after subsequent pausing/resuming; only after a power cycle, reprogram or restart of the robot or its embedded controller.
   (b) **Drive**: Your robot shall maintain ground orientation and drive forward while on level ground and clear of obstacles.
   (c) **Obstacle Avoidance**: Your robot shall avoid obstacles.
      i. **Cliff Avoidance**: Your robot shall not fall off of cliffs or edges.
      ii. **Wheel Hazard Avoidance**: Your robot shall avoid one or more wheels losing contact with the ground. If a wheel loses contact with the ground, your robot shall attempt to recover and move around the incident hazard.
      iii. **Object Avoidance**: Your robot shall avoid objects in its path. It is acceptable for your robot to touch objects as long as it immediately changes course in an attempt to avoid.
      iv. **Avoidance Robustness**: Obstacle avoidance must always be satisfied, even if multiple obstacles are encountered simultaneously or in short succession.
      v. **Reorientaton**: After avoiding an obstacle, your robot shall eventually reorient to ground orientation.
3. **Pause/Resume**: The 'Play' button on the top of the robot shall start/resume or pause movement of the robot. At any point the robot is moving, the 'Play' button shall cause it to immediately and completely stop. Subsequently pressing the 'Play' button shall cause the robot to resume operation.
4. **Performance**: When moving, your robot must satisfy the following performance characteristics.

(a) **Turnabout**: Your robot shall never rotate in place more than 180°.
(b) **Chattering**: Your robot shall not move erratically or exhibit chattering.
(c) **Abnormal Termination**: Your robot shall not abnormally terminate execution, with the exception of power or mechanical failure.
(d) **Obstacle Hugging**: Your robot shall not repeatedly encounter ('hug') an obstacle for the purpose of navigation.
(e) **Timeliness**: Your robot shall achieve its goals in a timely manner.

Hint: *Your robot does not need to follow a specific path or trajectory, or return to a specific path or trajectory following obstacle avoidance – it need only eventually return to and maintain its original ground orientation.*

6.2 Hill Climb Design Requirements

The following are the design requirements for the Hill Climb laboratory exercises. These requirements are in addition to the above navigation requirements.

1. **Hill Climb:** When an incline is encountered, your robot shall drive uphill towards the top of the incline.
2. **Hill Descend:** When the top of an incline is reached, as indicated by cliffs or impassible obstacles at the top of the incline, your robot shall drive downhill towards the bottom of the incline.
3. **Ground Stop:** After climbing and descending, when the bottom of an incline is reached, your robot shall stop and terminate execution – its final goal is achieved.

6.3 Program CyberSim from Visual Studio

These laboratory exercises serve as a tutorial on programming the CyberSim simulator using C and Microsoft Visual Studio. You will load a template project and verify your application is loaded by the simulator.

There are no prerequisites for this lab.

Figure 6.1: Microsoft Visual Studio.

6.3.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.7 (Equipment: Microsoft Visual Studio)
- §1.10 (Equipment: CyberSim)

6.3.2 Prelab Exercises

There are no prelab exercises for this lab.

6.3.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\C Statechart

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- Microsoft Visual Studio, one of the following:
  - Visual Studio Express 2013 for Desktop (Windows 7 and Windows 8)
  - Visual C++ Express 2010 (Windows XP SP3 and Windows Vista)
- National Instruments LabVIEW Run-Time Engine 2014

1. **Launch Visual Studio Express:** From the Windows Start Menu, launch Visual Studio Express. Depending on your version, the Start Menu shortcut may be named `VS Studio Express 2013 for Desktop` or `Visual C++ Express`.

2. **Load the Template C Statechart:** The C Statechart source code and build settings are maintained by a Visual Studio Solution. From the top menu bar, select "File→Open Project..." and navigate to the downloaded exercises,

src\C Statechart\Visual Studio Solution\libstatechart.sln

The solution references all source files needed to compile a statechart DLL for use with CyberSim.

3. **Review the Statechart source:** The Solution Explorer pane is used to navigate files in a solution. Open the source file **C Statechart\kobukiNavigationStatechart.c**, the only file you need to modify to change the controller behavior in CyberSim.

4. **Compile:** From the top menu bar, select "Build→Build Solution" to compile the solution. The output directory is automatically set to the CyberSim program folder. Ensure there are no errors in the error list, and following a successful build, check

that the output library was written to the CyberSim binary folder in the downloaded exercise files:

CyberSim\libstatechart.dll

5. **Launch CyberSim and Simulate:** Open CyberSim from the downloaded exercise files:

CyberSim\CyberSim.exe

Open the Configuration tab and in the "statechart" path field, enter **libstatechart.dll**. Press Start to begin a simulation. The simulation will progress, executing your controller at each simulation step.

6. **Stop the Simulation:** Before you can change and recompile your controller, you must stop the simulation to unload **libstatechart.dll**. Press the Stop button to stop the simulation. You do not need to exit CyberSim.

7. **Launch a Debug Session (optional):** Visual Studio can attach a debugger to CyberSim to allow you to debug your control software. Visual Studio will launch CyberSim automatically, using a command-line argument to point the simulator to the compiled library. Close all other instances of CyberSim before beginning a debug session.

In Visual Studio, from the top menu car select "Debug → Start Debugging". The solution should build and launch CyberSim. Press Start to begin a simulation. The debugger will attach when the simulation begins. You may use debugging features such as watch windows and breakpoints to step through your code.

When you are ready to terminate a debugging session, disable all breakpoints and from the Visual Studio top menu bar select "Debug→Continue" to allow CyberSim to continue executing uninterrupted. Return to CyberSim and press Exit to close CyberSim, detatch the Visual Studio Debugger, and close the debugging session.

8. **Close CyberSim** Use the Exit button to terminate CyberSim – do not close the window, as this will terminate CyberSim without unloading your statechart library or the ODE simulator.

Congratulations on using Visual Studio to write a controller for CyberSim!

9. **Share your Feedback:** what did you like about this lab, and what would you change?

6.4 Navigation in C (Simulation)

These exercises guide you in creating a state machine in C using Microsoft Visual Studio. Your state machine will read sensors and drive the Kobuki with a basic sense of orientation while avoiding obstacles in a simulated environment.

This lab builds on Lab 6.3: Program CyberSim from Visual Studio.

6.4.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 3: Discrete Dynamics §3.3 (Finite-State Machines)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Kobuki Documentation (YujinRobot, 2014b)

6.4.2 Prelab Exercises

1. §1.5.3 (Kobuki Equipment Exercises) (all exercises). You may skip these if you have already completed them for another laboratory in this chapter.

2. Review the template project provided for the lab exercises. The template code implements a state machine; sketch it.

6.4.3 Lab Exercises

For the exercises below, begin with the same source files you used in Lab 6.3: Program CyberSim from Visual Studio, located in the downloaded exercise files:

src\C Statechart\

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- Microsoft Visual Studio, one of the following:
  - Visual Studio Express 2013 for Desktop (Windows 7 and Windows 8)
  - Visual C++ Express 2010 (Windows XP SP3 and Windows Vista)
- National Instruments LabVIEW Run-Time Engine 2014

1. **Launch Visual Studio Express:** Open Microsoft Visual Studio Express for Desktop and import the solution **libstatchart.sln**. Clean and build the solution. The compiled library is written to the CyberSim folder.

2. **Execute the Template Code:** Launch `CyberSim.exe` . In the Configuration tab, browse for "Statechart" and select **libstatechart.dll**. Press Start to begin a simulation.

3. **Run a Debug Sequence:** In the Configuration tab, select feedback mode "Debug Navigation" and press Check Grade to run through a sequence of tests that may catch common errors or bugs as you build your solution.

4. **Navigate in Simulation:** Program the virtual robot. Program the robot to avoid obstacles and maintain a basic sense of orientation. You must satisfy all design requirements as specified in §6.1 (Navigation Design Requirements).

   (a) Describe (and sketch) your obstacle avoidance algorithm.
   (b) Did you follow a state machine architecture? If so, which states did you add to the default project?

5. **Share your Feedback:** what did you like about this lab, and what would you change?

6.5 Navigation in C (Deployment)

These exercises guide you in creating a state machine in C using C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition and myRIO. Your state machine will read sensors and drive the Kobuki with a basic sense of orientation while avoiding obstacles.

This lab builds on Lab 3.3: Program the myRIO Processor from Eclipse and Lab 6.4: Navigation in C (Simulation).

6.5.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 10: Input and Output §10.1.3 (Serial Interfaces)

6.5.2 Prelab Exercises

1. Review the template project provided for the lab exercises.

   (a) Does the template code use polling or interrupts to read the Kobuki sensors? Is this the same mechanism used by the Kobuki protocol?
   (b) You want the Kobuki to continue executing a single command for 5 seconds. How might you prevent it from shutting down mid-action?

6.5.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\C Statechart

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments NI-RIO 13.1
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition
- iClebo Kobuki

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer. See Lab 3.1: Connect to and Configure myRIO for instructions if you have not done this before.

2. **Verify the Robot Setup:** Your iRobot Create is interfaced with a myRIO. Verify the components are correctly wired, as shown in Fig. A.2.

3. **Launch Eclipse:** Open C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition and import the project archive **src\kobukiNavigation.zip**. Double-check that the `MyRio_1900` symbol is set and that the target has been correctly configured. Clean and build all projects. Create a target connection, connect to the target, and launch the Target Console. Create a run configuration for the **irobotNavigation** project. See Lab 3.3: Program the myRIO Processor from Eclipse for instructions if you have not done this before.

4. **Execute the Template Code:** Launch the run configuration and observe the messages printed to the Target Console. Place your robot in a clear space and press the 'Play' button on the top of the robot to toggle between pause and execute modes.

Hint: *The easiest way to revise code and restart your process is for the original process to complete. The template code in your project is designed to finish execution when the 'Advance' button is pressed on the Kobuki. When finished executing, press the 'Advance' button on the top of the robot to terminate the process. If you attempt to launch a new process before the old process terminates, you may receive an error that the executable could not be downloaded to the target.*

5. **Navigate the Kobuki:** Program the real robot. Program the robot to avoid obstacles and maintain a basic sense of orientation. You must satisfy all design requirements as specified in the previous lab §6.1 (Navigation Design Requirements).

If you have previously programmed the robot and tested it with CyberSim, remember to import your code from Visual Studio (as you did in Sec. 6.4) to Eclipse.

(a) Describe (and sketch) your obstacle avoidance algorithm.
(b) Did you follow a state machine architecture? If so, which states did you add to the default project?

6. **Share your Feedback:** what did you like about this lab, and what would you change?

6.5.4 Troubleshooting

Refer to §3.3.4 (Troubleshooting) for troubleshooting issues with Eclipse for NI Real-Time Linux Targets.

6.6 Hill Climb in C (Simulation)

This lab builds on programming cyber-physical systems, specifically feedback-control. Using Microsoft Visual Studio Express, you will implement a state machine in C that instructs the Kobuki to navigate to the top of an incline while avoiding cliffs and obstacles in a simulated environment.

This lab builds on Lab 6.4: Navigation in C (Simulation).

6.6.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.6.1 (Introduction to the myRIO Accelerometer)
- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 7: Sensors and Actuators §7.1 (Models of Sensors and Actuators)
  - Chapter 7: Sensors and Actuators §7.2.1 (Measuring Tilt and Acceleration)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Kobuki Documentation (YujinRobot, 2014b)

These documents go beyond the scope of the core concepts of these exercises, and are provided as additional resources.

- Using an Accelerometer for Inclination Sensing (Fisher, 2010)
- Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors (Ozyagcilar, 2011)
- Computational Principles of Mobile Robotics (Dudek and Jenkin, 2000)

6.6.2 Prelab Exercises

1. §1.6.3 (MyRIO Accelerometer Exercises) (all exercises). You may skip these if you have already completed them for another laboratory in this chapter.

6.6.3 Lab Exercises

For the exercises below, begin with the solution you developed for Lab 6.4: Navigation in C (Simulation), located in the downloaded exercise files:

src\C Statechart\

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- Microsoft Visual Studio, one of the following:
  - Visual Studio Express 2013 for Desktop (Windows 7 and Windows 8)
  - Visual C++ Express 2010 (Windows XP SP3 and Windows Vista)
- National Instruments LabVIEW Run-Time Engine 2014

1. **Hill Climb in Simulation:** Program the virtual robot. Using feedback from the accelerometer, your robot should differentiate between an incline and level ground, and when on an incline, orient and drive towards the top. You must satisfy all design requirements as specified in §6.2 (Hill Climb Design Requirements).

(a) Describe (and sketch) your climbing algorithm.
(b) Did you follow a state machine architecture? If so, which states did you add?
(c) How did you account for movement of the robot when reading from the accelerometer? Describe any algorithms or filters used.

2. **Share your Feedback:** what did you like about this lab, and what would you change?

6.7 Hill Climb in C (Deployment)

This lab builds on programming cyber-physical systems, specifically feedback-control. Using C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition, you will implement a state machine in C on myRIO that instructs the Kobuki to navigate to the top of an incline while avoiding cliffs and obstacles.

This lab builds on Lab 6.5: Navigation in C (Deployment) and Lab 6.6: Hill Climb in C (Simulation).

6.7.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.6.1 (Introduction to the myRIO Accelerometer)
- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 7: Sensors and Actuators §7.1 (Models of Sensors and Actuators)
  - Chapter 7: Sensors and Actuators §7.2.1 (Measuring Tilt and Acceleration)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Kobuki Documentation (YujinRobot, 2014b)

These documents go beyond the scope of the core concepts of these exercises, and are provided as additional resources.

- Using an Accelerometer for Inclination Sensing (Fisher, 2010)
- Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors (Ozyagcilar, 2011)
- Computational Principles of Mobile Robotics (Dudek and Jenkin, 2000)

6.7.2 Prelab Exercises

1. §1.6.3 (MyRIO Accelerometer Exercises) (all exercises). You may skip these if you have already completed them for another laboratory in this chapter.

6.7.3 Lab Exercises

For the exercises below, begin with the solution you developed for Lab 6.6: Hill Climb in C (Simulation), located in the downloaded exercise files:

src\C Statechart\

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments NI-RIO 13.1
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition
- iClebo Kobuki

1. **Hill Climb on Kobuki:** Program the real robot. Using feedback from the accelerometer, your robot should differentiate between an incline and level ground, and when on an incline, orient and drive towards the top. You must satisfy all design requirements as specified in §6.2 (Hill Climb Design Requirements).

(a) Describe (and sketch) your climbing algorithm.
(b) Did you follow a state machine architecture? If so, which states did you add?
(c) How did you account for movement of the robot when reading from the accelerometer? Describe any algorithms or filters used.

2. **Share your Feedback:** what did you like about this lab, and what would you change?

6.7.4 Troubleshooting

Refer to §3.3.4 (Troubleshooting) for troubleshooting issues with Eclipse.

6.8 Program CyberSim from LabVIEW

These laboratory exercises serve as a tutorial on programming the CyberSim simulator using Statecharts and LabVIEW. You will load a template project and verify your application is loaded by the simulator.

There are no prerequisites for this lab.

6.8.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.9 (Equipment: LabVIEW)
- §1.10 (Equipment: CyberSim)

6.8.2 Prelab Exercises

There are no prelab exercises for this lab.

6.8.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\LabVIEW\statechart

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW Statecharts Module 2014

1. **Launch LabVIEW:** From the Windows Start Menu, open National Instruments LabVIEW.

2. **Load the Template LabVIEW Statechart:** The LabVIEW Statechart source code and build settings are maintained by a LabVIEW Project. From the top menu bar, select "File→Open Project..." and navigate to the downloaded exercises,

src\LabVIEW\Statechart.lvproj

The project references all source files needed to create a LabVIEW Statechart for use with CyberSim.

3. **Review the Statechart source:** The Project Explorer pane is used to navigate files. Open the source file **Statechart.lvsc\Diagram.vi**, the only file you need to modify to change the controller behavior in CyberSim.

4. **Recompile the Statechart:** Press the "Force Code Generation for this Statechart" button to recompile the Statechart for myRIO.

5. **Verify the Statechart is Executable:** The LabVIEW Statechart file **Statechart.lvsc** is called from a wrapper VI, **Simulation Statechart.vi**. Open this VI from the Project Explorer and ensure the Run arrow is solid white – if a broken gray arrow is shown instead, press the broken arrow to open the errors dialog.

Save **Simulation Statechart.vi** after recompiling the Statechart.

6. **Launch CyberSim and Simulate:** Open CyberSim from the downloaded exercise files:

CyberSim\CyberSim.exe

Open the Configuration tab and in the "statechart" path field, enter

..\src\LabVIEW\statechart\Simulation Statechart.vi

Press Start to begin a simulation. You will need to press the 'Play' buttong to start the controller on the Kobuki. The simulation will progress, executing your controller at each simulation step. You will see your Statechart executing in the Statechart Display tab.

7. **Stop the Simulation:** Before you can change and recompile your controller, you must stop the simulation to unload **Statechart.lvsc**. Press the Stop button to stop the simulation. You do not need to exit CyberSim.

8. **Close CyberSim** Use the Exit button to terminate CyberSim – do not close the window, as this will terminate CyberSim without unloading your statechart or the ODE simulator.

Congratulations on using LabVIEW to write a controller for CyberSim!

9. **Share your Feedback:** what did you like about this lab, and what would you change?

6.9 Navigation in Statecharts (Simulation)

You will design a control algorithm using the Statecharts model of computation. Your task is to design a Statechart that instructs the robot to maintain a basic sense of orientation while avoiding obstacles in a virtual world.

This lab builds on Lab 6.8: Program CyberSim from LabVIEW.

6.9.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.5 (Equipment: iClebo Kobuki)
- §1.10.1 (Introduction to CyberSim)
- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 3: Discrete Dynamics §3.3 (Finite-State Machines)
  - Chapter 5: Composition of State Machines
- Designing Applications with the NI LabVIEW Statechart Module (Washington, 2009)
- Developing Applications with the NI LabVIEW Statechart Module (National Instruments, 2012b)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Kobuki Documentation (YujinRobot, 2014b)
- LabVIEW Statechart Module Help (National Instruments, 2012d)

6.9.2 Prelab Exercises

1. §1.5.3 (Kobuki Equipment Exercises) (all exercises). You may skip these if you have already completed them for another laboratory in this chapter.

2. How does the Statecharts model of computation differ from finite state machines?

3. In a Statechart, what is the difference between a state and a region?

4. In LabVIEW Statecharts, what is represented by an arrow containing three open or closed cells? What does each cell represent? What does this arrow connect?

6.9.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\LabVIEW\statechart

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW Statecharts Module 2014

The **LabVIEW Statecharts** editor enables developers to program a in the Statecharts model of computation. When when changing default value of a Statechart StateData control, open the Statechart diagram and press the "Force Code Generation for this Statechart" button to recompile the Statechart. 

1. **Open the Project:** From the extracted project files, open the file **Statechart.lvproj** (LabVIEW Project File). It may take a while to load the project. You may receive a warning that dependencies were loaded from a new path; this is normal and may be disregarded.

Review the steps listed in Sec. 6.8.3 to compile the Statechart.

2. **Execute the Template Code:** Launch `CyberSim.exe` . In the Configuration tab, browse for "Statechart" and select **src\LabVIEW\statechart\Simulation Statechart.vi**. Press Start to begin a simulation.

3. **Run a Debug Sequence:** In the Configuration tab, select feedback mode "Debug Navigation" and press Debug to run through a sequence of tests that may catch common errors or bugs as you build your solution.

4. **Navigate in Simulation:** Program the virtual robot. Your code should reside entirely in the Statechart – you do not need to modify the block diagram of **Simulation Statechart.vi**.

Program the robot to avoid obstacles and maintain a basic sense of orientation. You must satisfy all design requirements as specified below in §6.1 (Navigation Design Requirements).

(a) Provide a description and screenshot of your obstacle avoidance algorithm.

5. **Share your Feedback:** what did you like about this lab, and what would you change?

6.10 Navigation in Statecharts (Deployment)

You will design a control algorithm using the Statecharts model of computation on myRIO, verify in simulation in the LabVIEW Robotics Simulator, and deploy the solution to the Kobuki. Your task is to design a Statechart that instructs the robot to maintain a basic sense of orientation while avoiding obstacles.

This lab builds on Lab 3.4: Program the myRIO Processor from LabVIEW and Lab 6.9: Navigation in Statecharts (Simulation).

6.10.1 Prelab Reading

There is no prelab reading for this laboratory.

6.10.2 Prelab Exercises

There are no prelab exercises for this laboratory.

6.10.3 Lab Exercises

The following exercises reference files are distributed with this courseware. Once downloaded and extracted, navigate to the folder

src\LabVIEW\statechart

to locate files referenced by these exercises.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW for myRIO Module 2014
- National Instruments LabVIEW Real-Time Module 2014
- National Instruments NI-RIO 13.1
- National Instruments LabVIEW Statecharts Module 2014
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- iClebo Kobuki

1. **Setup myRIO:** Ensure myRIO is powered and connected to your computer. See Lab 3.1: Connect to and Configure myRIO for instructions if you have not done this before.

2. **Verify the Robot Setup:** Your Kobuki is interfaced with a myRIO. Verify that the components are correctly wired, as shown in Fig. A.4.

3. **Open the Project:** From the extracted project files, open the file **Statechart.lvproj** (LabVIEW Project File). It may take a while to load the project. You may receive a warning that dependencies were loaded from a new path; this is normal and may be disregarded.

4. **Navigate the Kobuki:** Program the real robot. The Statechart you designed in the simulator is configured to deploy on your embedded target. In the Project Explorer, navigate to "myRIO→Statechart" and open **Diagram.vi**. Press the "Force Code Generation for this Statechart" button to recompile the Statechart for myRIO.

Open **Target.vi**. This is the main VI for the myRIO and is responsible for sampling the accelerometer, reading the robot's sensors, and transmitting drive commands. You code should reside entirely in the Statechart – you do not need to modify any others VIs. A lowpass filter has been provided to smooth the accelerometer signal.

Run **Target.vi** to deploy your Statechart to your target.

Program the robot to avoid obstacles and maintain a basic sense of orientation. You must satisfy all design requirements as specified in the previous lab §6.1 (Navigation Design Requirements).

(a) Did your Statechart require modification to work on the physical target? If so, what modifications did you make?

(b) How would you compare programming in Statecharts to programming in a traditional programming language such as C?

5. **Share your Feedback:** what did you like about this lab, and what would you change?

6.10.4 Troubleshooting

I receive the error "Kobuki Navigation Target loaded with errors and was closed" when attempting to run on target: This occurs when cached generated code becomes corrupt. To resolve the issue, open the LabVIEW configuration file from the LabVIEW program folder, <LabVIEW>\LabVIEW.ini . Under the `[LabVIEW]` section add the line

```
useCacheForDeployment=False
```

Save and restart LabVIEW.

Alternatively, you may resolve the issue by renaming your Statechart.

I receive the error "LabVIEW: Resource not found. An error occurred loading VI." This occurs the custom display element of a Statechart becomes corrupt. The custom data display is not used by this lab and can be restored to resolve the issue. From the downloaded project files, open

src\LabVIEW\statechart\Restore Corrupt Statechart\Restore Corrupt Statechart.vi

Follow the instructions on the VI to restore your Statechart.

6.11 Hill Climb in Statecharts (Simulation)

This lab builds on the concept of model-based design for cyber-physical systems and focuses on feedback-control. Your goal in this lab is to implement a Statechart that instructs the Kobuki to navigate to the top of an incline while avoiding cliffs and obstacles in a simulated environment.

This lab builds on Lab 6.9: Navigation in Statecharts (Simulation).

6.11.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

- §1.6.1 (Introduction to the myRIO Accelerometer).
- Introduction to Embedded Systems (Lee and Seshia, 2015)
  - Chapter 7: Sensors and Actuators §7.1 (Models of Sensors and Actuators)
  - Chapter 7: Sensors and Actuators §7.2.1 (Measuring Tilt and Acceleration)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

- Kobuki Documentation (YujinRobot, 2014b)

These documents go beyond the scope of the core concepts of these exercises, and are provided as additional resources.

- Using an Accelerometer for Inclination Sensing (Fisher, 2010)
- Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors (Ozyagcilar, 2011)
- Computational Principles of Mobile Robotics (Dudek and Jenkin, 2000)

6.11.2 Prelab Exercises

1. §1.6.3 (MyRIO Accelerometer Exercises) (all exercises). You may skip these if you have already completed them for another laboratory in this chapter.

6.11.3 Lab Exercises

For the exercises below, begin with the solution you developed for Lab 6.9: Navigation in Statecharts (Simulation), located in the downloaded exercise files:

src\LabVIEW\statechart

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW Statecharts Module 2014

1. **Hill Climb in Simulation:** Program the virtual robot. Using feedback from the accelerometer, your robot should differentiate between an incline and level ground, and when on an incline, orient and drive towards the top. You must satisfy all design requirements as specified in §6.2 (Hill Climb Design Requirements).

(a) Provide a screenshot of your climbing algorithm.
(b) Which states did you add from the previous lab?

2. **Share your Feedback:** what did you like about this lab, and what would you change?

6.12 Hill Climb in Statecharts (Deployment)

This lab builds on the concept of model-based design for cyber-physical systems and focuses on feedback-control. Your goal in this lab is to implement a Statechart on myRIO that instructs the Kobuki to navigate to the top of an incline while avoiding cliffs and obstacles.

This lab builds on Lab 6.10: Navigation in Statecharts (Deployment) and Lab 6.11: Hill Climb in Statecharts (Simulation).

6.12.1 Prelab Reading

There is no prelab reading for this laboratory.

6.12.2 Prelab Exercises

There are no prelab exercises for this laboratory.

6.12.3 Lab Exercises

For the exercises below, begin with the solution you developed for Lab 6.10: Navigation in Statecharts (Deployment); be sure to review the background from this lab, as it underscores important differences between the simulator and the real robot.

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

- Windows XP SP3, Windows Vista, Windows 7, or Windows 8
- National Instruments LabVIEW 2014
- National Instruments LabVIEW for myRIO Module 2014
- National Instruments LabVIEW Real-Time Module 2014
- National Instruments NI-RIO 13.1
- National Instruments LabVIEW Statecharts Module 2014
- National Instruments myRIO 1900 or 1950
- DC power supply
- USB cable
- iClebo Kobuki

1. **Hill Climb on the Kobuki:** Program the real robot. Using feedback from the accelerometer, your robot should differentiate between an incline and level ground, and when on an incline, orient and drive towards the top. You must satisfy all design requirements as specified in §6.2 (Hill Climb Design Requirements).

(a) Did your Statechart require modification to work on the physical target? If so, what modifications did you make?
(b) How did you account for movement of the robot when reading from the accelerometer? Describe any algorithms or filters used.

2. **Share your Feedback:** what did you like about this lab, and what would you change?

6.12.4 Troubleshooting

Refer to Lab 6.10: Navigation in Statecharts (Deployment) §6.10.4 (Troubleshooting) for troubleshooting issues with the robot configuration.

7 Projects

The exercises in this chapter guide you through basic skills in project management, a critical component of any successful capstone design project. Additionally, a series of open-ended projects are proposed as starting points for a significant design project.

7.1 Project Management

Contributing Author: Christopher X. Brooks

A capstone design project is a significant team undertaking, and can benefit from some simple project management. Well-managed projects are less stressful and ultimately more successful. Some useful project management tasks include:

- Specify the project by submitting a one-page charter.
- Once a project charter has been reviewed and approved, submit a project plan of action, a timeline for milestones, and a division of responsibilities for team members.
- Indicate core concepts addressed by the project, such as concurrency, modeling of physical dynamics, reliable real-time behavior, modal behavior governed by finite state machines coupled with formal analysis, real-time networks, simulation strategies, and design methodologies for embedded systems design.
- Pair with a mentor with relevant experience who is a professor, graduate student, researcher, or industry professional.
- Perform weekly progress checkpoints, which may alternate between in-class presentations and one-page milestone reports comparing progress to the original project charter.
- At the project halfway point, host live demonstrations during a department open-house, which facilitates functional milestones.
- Periodically submit peer evaluation forms to identify any issues with a particular team member.

7.1.1 Prelab Reading

Carefully read the following content and be comfortable with the concepts covered *before beginning these exercises.*

1. Project Management Instructions (Brooks, 2008)
2. Tortoise SVN (Kung et al., 2011)

Skim these documents to understand their content and layout. You should be able to quickly find relevant information within them. Sections pertaining to key topics may be explicitly referenced, but you may need to reference other sections to complete these exercises.

1. Project Management and Embedded Systems (Brooks, 2012b)
2. Wikipedia
   (a) Project Charter (Wikipedia, 2013c)
   (b) Milestone (Project Management) (Wikipedia, 2013b)
   (c) Work Breakdown Structure (Wikipedia, 2013e)
3. EECS Instructional Public Documentation - SVN Help (Brooks, 2012a)

You will need the following equipment and software to complete this lab, in addition to equipment used in prerequisite labs (if applicable):

1. Desktop computer
2. Word processor
3. Tortoise SVN Product Page (Tortoise SVN Team, 2013)

7.1.2 Prelab Exercises

1. What are two advantages of using a version control system such as SVN?

2. Alice and Bob both update their working copies at the beginning of the day. They start editing the same plain-text file. Alice commits her changes in the afternoon. What happens when Bob tries to commit his changes at the end of the day? What should Bob do?

3. What is the SVN checkout URL for your group (your group number is on the Google spreadsheet)?

4. How long should a project charter be?

5. Why is it important to have a *brief* charter?

6. Suppose you are writing the work breakdown to design an automobile, which has subsystems such as steering, horn, drivetrain, transmission, motor, etc. Give an example of a breakdown which would

(a) Violate the 100% rule.
(b) Violate the Mutually Exclusive rule

7. Make a short list of skills / experience / interests you think would be most relevant for your project. (i.e. Microcontroller coding, Circuit design, Project Management, etc.) Next to each, give an example project or class you took which demonstrates that skill.

8. If you know you will be using a mentored project from the website, communicate with them *before this lab*, asking for whatever details you need to know to complete the charter & schedule. i.e. you will need to know what components need to be designed, whether there is hardware tasks involved, if there are deadlines outside of this class, and other considerations. You should also schedule regular meetings with them for them to track your progress. List all such components, tasks, and deadlines.

7.1.3 Lab Exercises

The exercises here guide you in the creation of a project charter. Your charter serves as the lab report for these exercises. Do not enumerate the questions below in your charter, but instead make sure that each question is answered somewhere in your project charter.

TortoiseSVN is a *frontend* for the program SVN, a version-control system. TortoiseSVN provides handy right-click menus for updating, committing and reviewing your files.

1. **Check out a Repository:** To checkout your repository, create a new directory where you want the files (preferably one on a network drive). Right-click on the new directory and select "SVN Checkout...". In "URL Repository" enter the URL of your repository. When a module is checked out, notice a check mark appears next to the folder indicating your local copy is up-to-date.

2. **Add Files to your Repository:** Copy or move the files from your project charter and schedule into this newly created folder. Select both and right-click on them. Select "Tortoise SVN → Add...". Notice there are now plus-signs next to the files. This means that this file is to be added to the repository on the next commit.

Go up a directory. The folder containing your local copy should now have a red exclamation point indicating your local copy is out-of-date. Right-click on the folder and select "SVN Commit...". Enter a message such as "Added project charter & schedule". *Although not strictly enforced, it is good practice to always provide a commit message for other users and for remembering what you did in the future* . The folder should now turn green again.

Go back into the directory. Your project charter and schedule should now have green checkmarks indicating they are up-to-date. Try changing one of them– you should get a red exclamation point indicating local changes are out-of-date. You can commit these changes in the same way as before, but suppose you made a mistake and you want to undo your changes. Right-click on the file and select "Tortoise SVN → Revert...". If you press "OK", the latest version in the repository will overwrite your copy. You can also revert to several versions behind if you want.

Note that because your files are probably in MS Word or some other binary format, then you cannot *diff* them easily. The *diff* feature works on plain-text files, such as C source files, and highlights the changes, line-by-line, that you changed between two revisions. This is essential for finding out which lines of code broke your program.

For plain-text files, you can also handle *merges* which happen if two people tried to change the same lines– you will have to pick which one is used. For binary files, to "merge" you must simply pick one entire file or the other.

You will probably want to have every member set up their respective checkouts from your repository. This will allow you to work on the code somewhat independently, but with a central master copy. You should get in the habit of committing your changes at least once a day (like the CTRL-S of code development). Your TAs and mentors will also be using your repository to be familiar with your code.

3. **(optional) Install SVN on Another Computer:** This section is for your information in case you want to set this up on your own computer.

Windows: Installation is fairly easy– simply download and install TortoiseSVN from Tortoise SVN Product Page↗ (Tortoise SVN Team, 2013), making sure to get 32-bit or 64-bit based on what your OS is. On Windows 7, there may be problems with the icons showing properly. Therefore, you will need to use the "check for modifications" item in TortiseSVN's right-click menu.

Linux: The "svn" command-line program should be available in most package repositories. For example, on Ubuntu systems, you can use "sudo apt-get install subversion". Refer to the man-page or online documentation for information on how to use the command-line program. There are also frontends available, such as kdesvn ("sudo apt-get install kdesvn").

Mac: Mac OS X 10.7 may already have Subversion installed. To test this, start up a Terminal window and run "svn help". If svn is not found, then install XCode from http://developer.apple.com/xcode. Note that XCode is roughly a 1 gigabyte download. Apple is charging $5 for XCode 4, but XCode 3 will also work.

Unfortunately for the Mac, the free GUI Subversion options are fairly limited. SCPlugin seems to be the best option out there, coming from the makers of subversion. It can be found at http://scplugin.tigris.org/.

4. **Team Member and Project Selection:** During this lab (your choice – before or after your charter is written) discuss with your lab TA that all skills requirements for your project are satisfied by your group members. If you have a large (4 or 5) group, be prepared to justify why the extra manpower is needed.

(a) Who are your project members?
(b) Give a one or two sentence description of your project.

5. **Plan your Project:** Follow Project Management Instructions↗ (Brooks, 2008) to create a project charter.

Ensure that you are reasonable in your objectives and deliverables, especially considering the experience of your group. i.e. Do not make a circuit board a major deliverable if no group member has any circuit design experience. On the same token, the scope of the project should match the size of the group and the goal be to produce an interesting and impressive product.

Your charter should include:

(a) Overview
(b) Approach
(c) Objectives
(d) Major Deliverables
(e) Constraints
(f) Risk and Feasibility

Your charter should be one (1) page. The point is succinctly summarize your project for a busy manager.

Each group should submit one charter.

6. **Set Milestones and a Schedule:** Working backwards from the completion due date, determine when tasks need to be done to complete the project on time. *Ensure you are allowing adequate time for debugging – integrating disjoint components from several people can take a week or more*. Be sure to consider events such as Spring Break, Finals Week, etc.. Milestones should indicate which team member is responsible and be no more than one week apart.

Be sure to include non-technical tasks such as practicing the presentation or writing the report. Also include a plan to meet with your mentor, preferably once a week (you may want to organize your milestones to land on these dates). Be sure to include vacations like Spring Break.

Do not hesitate to be painfully specific. Through the duration of this project, you should submit milestone reports to track your progress. Certainly, the milestones could change during the course of the project as you learn and debug things.

7. **Set a Work Breakdown Structure:** Generate a Work Breakdown Structure (WBS) for your project, being careful to satisfy the 100% and mutual exclusion rules. The

diagram should show dependencies between milestones and which tasks can be performed in parallel. For simpler projects, this could be very linear, but try to divide the tasks so as much as possible can be done in parallel so work can be split between people. Ensure that the WBS answers when a milestone will be finished and who is responsible for it finishing.

8. **Check your Charter into SVN.**

9. **Individual Writeup:** In addition to the project charter, each team member should write an individual report which answers the following questions:

(a) What is the critical path on your WBS? How long will it take to complete?
(b) What are the amounts time you have allotted to:
    i. Design
    ii. Development (i.e. new stuff being written)
    iii. Debugging
    iv. Other tasks (such as writing presentations or reports)
(c) How have you designed your schedule adjust for unforeseen delays or early completions?
(d) When have you regularly scheduled to meet with your mentor?

10. **Share your Feedback:** what did you like about this lab, and what would you change?

A

Lab Setup

This appendix describes how to configure hardware and software to support all of the laboratory exercises in this text.

A.1 Install LabVIEW

From the National Instruments LabVIEW National Instruments LabVIEW 2014 Platform DVD or Academic Site License DVD, install the following products

- LabVIEW English
- VI Package Manager
- Device Drivers
- Real-Time → Real-Time Module
- FPGA → FPGA Module
- FPGA → Compilation Tools for FPGA Devices other than Virtex-II
- Simulation and Control → PID and Fuzzy Logic Toolkit
- Signal Processing, Analysis, and Math → MathScript RT Module
- Software Development and Deployment → Statechart Module
- Robotics → Robotics Module

All other packages may be deselected. When prompted for drivers to install, the default settings will suffice.

From the LabVIEW National Instruments LabVIEW 2014 for myRIO DVD, use the default install settings to install the LabVIEW National Instruments LabVIEW 2014 for myRIO Module and supporting drivers. The LabVIEW National Instruments LabVIEW 2014 module for myRIO may also be downloaded from

http://joule.ni.com/nidu/cds/view/p/id/4223/lang/en

Once all installers are complete, run NI Update Service to update all NI software.

Download two configuration files from the laboratory website,

http://LeeSeshia.org/lab/releases/1.70/labviewConfigurationFiles.zip

and extract its contents to your LabVIEW program directory, commonly

C:\Program Files (x86)\National Instruments\LabVIEW 2014

Run through Lab 3.4: Program the myRIO Processor from LabVIEW to ensure the software is configured correctly and to add any firewall exceptions needed.

A.1.1 Troubleshooting

When running NI Update Service, I receive the error message "This service has been disabled by your administrator": This indicates an environment variable is preventing the FlexLM software from querying the academic site license – this is a variable set by either Xilinx SDK or other software using FlexLM licensing services. From the System settings, find Environment Variables and record the value of the key "LM_LICENSE_FILE". Clear the value (temporarily) and run NI Update again. When the update is complete, restore the value of the LM_LICENSE_FILE key.

A.2 Install Visual Studio Express

Microsoft Visual Studio is used to program CyberSim in C. Microsoft Visual Studio Express is free for commercial use, and may be downloaded from:

- Microsoft Visual Studio Express 2013 for Desktop (Windows 7 and 8):

  http://www.visualstudio.com/en-us/downloads

- Microsoft Visual C++ 2010 Express (Windows XP and Windows Vista):

  http://www.visualstudio.com/en-us/downloads/download-visual-studio-vs

A.3 Install C & C++ Development Tools

C & C++ Development Tools for NI Linux Real-Time, Eclipse Edition is used to program the myRIO processor in C. It is distributed for free and may be downloaded from

http://joule.ni.com/nidu/cds/view/p/id/4286/lang/en

Eclipse requires the Java Runtime Environment which may be downloaded from

http://www.java.com/getjava

Next, add the path of the compiler toolchain to the system path. From the Windows Control Panel, open System and search for "Advanced System Settings→Environment Variables". Append the toolchain location to the PATH variable, ensuring that you separate paths with a semicolon. The compile toolchain is installed by default to:

C:\Program Files (x86)\National Instruments\Eclipse\toolchain\gcc-4.4-arm\i386\bin

Remove " (x86)" if you are on a 32-bit operating system.

Run through Lab 3.3: Program the myRIO Processor from Eclipse to ensure the software is configured correctly and to add any firewall exceptions needed.

A.4 Install Xilinx SDK

Download Xilinx SDK 14.4 from the Xilinx website.

http://xilinx.com/support/download/

You will need to search to find Xilinx Software Development Kit 14.4 (2012.4). You may need to create a Xilinx online profile to download the software.

Extract the downloaded archive and run **xsetup.exe** to begin installation. Once the installer completes, run a supplemental installer that is copied by default to

C:\Xilinx\14.4\SDK\common\bin\nt64\vcredist_x64.exe

If you are not installing onto a 64 bit operating system, the path for the installer is

C:\Xilinx\14.4\SDK\common\bin\nt\vcredist_x86.exe

Plug in the Xilinx Platform USB Cable II and ensure the driver installs correctly, then restart the machine.

If you are on a 64-bit operating system, ensure that the 64-bit SDK is in the desktop and Start Menu (and not the 32-bit, for which JTAG drivers will not be installed).

To allow Xilinx SDK to see myRIO over JTAG, download

http://LeeSeshia.org/lab/releases/1.70/idecode.lst

and replace the existing file

C:\Xilinx\14.4\SDK\SDK\data\cse\idcode.lst

Run through Lab 3.2: Program MicroBlaze from Xilinx SDK to ensure the software is configured correctly and to add any firewall exceptions needed.

A.5 myRIO JTAG Wiring

Xilinx SDK uses JTAG to download and debug applications for MicroBlaze running on the myRIO FPGA. This is only supported on myRIO 1950, and you may need to soldier pin headers onto the JTAG header (J5) (Fig. A.1 and Table A.1).

See myRIO JTAG Instructions[ω] (National Instruments, 2013j) for more information on JTAG on myRIO.

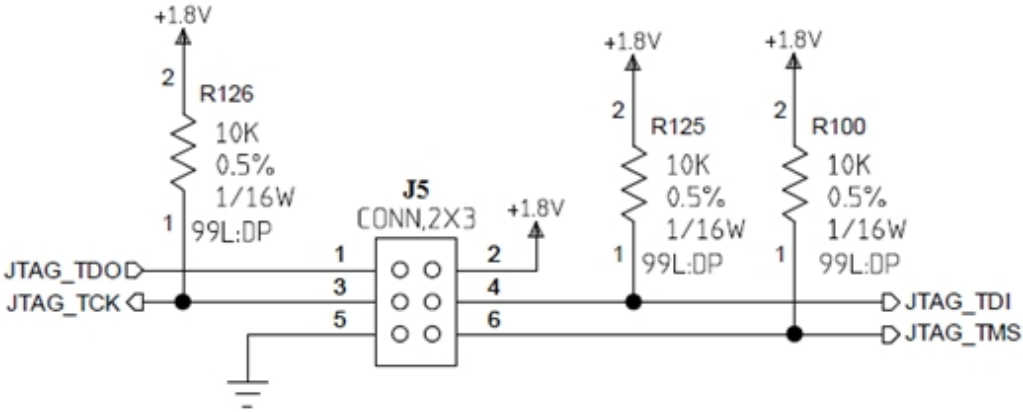


Figure A.1: JTAG pinout on myRIO 1950.

JTAG Signal	JTAG Color	myRIO Pin (J5)
TDO	purple	1 (JTAG_TDO)
VREF	red	2 (+1.8V)
TCK	yellow	3 (JTAG_TCK)
TDI	white	4 (JTAG_TDI)
GND	black	5 (GND)
TMS	green	6 (JTAG_TMS)

Table A.1: myRIO 1950 to JTAG wiring diagram.

A.6 myRIO iRobot Create Wiring

A complete wiring diagram is shown in Fig. A.2, and mounting of myRIO in the iRobot Create is shown in Fig. A.3.

Run through the template project in Lab 5.10: Navigation in Statecharts (Deployment) to verify that communication is established between myRIO and the iRobot Create. If sensor data appears on the Front Panel, the configuration is correct.

Visit the myRIO Community (National Instruments, 2013h) for additional information on myRIO.

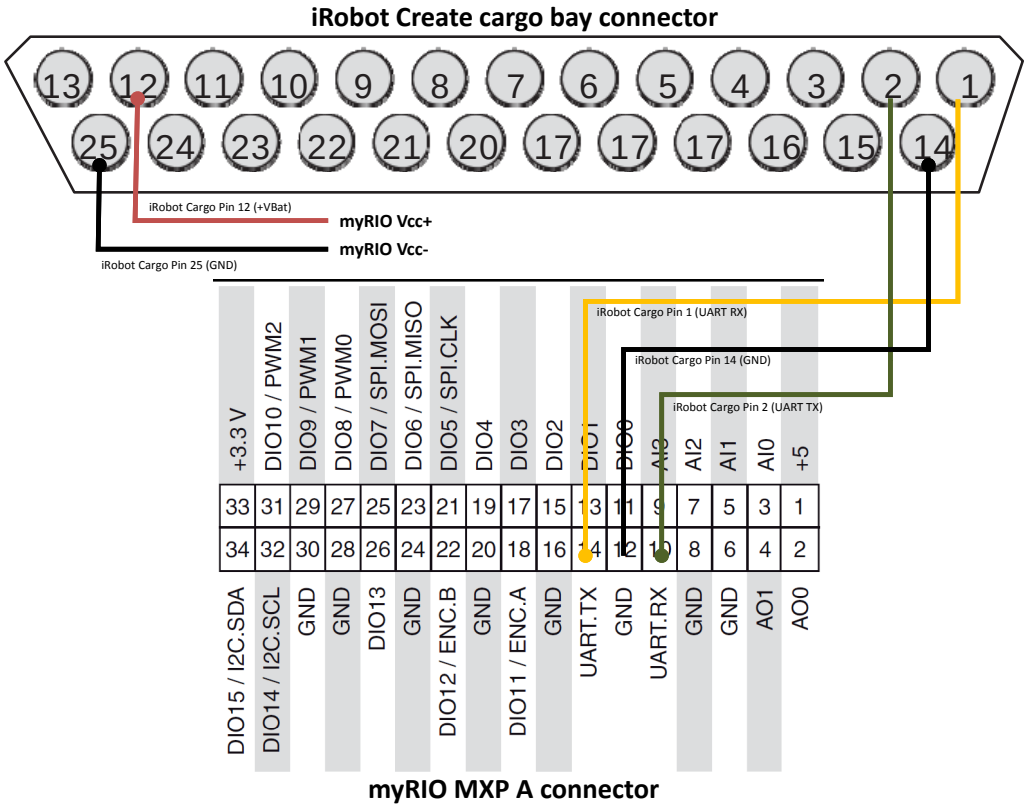


Figure A.2: iRobot Create and myRIO wiring diagram. Pinout diagrams replicated from the iRobot Create Open Interface (OI) Specification (iRobot, 2006a).

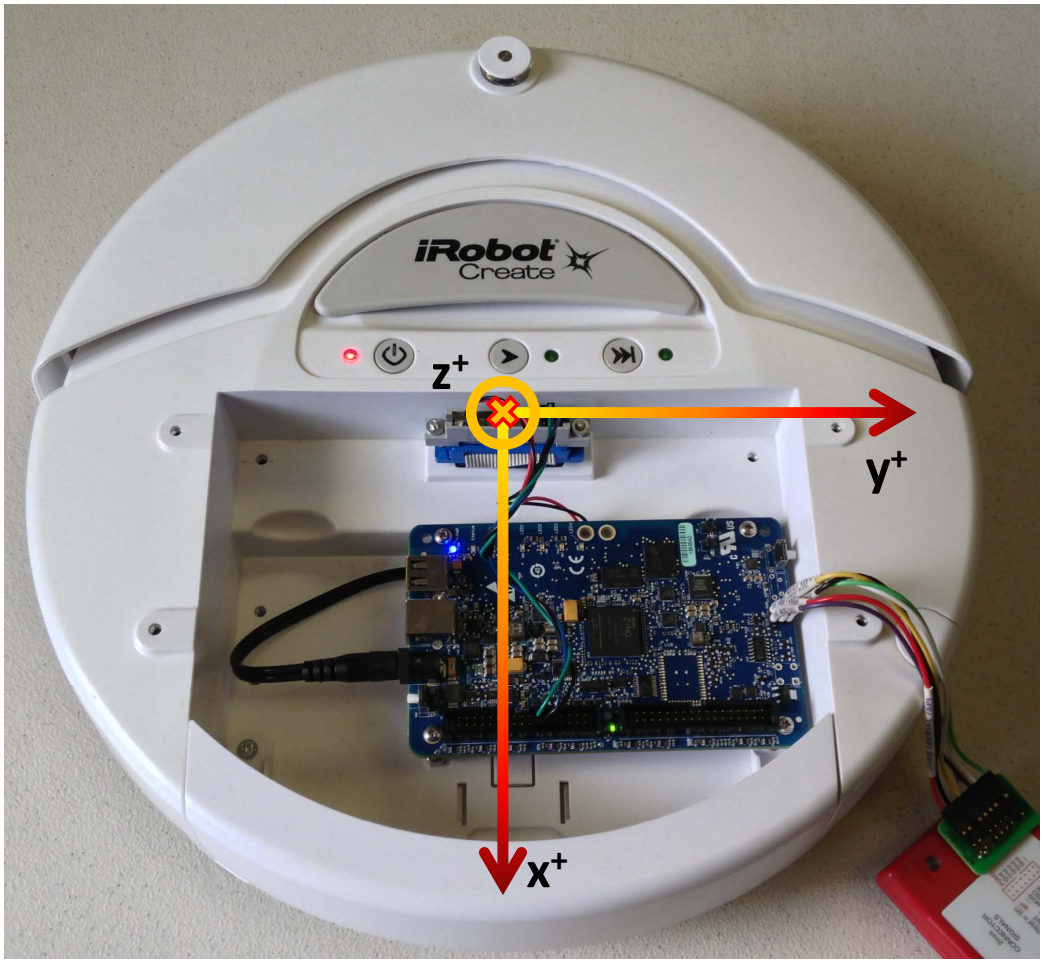


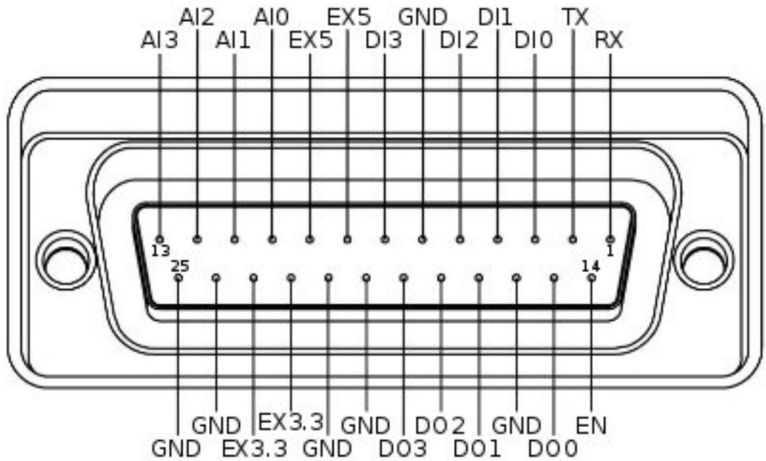
Figure A.3: Mounting of myRIO in the iRobot Create, and the resulting coordinate system for the onboard accelerometer.

A.7 Kobuki Wiring

Run through the template project in Lab 6.10: Navigation in Statecharts (Deployment) to verify that communication is established between myRIO and the Kobuki. If sensor data appears on the Front Panel, the configuration is correct.

Visit the myRIO Community (National Instruments, 2013h) for additional information on myRIO.

Serial Port



- RX / TX: Serial data connection (RS232; used voltage level is 3.3V!)
- EX3.3 / EX5: 3.3V/1A and 5V/1A power supply
- DI0 - 3: 4 x Digital input (high: 3.3 - 5V, low: 0V)
- DO0 - 3: 4 x Digital output (open-drain, pull-up resistor required)
- AI0-3: 4 x Analog input (12bit ADC: 0 - 4095, 0 - 3.3V)
- GND: Ground
- EN: Used for detecting an external board (connect to external ground)

Figure A.4: Kobuki wiring diagram. Pinout diagrams replicated from the Kobuki User Guide Kobuki User Guide (YujinRobot, 2014d).

Bibliography

Bluetooth Special Interest Group, 2003: Bluetooth HID Profile. v1.0. Available: https://bluetooth.org/Technical/Specifications/adopted.htm.

Brooks, C., 2008: Project Management Instructions. Available: http://technology.berkeley.edu/cio/tpo/project/pmresources/tools/Project_Charter_Instructions.doc.

—, 2012a: EECS Instructional Public Documentation - SVN Help. Available: https://inst.eecs.berkeley.edu/cgi-bin/pub.cgi?file=svn.help.

—, 2012b: Project Management and Embedded Systems. lecture slides for EECS 149, University of California, Berkeley. Available: http://chess.eecs.berkeley.edu/eecs149/projects/Brooks_eecs149_sp12_ProjectManagementOverview.ppt.

Defense Advanced Research Projects Agency, 2012: Single chip timing and inertial measurement unit. Available: https://commons.wikimedia.org/wiki/File:DARPA_TIMU_Michigan_Penny.jpg.

Dudek, G. and M. Jenkin, 2000: *Computational Principles of Mobile Robotics*. Cambridge University Press.

Eclipse Foundation, 2013a: Eclipse. Available: http://eclipse.org.

—, 2013b: Eclipse Juno Documentation. Available: http://help.eclipse.org/juno/.

Fisher, C. J., 2010: Using an Accelerometer for Inclination Sensing. Analog Devices application note AN-1057 v0. Available: http://analog.com/static/imported-files/application_notes/AN-1057.pdf.

Freescale Semiconductor, 2013: Xtrinsic MMA8452Q 3-Axis 12-bit/8-bit Digital Accelerometer. document #MMA8452Q. Available: http://cache.freescale.com/files/sensors/doc/data_sheet/MMA8452Q.pdf.

Hickok, T., 2009: MicroBlaze Interrupts and the EDK. Available: http://chess.eecs.berkeley.edu/eecs149/documentation/th_ublaze_interrupts.pdf.

iRobot, 2006a: Create Open Interface Specification. v2. Available: http://irobot.com/filelibrary/create/Create%20Open%20Interface_v2.pdf.

—, 2006b: Create Owner's Guide. v430.06. Available: http://irobot.com/filelibrary/create/Create%20Manual_Final.pdf.

—, 2013: iRobot Create Product Page. Available: http://irobot.com/create.

Jensen, J. C., E. A. Lee, and S. A. Seshia, 2011: An introductory capstone design course on embedded systems. In *IEEE International Symposium on Circuits & Systems (ISCAS), special session: Design, Project, & Learning Technology Innovations in Circuits, Signals, & Systems Education*, Rio de Janeiro, Brazil, pp. 1199–1202. doi:10.1109/ISCAS.2011.5937784.

—, 2012: Teaching embedded systems the berkeley way.

Kodosky, J., J. MacCrisken, and G. Rymar, 1991: Visual programming using structured data flow. In *IEEE Workshop on Visual Languages*, IEEE Computer Society Press, Kobe, Japan, pp. 34–39.

Kung, S., L. Onken, and S. Large, 2011: Tortoise SVN. Available: http://tortoisesvn.net/docs/release/TortoiseSVN_en/.

Lee, E. A., 2006: The problem with threads. Tech. Rep. UCB/EECS-2006-1, EECS Department, University of California, Berkeley. Available: http://eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.html.

Lee, E. A. and S. A. Seshia, 2010: An introductory textbook on cyber-physical systems. In *Proc. Workshop on Embedded Systems Education (WESE)*, pp. 1–6. doi:10.1145/1930277.1930278.

—, 2015: Introduction to Embedded Systems. v2.0. Available: http://LeeSeshia.org.

Lee, E. A., S. A. Seshia, and J. C. Jensen, 2013: EECS 149 - Introduction to Embedded Systems. course website. Available: http://chess.eecs.berkeley.edu/eecs149.

Microsoft, 2014a: Visual Studio. Available: http://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx.

—, 2014b: Visual Studio Get Started. Available: http://www.visualstudio.com/get-started/overview-of-get-started-tasks-vs.

National Instruments, 2010: LabVIEW Quick Reference Card. P/N #373353D-01. Available: http://ni.com/pdf/manuals/373353d.pdf.

—, 2012a: Developing Algorithms Using LabVIEW MathScript RT Module: Part 1 - The LabVIEW MathScript Node. Available: http://www.ni.com/white-paper/3256/en.

—, 2012b: Developing Applications with the NI LabVIEW Statechart Module. Available: http://zone.ni.com/devzone/cda/tut/p/id/6194.

—, 2012c: Getting Started with MathScript RT. P/N #373123C-01. Available: http://zone.ni.com/reference/en-XX/help/373123C-01/lvtextmathmain/ms_getting_started/.

—, 2012d: LabVIEW Statechart Module Help. P/N #372103D-01. Available: http://zone.ni.com/reference/en-XX/help/372103D-01/.

—, 2012e: MathScript RT Module Functions. P/N #373123C-01. Available: http://zone.ni.com/reference/en-XX/help/373123C-01/lvtextmath/msfunc_classes/.

—, 2013a: C Support for NI myRIO User Guide. Available: http://joule.ni.com/nidu/cds/view/p/id/4259/lang/en.

—, 2013b: Getting Started with LabVIEW. P/N #373427J-01. Available: http://ni.com/pdf/manuals/373427j.pdf.

—, 2013c: LabVIEW 2013 Help. P/N #371361K-01. Available: http://zone.ni.com/reference/en-XX/help/371361K-01/.

—, 2013d: LabVIEW 2013 Real-Time Module Help. P/N #370622L-01. Available: http://zone.ni.com/reference/en-XX/help/370622L-01/.

—, 2013e: LabVIEW Product Page. Available: http://ni.com/labview.

—, 2013f: LabVIEW Robotics Simulator Wizard. P/N #372983D-01. Available: http://zone.ni.com/reference/en-XX/help/372983D-01/lvrobodialog/robo_sim_wizard/.

—, 2013g: myRIO 1950 User Guide and Specifications.

—, 2013h: myRIO Community. Available: http://decibel.ni.com/content/community/academic/products_and_projects/myrio.

—, 2013i: myRIO Getting Started. Available: http://ni.com/myrio/setup/getting-started/.

—, 2013j: myRIO JTAG Instructions.

—, 2013k: myRIO Product Page. Available: http://ni.com/myrio.

—, 2013l: myRIO Shipping Personality Reference. Available: http://www.ni.com/white-paper/14655/en.

—, 2014: Learn LabVIEW. Available: http://www.ni.com/academic/students/learn-labview/.

Nintendo, 2013: Nintendo Controllers. Available: http://nintendo.com/wii/console/controllers.

Orlando, G., 2010: Wii remote image. Creative Commons BY-SA 3.0 via Wikipedia. Available: http://en.wikipedia.org/wiki/File:Wiimote.png.

Ozyagcilar, T., 2011: Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors. Analog Devices application note AN-4248 v2. Available: http://freescale.com/files/sensors/doc/app_note/AN4248.pdf.

Smith, R., 2012: Open dynamics engine. Available: http://ode.org.

SparkFun Electronics, 2013: Triple Axis Accelerometer Breakout - ADXL335. CC BY-NC-SA 3.0. Available: http://sparkfun.com/products/9269.

Stoops, M., 2008: Wii remote uncovered. GNU Free Documentation License 1.2. Available: http://wiibrew.org/wiki/File:Wii_Remote_uncovered.jpg.

Suvasa, E., 2011: Gnu logo. Free Art License. Available: https://en.wikipedia.org/wiki/File:Heckert_GNU_white.svg.

Tortoise SVN Team, 2013: Tortoise SVN Product Page. Available: http://tortoisesvn.net/.

Washington, C., 2009: Designing Applications with the NI LabVIEW Statechart Module. National Instruments webcast. Available: http://www.ni.com/webcast/225/en/.

WiiBrew, 2012: WiiMote. Available: http://wiibrew.org/wiki/Wiimote.

Wikipedia, 2013a: Joint Test Action Group. Available: http://en.wikipedia.org/wiki/Jtag.

—, 2013b: Milestone (Project Management). Available: http://en.wikipedia.org/wiki/Milestone_(project_management).

—, 2013c: Project Charter. Available: http://en.wikipedia.org/wiki/Project_charter.

—, 2013d: Wii Remote. Available: http://en.wikipedia.org/wiki/Wii_Remote.

—, 2013e: Work Breakdown Structure. Available: http://en.wikipedia.org/wiki/Work_breakdown_structure.

Xilinx, 2012a: Embedded System Tools Reference Manual. UG111 v14.3. Available: http://xilinx.com/support/documentation/sw_manuals/xilinx14_4/est_rm.pdf.

—, 2012b: LogiCORE IP AXI Timer. DS764 v1.03.a. Available: http://xilinx.com/support/documentation/ip_documentation/axi_timer/v1_03_a/axi_timer_ds764.pdf.

—, 2012c: MicroBlaze Processor Reference Guide. UG081 v14.4. Available: http://xilinx.com/support/documentation/sw_manuals/xilinx14_4/mb_ref_guide.pdf.

—, 2013a: EDK support. Available: http://xilinx.com/support/index.html/content/xilinx/en/supportNav/design_tools/embedded_development_kit__edk.html.

—, 2013b: LogiCORE IP AXI INTC. DS747 v1.04.a. Available: http://xilinx.com/support/documentation/ip_documentation/axi_intc/v1_04_a/ds747_axi_intc.pdf.

—, 2013c: MicroBlaze Soft Processor Core. Available: http://xilinx.com/tools/microblaze.htm.

YujinRobot, 2014a: Kobuki C++ Driver Specification. Available: https://yujinrobot.github.io/kobuki/doxygen/enMainPage.html.

—, 2014b: Kobuki Documentation. Available: http://kobuki.yujinrobot.com/home-en/documentation/.

—, 2014c: Kobuki Quick Guide. Available: http://kobuki.yujinrobot.com/files/3413/5752/0704/kobuki_quickguide_A3.pdf.

—, 2014d: Kobuki User Guide. Available: http://kobuki.yujinrobot.com/home-en/documentation/online-user-guide/.

What is the feeling when you're driving away from people,
and they recede on the plain
till you see their specks dispersing?
It's the too huge world vaulting us,
and it's goodbye.
But we lean forward
to the next crazy venture
beneath the skies.

–Jack Kerouac, *On the Road*