Cross Development for Real-Time Systems and Embedded Automation

Diploma Thesis

Zhuang Yifan

Betreuer
O.Univ.-Prof. Dipl.-Ing. Dr.techn. Paul O'Leary B.A., B.A.I., M.E.E.
Ass.Prof. Dipl.-Ing. Dr.mont. Gerhard Rath

Montanuniversität Leoben
Institut für Automation

December 2012

Abstract

Nowadays industrial automation control tasks are increasingly implemented in embedded real-time computer systems. Modern microprocessors developed for mobile computing and innovative sensors allow more sophisticated algorithms on a higher level of mathematics. For this purpose newer software techniques are required, which are not provided by platforms with traditional automation languages. Furthermore the developers have to speed up the design process even with increasing complexity of the control tasks. This thesis aims to explore the capabilities of model-based software design on such systems. After an introduction to real-time systems and model-based design process, a cross-development platform was implemented on two popular target hardware systems. For this purpose Matlab/Simulink was chosen. The experiences made during the development of application examples on both systems are documented. Finally, a concrete signal processing algorithm for a recently developed acoustic sensor for injection molding machines was implemented and tested under real-time conditions.

Keywords

Embedded systems; Real-time systems; Model-based Design; Signal-processing

Kurzfassung

Steuerungsaufgaben in Maschinen und Anlagen werden heute zunehmend auf eingebetteten Systemen implementiert. Moderne Mikroprozessoren, die für Mobile Computing entwickelt wurden, sowie neuentwickelte Sensoren erlauben die Anwendung von komplexeren Algorithmen auf einem höheren mathematischen Niveau. Zu diesem Zweck sind Software-Techniken notwendig, die von traditionellen Programmierumgebungen für industrielle Systeme nicht unterstützt werden. Zusätzlich ist eine Verkürzung des Entwicklungsprozesses trotz steigender Komplexität der Aufgaben gefordert. Diese Arbeit untersucht die Möglichkeiten, die sich durch Model-Based Design bei der Programmierung von eingebetteten Systemen ergeben. Nach einer Einführung in Echtzeit-Systeme und Model-Based Design in der Software-Entwicklung wird die Implementation einer Cross-Plattform-Entwicklungsumgebung anhand von zwei bekannten Zielsystemen gezeigt. Zu diesen Zweck wurde Matlab/Simulink als Plattform gewählt. Die Erfahrungen bei der Entwicklung von Beispiel-Applikationen wurden dokumentiert. Algorithmen zur digitalen Signalverarbeitung für einen neuentwickelten akustischen Sensor aus der Spritzgusstechnik für Kunststoffe wurden realisiert und unter Echtzeitbedingungen erfolgreich getestet.

Schlagwörter

Eingebettete Systeme; Echtzeit-Systeme; Model-based Design; Signalverarbeitung

Affidavit

Acknowledgements

- First and foremost, I would like to show my deepest gratitude to my co-supervisor, Prof. Gerhard Rath, a respectable, responsible and resourceful scholar, who has provided me with precious guidance in every stage of the writing this thesis. Without his enlightening instruction, impressive kindness and patience, this thesis could not have reached its present form.
- I appreciate my supervisor, Prof. Paul O'Leary who gives me the chance complete my thesis in automation institute, gives me his valuable advice and provides the support and equipment I have needed .
- I shall extend my thanks to Ms. Doris Widek and Mr. Gerold Probst for all their kindness and help.
- In my daily work I have been blessed with a friendly and cheerful group of colleagues.
- I thank my family for supporting me throughout all my life with their love.
- Last but not least, I want to thank all my friends in Leoben, especially my best friend forever Xu Tian for their encouragement and support.

Table of Contents

Chapter 1

Theoretical Background

1.1 Introduction

Cross development is used to generate executable code on one system (host) for executing on the other platform (target). It takes less time by cross development to bring a new product into a functional prototype. This kind development also improves the maintainability of a product and enables development of complex subsystems, which include sensors, actuators, communication interfaces and provide significant computation power for control tasks.

A number of developments have been emerged these years, which use stringent languages to define the system specifications. In this way the code could be generated and a formal testing of correctness could be performed. But these developments are not yet accepted by industry, since it doesn't reduce the difficulties by writing specifications instead of writing computer code. This approach can be described as 'correct by design'.

The tools such as Matlab$^{®}$/Simulink$^{®}$ or some other suites can be applied for simulation of large scale systems. In these systems correct function is tested based on interaction of electromechanical, physical or components of control systems and their communications. The simulation can be used as specification document, which is better than the classical paper version or code description, since the simulation tools can abstract the specification in a higher level than a computing language. From a single model different code is generated for different platforms, so it is obviously possible to use the model for defining specifications.

From a single model code can be generated for different platforms using the specific cross-compilers. Consequently, the simulation model represents the system specifications regardless of the target hardware. If the simulation behaviour meets the requirements as a specification, this is a method for verification or validation. Mistakes are found before the deployment of the system. We call this procedure ”correct by simulation”.

This change of paradigm in the development process is still not very evident at present. So the aims of this thesis are:

- To install and commission a complete environment for cross development.
- To investigate the intuitive workflow of a development process.
- To test single simulation model on different hardware targets.
- To use some applications to show the capability of the new development approach and furthermore with these examples the future investigations should know where they are going to be focused.

1.2 Embedded System

Sophisticated systems such as rockets or satellites contain many embedded systems (ES). Many household items such as cell phones, toys or MP3 players have embedded systems. Even an electric shaver, which has a small micro-controller to provide variable speed. Automobiles can contain more than fifty embedded micro-controllers. Most of all electronic chips produced today are for ES, they are all around us. We probably have more than a dozen in our homes right now. On the other hand, embedded systems are also significant for the industrial production.

1.2.1 Definition and Characteristics

Definition ***An embedded system is a computer system designed for specific control functions within a larger system, often with real-time computing constraints [25].***

The following characteristics are important for ES:

- Dependability,

  Reliability

  Maintainability

  Availability

  Safety and security

- Must be efficient with respect to,

  Energy

  Code-size

  Run-time

  Weight

  Cost

- Developed for a dedicated application and

- Many ES must meet real-time constraints (details will be talked later in section 1.2).

1.2.2 Difference between Embedded System and PC

An embedded system is an applied computer system, which is distinguished from other types of computer systems such as personal computers (PC) or supercomputers from following points:

1. ES are designed for some specific task rather than a multi-functional PC. Some ES also have real-time performance constraints that must be met, for reasons such as safety. Others may have low performance requirements, allowing the system hardware to be simplified to reduce costs.

2. Many ES are not standalone devices. They consist of small parts within a larger device, which serves a general purpose. For example an ES in a car provides a specific function as a subsystem and not a function as a car itself.

3. The program instructions written for ES are stored usually in Flash memory chips. It may have its own operating system and does not rely on having a hard disk.

4. Typically, the end user does not develop new software for the embedded device.

5. They are usually delivered with a fixed program, which is virtually never changed.

6. The embedded systems have always peripherals for inputs and outputs.

1.2.3 Embedded Operating Systems

The operating system manages computer hardware resources and provides common services for computer programs interfaces for the user. It is that piece of software that turns the collection of hardware blocks into a powerful computing platform [23]. They broadly have the tasks:

1. Processor and peripherals management
2. Memory and storage management
3. Providing common application interface
4. Providing common user interface

Most modern ES have complex software that needs to provide multitasking, a large amount of inputs and outputs, memory management, graphics displays, networking, etc. An embedded OS provides all these features to help support application developers. ES usually uses existing OS for saving development time and costs.

Embedded OS are often associated to real-time OS, which will be described later in section 1.2.

1.2.4 Design Tools for Embedded Systems

The software design happens in the development phase, which will be described now in more detail. Since now most ES have OS, the major factors to focus on a new design are availability of a proper OS, device drivers, applications and software development tools.

Tool Selection

Before developing an ES, the designer must know, which demands must be met, and choose the optimal tool. The following parameters can be used to judge the quality of a software for ES:

- Dynamic efficiency, which means in memory management, task switching, pre-fetch pipelining, etc.
- Static efficiency, which means in consumption of RAM size and ROM size.

- Power consumption depending on efficient switching and enabling of peripheral hardware. This is important for battery driven devices and sometimes relevant for EMI properties (electromagnetic interference).
- Easy to understand
- Easy to change

Software for Embedded System

A survey shows that today one third of the designers still use assembly language somewhere in their design [9]. The reason may probably be, that it is processor specific and efficient. But since it is hard to write and complex for a large program, the engineer would rather choose a high level language like C family.

Because the large variety of tasks of embedded systems, a general-purpose language would not be enough to solve them all. So always more than one languages are involved, each best suited to a particular problem domain.

In this thesis a model based development tool is presented in details. The advantage of this design method will be discussed in section 1.3. Such kind of tools lets you create and simulate graphical data flow and State chart diagrams of components like digital filters, communication protocol decoders and multi-rate tasks.

1.3 Real-Time Simulation and Control

1.3.1 Basic Real-Time Concepts

The notion of a system is very important to engineering and it can be defined like this: *A system is a mapping of a set of inputs into a set of outputs* [15]. Most real-world entities can be modeled as systems.

A system must respond to external inputs and produce new outputs in a limited amount of time as seen in Fig. 1.2. This fact can be described as follows: *The time between the presentation of a set of inputs to a system (stimulus) and the realization of the required behavior (response), including the availability of all associated outputs, is called the response time of the system* [15].

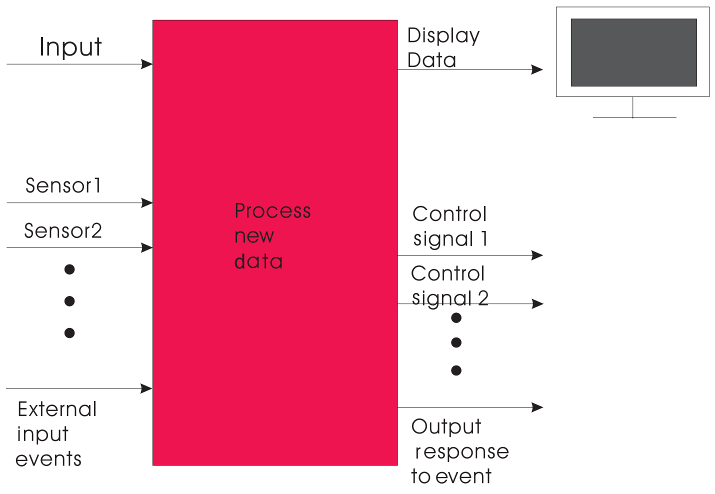


Figure 1.1: A typical real-time control system including inputs from sensors and imaging devices and producing control signals and display information

Real-Time Definitions

Definition *A real-time system is a system that must satisfy explicit (bounded) response-time constraints or risk severe consequences, including failure,* and a failed system means *this system cannot satisfy one or more of the requirements stipulated in the formal system specification [15].*

Classification An illustrative example of a real-time (RT for short) system is an automatic door. When the door's motion sensor detects any person standing at a distance in front of it, the system needs to respond by opening the door within 100 ms or the system fails. If someone enters the room at high speed, with any delay longer than 100 ms this person will hit the door.

RT systems are usually classified into [15][19]:

- *A soft real-time system is one in which performance is degraded but not destroyed by failure to meet response-time constraints.*
- *A hard real-time system is one in which failure to meet a single dead line may lead to complete and catastrophic system failure.*

A typical example for soft RT is an automated teller machine. The machine could occasionally miss the deadline, which does not lead to catastrophic failure, only degraded performance.

For a hard RT example, consider a airbag controller in the car. Missing the deadline to deploy the airbag within a specified time after the sensor detects a collision will lead to injury of the passengers [13].

Real-Time System Design Issues

Many skills qualify the engineers for the design and analysis of RT systems. The design and implementation of RT systems require attention to numerous problems. These include [15]:

- The selection of hardware and software, and evaluation of the trade-off needed for a cost-effective solution, including dealing with distributed computing systems and the issues of parallelism and synchronisation.
- Specification and design of RT systems and correct representation of temporal behavior.
- Understanding the nuances of the programming language(s) and the RT implications resulting from their translation into machine code.
- Maximising of system fault tolerance and reliability through careful design.
- The design and administration of tests, and the selection of test and development equipment.
- Taking advantage of open systems technology and inter-operability. An open system is an extensible collection of independently written applications that cooperate to function as an integrated system. For example, a number of versions of the open OS, Linux, have emerged for use in RT applications. Inter-operability can be measured in terms of compliance with open system standards, such as the CORBA RT standard.
- Finally, measuring and predicting response time and reducing it. Performing a schedulability analysis, that is, determining and guaranteeing deadline satisfaction, a priori, is the focus of most of scheduling theory.

1.3.2 Embedded Real-Time Operating System

Real-time Operating System (RTOS) allows RT applications to be designed easily. Functions can be added without major changes to the software. The use of an RTOS simplifies the design process by splitting the application code into separate tasks. With a preemptive RTOS all time critical events are handled as quickly and efficiently as possible. An RTOS allows one to make better use of the system resources by providing valuable services such as semaphores,mailboxes, queues, time delays, time outs, etc.

A recent study [9] concluded that 95% of real-time applications require a bounded response time in the range of 0.5 ms to 10 ms. Only one tenth variance (50 us to 1 ms jitter) in the response time can be tolerated. By these measures, most general purpose operating systems are not real-time [9].

The Kernel code in an RTOS is written so that processor interrupts are only disabled for very short periods of time. The maximum interrupt response time (latency) is a key factor in the response time of an RTOS. An interrupt is a hardware mechanism used to inform the CPU that an asynchronous event has occurred. When CPU recognizes an interrupt, it saves its context and jumps to a subroutine known as Interrupt Service Routine (ISR). Each OS needs to disable interrupts from time to time to execute critical code that should not be interrupted.

1.4 Model Based Design

Model based design (MBD) is a mathematical and visual method of addressing problems associated with designing complex control [26], signal processing and communication systems. It is used in many motion control, industrial equipment, aerospace, and automotive applications. MBD is a methodology applied in designing embedded software. Four steps are necessary in the process [26]:

1. Build the plant model;
2. Analyze the plant model and synthesize a controller for it;
3. Simulate the plant and controller together and
4. Deploy the controller.

MBD process is based on a work flow known as the ’V’ diagram, which is illustrated in Fig. 1.3.

1.4.1 Purpose and Advantage of MBD

ES engineers are also very familiar with problems such as the following, which may destroy the results of the project:

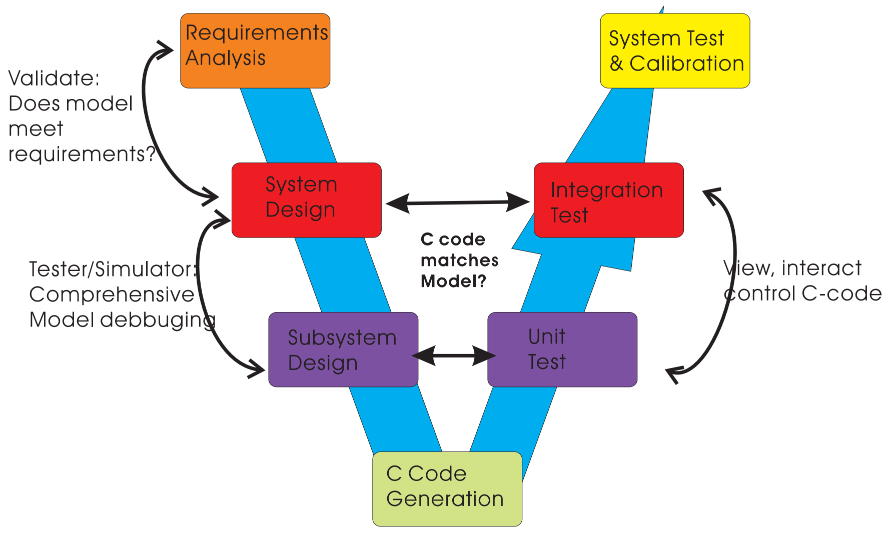

Requirements Analysis
System Test & Calibration
Validate: Does model meet requirements?
System Design
Integration Test
C code matches Model?
Tester/Simulator: Comprehensive Model debbuging
View, interact control C-code
Subsystem Design
Unit Test
C Code Generation

Figure 1.2: A model based design process (http://www.reactive-systems.com/images/mbdProcess.1.png)

- The software engineers interpret to write code in according to the specifications which are defined by the system engineers and the system will not correspond to the original concept.
- The testing cost on expensive hardware should be reduced but the system performance must be optimized.
- You work with R&D teams, which design efficient algorithms and it's hard to let the teams cooperate and share the designs.

MBD handle all the problems, which delay the projects and offer many advantages, such as [5]:

- This model becomes an executable specification and includes all the information needed to specify the software or hardware implementation, including fixed-point and timing behavior.
- Simulation is used to show that the model is complete and works correctly.
- All the code and test benches for final system testing, verification, and deployment can be automatically generated from the model, saving time and avoiding the introduction of errors.

- Model-Based Design reduces the complexity of large-scale system development by letting multiple design teams work in parallel to refine and optimize subsystem designs. As each subsystem is finalized, it can be incorporated into the overall system model.

Executable Specifications The traditional specifications are written on paper, which leads to misinterpretation among different teams. Also hand-written code is too large and leads to errors, that will spread through the process. By sharing the same model as executable specification can get rid of all these problems.

Design with Simulation Since developing hardware prototypes is expensive and costs much time, in addition a multiple prototype is hard to create, the MBD shows the advantage of providing a comprehensive description of design. In this way, they can find and correct problems much earlier in the process and at the same time update automatically the executable specification.

The model provides continuity in the design and development process and can be used to test any optimizations proposed by the implementation team. As a result, the final product will be much closer to the original design [5].

Implementation with Automatic Code Generation The transferring plans to an implementation team can invite mistakes. Models in this respect shows the advantage again. It can be automatically generated into code for RT prototyping and implement in the target system. So that the hand-code error will be also avoided and save the time on coding.

Continuous Test and Verification If the engineers find out problems in this final step before production, they will face a bad situation, meanwhile the model provides a better way. Software models must be tested as often as possible. simulation models allow testing of control software earlier in the design cycle and more frequently during the development.

1.4.2 Relevance to Automation

In automation technologies modern languages such as C, C++, Python and so on, are not preferred, since written code is error-prone and hard to understand for people who did not write it. Special automation languages according to the standard IEC-1131 had no further development and are not suitable to utilise the computing power of modern processors.

Engineers of automation are not software experts and prefer a configuration rather than a programming job. Model based design supports this development approach.

Chapter 2

A New Support for Hardware Platforms

After introducing to the ES design, we know that it's highly demanding of engineers: Good knowledge of hardware and the operating system, and being skilled in at least one computer programming language. This makes the engineers who doesn't major in computer science step back. In the spring of 2012, Matlab announced that Simulink models can run directly on Arduino, Beagle Board, and LEGO MINDSTORMS NXT platforms [29]. This means, the engineers can now benefit from creating and running the model to test and tune their automatic control algorithms much easily without knowing the ES. Specially for students who learn control theory can have industry best practices for DSP, robotics, and mechatronics design. From now on, this new built-in support for the hardware platforms will be gone to particulars.

2.1 System Configuration

Fig. 2.1 shows a host-target cross development platform. In this manner a working environment for the ES design can be built.

The target hardware is booted from a RT-kernel and works with the model from the Simulink application of a non RT host PC and a C compiler also from the Simulink to build the RT target application, which can run on the target hardware once and is downloaded from the host.

This RT-kernel on the target PC initiates the host-target communication, activates the application loader, and waits for the target application to be downloaded from the host

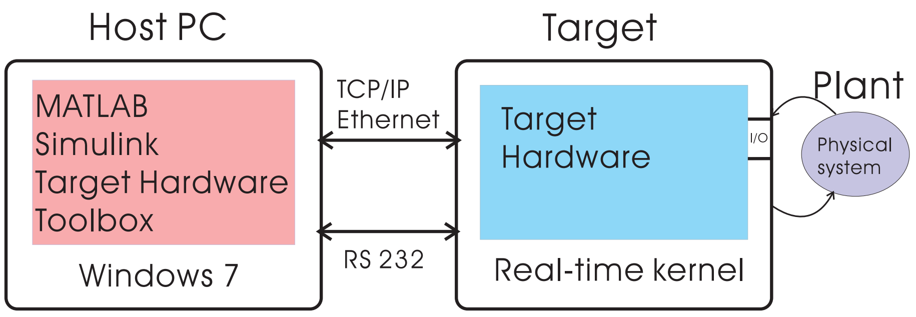


Figure 2.1: System Configuration

PC [17]. This communication can through either serial or TCP/IP protocols. Once the application has been written on the target hardware, it can be controlled and modified from the host.

2.2 Target Hardware Configuration

First the hardware site will be introduced. The new support provided only for some low-cost hardware platforms, such as Arduino, Beagle Board, and LEGO MINDSTROMS NXT. This work refers to two of them, which are Arduino UNO and Beagle Board XM.

2.2.1 Arduino

Arduino is an open source platform, which consists of a wiring I/O board with many connection capabilities and a programming environment similar to C++.

Arduino began as a project in Italy in 2005 and since then it has widely spread. There are thousands of Arduino projects today such as electric meters, guitar amplifiers and Arduino-based gadgets that can tell you when your plants need water.

The wiring I/O board can control sensors and actuators. Sensors allow the board to get information from the surrounding environment (temperature, distance etc). Actuators are devices that used for changing in the physical world (servos, motors, etc). So it can be considered for robotics, mechatronics, and hardware-connectivity tasks.

Boards

The major component of Arduino is its Board, which based on micro-controller from Atmel. In this thesis the latest in a series of USB Arduino boards UNO is used as the target system.

The picture below (Fig. 2.2) is the UNO board, on which the main components of the board is labeled.

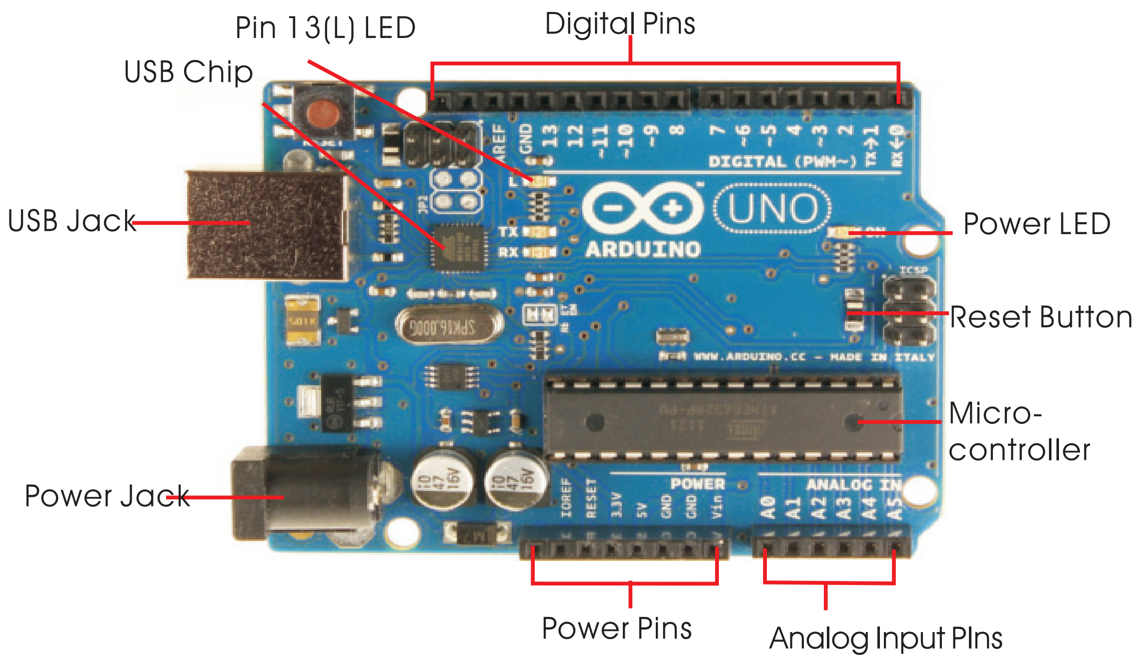


Figure 2.2: Arduino UNO

UNO connects to the computer with a standard USB cable and contains everything else you need to program and use the board. It can be extended with a variety of shields: custom daughter-boards with specific features. It is similar to the Duemilanove board, but has a different USB-to-serial chip the ATMega8U2, and newly designed labeling to make inputs and outputs easier to identify [4].

Shields

Shields are boards that can be plugged on top of the Arduino PCB extending its capabilities. For example [4]: Xbee shield:

The Arduino Xbee shield allows multiple Arduino boards to communicate wirelessly over distances up to 100 feet (indoors) or 300 feet (outdoors) using the Maxstream Xbee Zigbee module.

Motor Control v1.1:

The Arduino Motor shield allows you to control DC motors and read encoders.

Under this website http://shieldlist.org/ more shields can be found.

In this work the TinkerKit sensor shield is added on the Uno board: Fig. 2.3 and Fig 2.4.

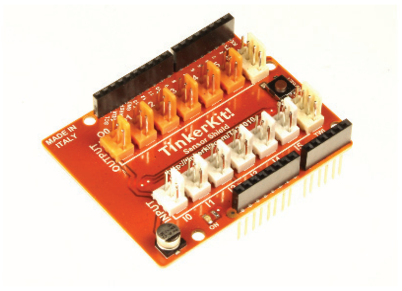


Figure 2.3: TinkerKit sensor shield

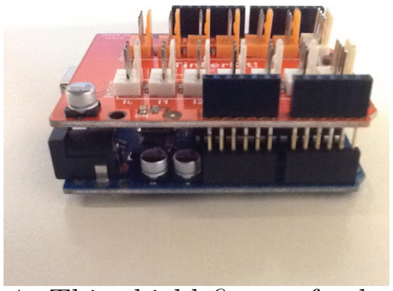


Figure 2.4: This shield fits perfectly upon an UNO.

The Sensor Shield allows the user to hook up the TinkerKit Sensors and Actuators directly to the Arduino, without the use of the breadboard.

It has 12 standard TinkeKit 3 pin connectors. The 6 labeled I0 through I5 are Analog Inputs. The ones labeled O0 through O5(which equal to Pin 11 through Pin 3 on the Arduino) are Analog Outputs connected to the PWM capable outputs of the Arduino Board.

The peripherals

The peripheral hardware consists of a variety of visual, Audio, mechanical, electrical inputs and outputs possibility for Arduino. More details can be found under: http://www.arduino.cc/playgrour

2.2.2 Beagle Board

The Beagle board is a low-power, single-board computer with all expandability of today's desktop machines, whose latest version is based on the same 1GHz ARM Cortex A8 processor that drives the most sophisticated smartphones today.This gives it far more processing power than the Arduino. It applies mainly for audio, video, and digital signal processing, which is totally different from the Arduino. But also it has application possibilities such as robotics, home automation, web services, 3D gaming, etc. A modified version of the Beagle Board called the Beagle Board-XM started shipping in 2010, that will be used in this work. Fig. 2.5 shows this version of Beagle Board.

Specifications [1]:

1. USB OTG Connection, which allows USB devices such as digital audio players or

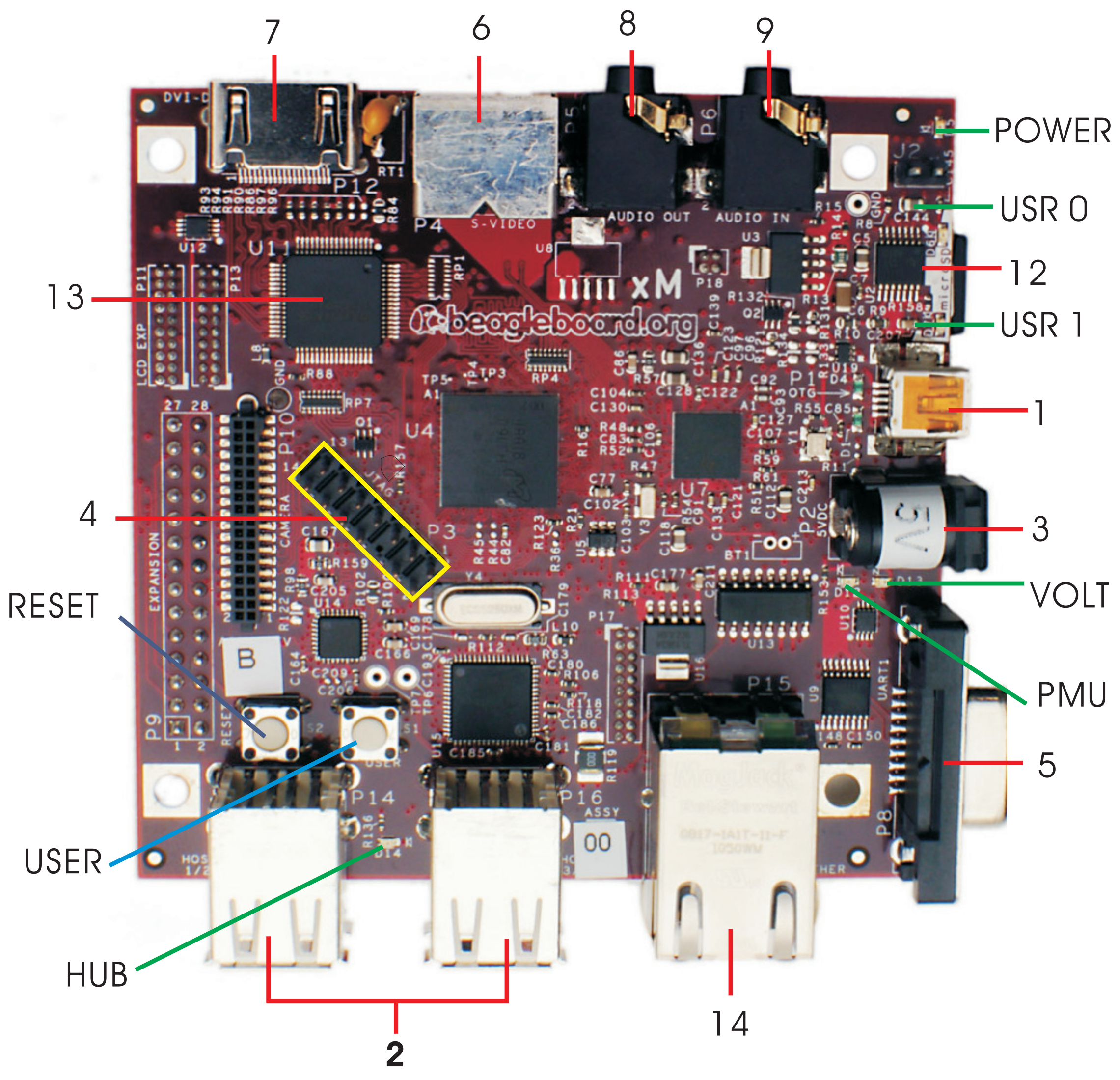


Figure 2.5: Beagle Board XM

mobile phones to act as a host allowing a USB flash drive, mouse, or keyboard to be attached and also connecting USB peripherals directly for communication purposes among them.

2. USB Host Connection.

3. DC Power Connection.

4. JTAG Connection was initially devised for testing PCB using boundary scan and is still widely used for this application.

5. Serial Cable Connection is the process of sending data one bit at a time, sequentially over a communication channel or computer bus.

6. S-Video Connection: Separate video is an analog video transmission scheme, in which video information is encoded on 2 channels: intensity and colour. S-video carries standard definition video, but does not carry audio on the same cable.

7. DVI-D Connection: The digital interface is used to connect a video source to a display device, such as a computer monitor.

8. Audio out Cable Connection.

9. Audio in Cable Connection.

10. Indicator Locations

    5 green and 1 red indicator

    POWER: power is applied to the board.

    USR 0/1: can be used by the SW as needed.

    PMU: controlled from the power management chip and can be connected to a PWM.

    VOLT: will turn on when the DC voltage exceeds specification.

    HUB: turns on when power is applied to the UAB HUB.

11. Button:

    R: Reset and

    U: User can be used by the SW for user interaction.

12. Micro SD: boot source for the board.

13. LCD Connection.

14. Ethernet Connection.

2.3 Real-Time Operating System

As mentioned before, that the OS supports a variety of open standard application programming interfaces. The Arduino actually has no OS, or someone has to write an OS himself. So in this section only the OS for Beagle Board will be covered in detail.

2.3.1 Operating Systems on Beagle Board

Beagle Board gives you several choices for Operating System [8]:

1. Angström Distribution (Most widely used and recommended)
2. Android for Beagle Board (Google Mobile System)
3. Ubuntu on the Beagle Board (which used in this thesis)
4. Windows Embedded
5. QNX Neutrino

Ubuntu's parent company, has dedicated resources to porting Ubuntu to ARM processors such that used on the Beagle Board. The Linux-realtime package *eLinux* gives RT capabilities to Ubuntu.

Booting refers to the process by which a newly reset computer prepares itself to load an operating system. Modern desktop computer usually do this process with the BIOS, which automatically discovers and initializes the hardware. Beagle Board use a different, multi-step process because of constrained memory. This process often takes three steps [22].

1. A small boot loader, which fits neatly into the small ROM provided on the system, locates and loads a larger,
2. Boot loader into RAM,
3. Boot loader initializes the system and boots the operating system.

Booting Ubuntu To boot Ubuntu it is necessary to download the pre-compiled binary image. Then insert the micro SD card in the USB card reader of the host PC and determine its device name. Next write the image directly to the card and finally insert this card into the Beagle Board. All these can step by step tutorial on simple installing a Linux on the Beagle Board with Matlab 2012a to build the Ubuntu image on the board. This will be introduced in section 2.5.

Linux kernel of Ubuntu A kernel is a central component of an operating system. If some useful applications and utilities are added over the kernel, then the complete package becomes an OS. Linux is a kernel as it does not include applications like file-system utilities, windowing systems and graphical desktop, system administrator commands, text editors, compilers etc. So, various companies add these kind of applications over Linux kernel and provide their operating system like Ubuntu. Is the Ubuntu Linux kernel running on Beagle Board, provided by Simulink support for Beagle Board hardware (of section 2.5), a "real-time" kernel? The answer is yes but not a hard real-time kernel as defined [13]. The interrupt response times are not deterministic. Assumed the user needs a system that can process video at 30 fps or process an audio stream without interruptions. The Beagle Board can do both. For control applications, the Beagle Board can probably handle a 1KHz loop, but anything below that is probably not possible [6].

2.4 Host Target Connection

Arduino has no OS and its connection through the USB port is simple, so this is not going to be described here. It is also possible for Arduino to connect the host through serial port, which won't be discussed in this work. This section will only show the connection between Beagle Board XM and the host PC.

2.4.1 Components of Beagle Board

The Beagle Board XM is packaged with a pre-formatted 4GB micro SD card and a 5V external power supply but no cables. For connection the components below are needed:

- A desktop or laptop computer with a serial port as host PC.
- A serial cable

  The target PC provides a female DB9 port.

  If there is no COM port on the host computer, you can use a USB to serial cable.
- An ethernet cable.
- 4GB+ micro SD cards

  Use the one, which has been already overwritten with the Ubuntu as OS.

- Monitor, USB keyboard and mouse

  in order to show display and operate the target PC with Ubuntu distribution

2.4.2 Connecting the Components

When all the parts have been collected, it's time for plugging things in.

1. Connect a serial cable between the board an the host computer.
2. Connect an Ethernet cable to the board, Connect the other end of the Ethernet cable to a network or directly to the host computer.
3. Plug the DVI-D end into the monitor and the USB keyboard and mouse into the Hub.
4. Connect a 5 Volt power supply.

Figure 2.6 shows the connected components.

2.4.3 Setting up the console

Since in this case 64-bit Windows 7 is host PC for running the programming software Matlab, so the first step is to set up a Windows-based development environment for Beagle board.

Setting up the console on Windows

Connect host computer to Beagle board via serial terminal A serial console can use to:

- View the standard output while the board boots.
- Get or set the IP address, as described in Get IP Address of Beagle Board Hardware topic (which will be used in the next paragraph).

To open a serial connection to your board [28]:

- Identify the COM port for your serial connection. In Windows 7, use the "search programs and files" feature in the Start menu to find "Device Manager". Open Device Manager, and expand Ports (COM & LPT) to see the list of serial connections.

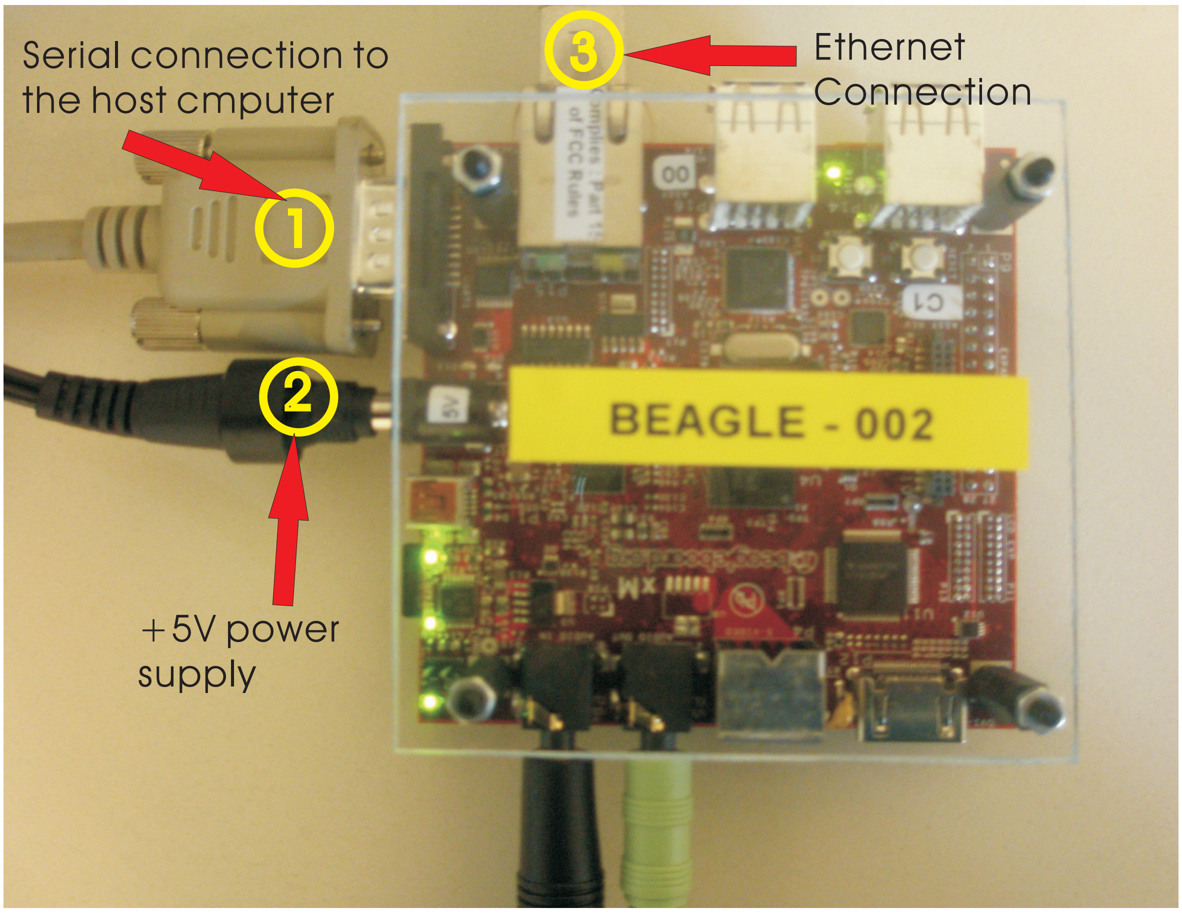


Figure 2.6: The connected Beagle Board

- Start PuTTY by typing 'Putty' at the 'Search Programmes and files' box in the Start menu and click the
- In the PuTTY window, select the Serial Category and configure the selected COM port to:

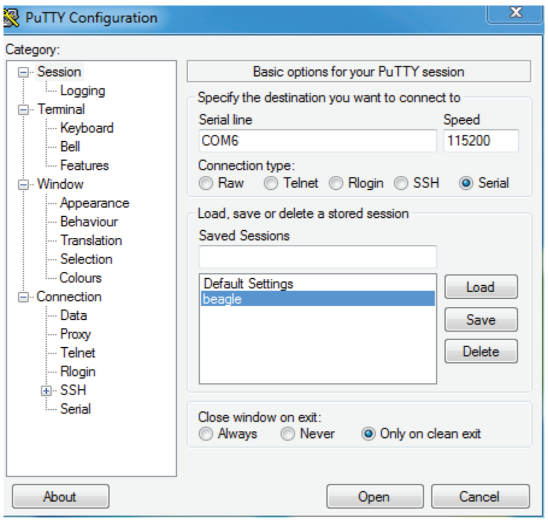
  Serial line to connect to: COM# (in this case, it is COM6)

  Speed (baud) 115200

  flow control: None

  look at the Fig. 2.7
- In the PuTTY dialog box, select the Session category.
- Set the Connection type parameter to Serial, enter a new name for Saved Sessions, click Save, and click Open (see Fig. 2.8).

- When a terminal window opens, press the Enter. Then type in the user name ubuntu and the password temppwd to login.



Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM6
Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

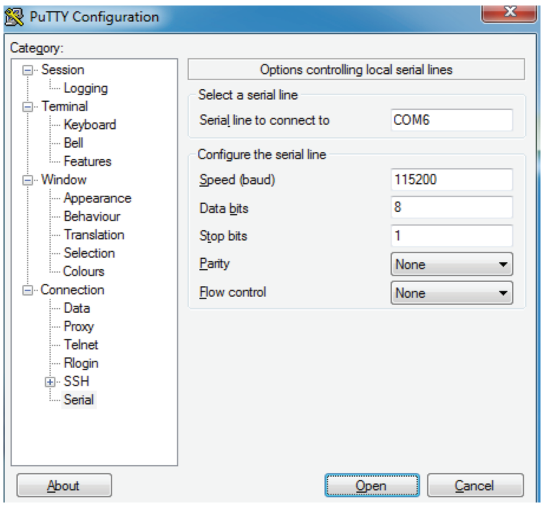
Load, save or delete a stored session

Saved Sessions

Default Settings
beagle

Close window on exit: Always Never [✓ Only on clean exit]

Figure 2.7: Configure the COM port in Putty



Options controlling local serial lines

Select a serial line

Serial line to connect to: COM6

Configure the serial line

Speed (baud): 115200
Data bits: 8
Stop bits: 1
Parity: None
Flow control: None

Figure 2.8: Serial connection to the target with Putty

Configure Network Connection with Beagle Board Hardware To set up a console under Windows XP, Vista or Win 7 operating system , SSH Secure Shell Client is recommended to download, a free program for windows. Setting up an SSH server on the board allows to open up a terminal on any Internet connected computer as if it was opened locally on it and be able to log into the board and copy a file from the network back to the local

PC. The SSH Secure Shell Client is a client for secure, encrypted communication, including file transfer, across networks.

Before transfer the file at SSH Secure Shell Client, the IP configuration needs to be set up at both Beagle Board and the host PC.It is assumed that the board has a static IP. For setting up static IP, these steps are required under the terminal(which named konsole of the system tools' submenu) in the Ubuntu applications [27]:

- Display the contents of the /etc/network/interfaces file. Enter:

  ```
  cat /etc/network/interfaces
  ```

  If the board is configured to use DHCP services (the default configuration), dhcp appears at the end of the following line: iface eth0 inet dhcp

  If the board is configured to use static IP settings, static appears at the end of the following line: iface eth0 inet static

- Create a backup of the /etc/network/interfaces file. Enter:

  ```
  sudo cp /etc/network/interfaces /etc/network/interfaces.backup
  ```

  Enter the root password when prompted (default: temppwd).

- Change the permissions of /etc/network/interfaces so you can edit the file. Enter:

  ```
  sudo chmod 777 /etc/network/interfaces
  ```

- Edit interfaces using a simple editor called nano. Enter:
- nano interfaces
- Edit the last word of line that starts with iface eth0 inet.

  To use DHCP services, edit the line to say:iface eth0 inet dhcp

  To use static IP settings, edit the line to say:iface eth0 inet static

- For static IP settings, add lines for address, netmask, and gateway. In this case:

  ```
  iface eth0 inet static
  address 193.170.30.144
  netmask 255.255.255.0
  gateway 193.170.30.17
  ```

- Tell nano to exit and save the changes:

  Press Ctrl+x.

  Enter Y to save the modified buffer.

  For "File Name to Write: "interfaces", press Enter.

  The nano editor confirms that it "Wrote # lines" and returns control to the command line. Restore the original file permissions. Enter:

  sudo chmod u=rw,g=r,o=r interfaces

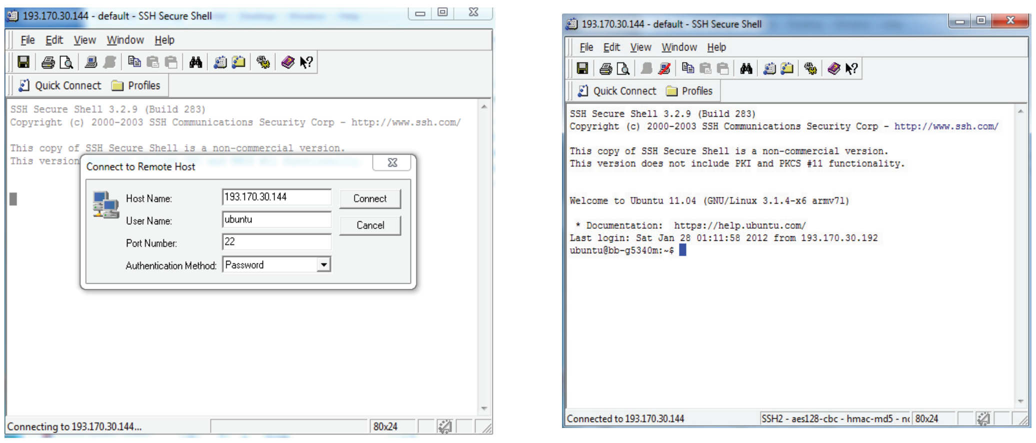
Installing the SSH Server is a matter of installing after having the static IP address: *sudo apt-get install ssh* type this command in the terminal window of the Ubuntu desktop.

To connect

- After Installing this secure software click on the Connect computer icon, or go to the File –> Quick Connect.
- Type in your host name (static IP address)and user name (Ubuntu).
- Make sure the Port Number is 22.
- Click Connect.
- At the prompt, type in your password (temppwd for here)and click OK.
- You will then be at a Unix command line and can work from there. Fig. 2.9.

2.5 Software Configuration

Some softwares suites for ES, specially the MBD software is preferred base on its full advantage. The engineers aim at using the model to do not only simulation but also implementation. In the past Simulink offers this possibility by embedded code generation, which can produce the code like C/C++ for the Embedded RT target. But it left still a complicated work to do. For example, some engineers have limited experience of Linux, since a large number of ES base on Linux OS. So they need to use cross compiler on the host platform which use Windows OS for the target Linux OS execution. Some other need a IDE (Integrated Development Environment) such as Eclipse, which is a programming environment



193.170.30.144 - default - SSH Secure Shell

```
SSH Secure Shell 3.2.9 (Build 283)
Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/

This copy of SSH Secure Shell is a non-commercial version.
This version
```

Connect to Remote Host

Host Name: 193.170.30.144
User Name: ubuntu
Port Number: 22
Authentication Method: Password

Connecting to 193.170.30.144... 80x24

193.170.30.144 - default - SSH Secure Shell

```
SSH Secure Shell 3.2.9 (Build 283)
Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/

This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.


Welcome to Ubuntu 11.04 (GNU/Linux 3.1.4-x6 armv7l)

 * Documentation:  https://help.ubuntu.com/
Last login: Sat Jan 28 01:11:58 2012 from 193.170.30.192
ubuntu@bb-g5340m:~$
```

Connected to 193.170.30.144 SSH2 - aes128-cbc - hmac-md5 - nc 80x24

Figure 2.9: SSH connection with the target.

that has been packaged as an application program, typically consists of a code editor, a compiler, a debugger, and a graphical user interface (GUI) to import the generated code into the IDE. Someone else needs at least to write the drive to control the I/O devices on the target. An example of the Beagle Board with Simulink has been done before [3]. From this can be found out that this work before is not as simple as it looks. It based on much knowledge about hardware and software. So the MATLAB new support provides a way to develop the ES for simulation, control and implementation. In this manner Simulink models can run directly on some hardware platforms with one click of a mouse. In other hand, it offers also automatic booting stage, which saves both time and energy to learn the hardware and OS.

2.5.1 Installation

The Target Installation

The new support for the target can only installed in MATLAB 2012a and 2012b. 3 ways can be used to get this support package:

1. In a MATLAB Command Window, enter targetinstaller and the package will be automatically downloaded.
2. In a model, select Tools > Run on Target Hardware > Install/Update Support Package.

3. Download the support package under

   http://www.mathworks.cn/matlabcentral/fileexchange/37897

With the Targetinstaller or Install/Update Support Package the friendly installation proceeds step by step both for Arduino and Beagle Board. It's very user-friend installation, so will not be discussed in details here and can be seen in the appendix. For Arduino that's all before running Simunlink, but for Beagle the factory-installed firmware must be replaced on the Beagle Board hardware with Ubuntu Linux.

Replacing Firmware

The following steps provide an overview of the firmware replacement process [30]:

1. The Target Installer locates a firmware image on your host computer or downloads new one.
2. The Target Installer uses the host computer to write the firmware image to a micro SD or SD memory card.
3. You transfer the micro SD or SD memory card to the Beagle Board hardware.
4. The Target Installer applies the IP settings you choose to the firmware on the Beagle Board hardware.

Details can also be found in appendix.

2.5.2 The New Toolbox for Target in Simulink

After installing support for your Arduino hardware, its block library can be opened from the command window or from the Simulink Library Browser.

Arduino

The new Simulink Blocks of Arduino target included blocks listing below:

- Arduino Analog Input: Measure voltage of analog input pin,
- Arduino PWM: Generate PWM waveform on analog output pin,

- Arduino Digital Input: Get logical value of digital input pin,
- Arduino Digital Output: Set logical value of digital output pin,
- Arduino Serial Receive: Get one byte of data from serial port,
- Arduino Serial Transmit: Send buffered data to serial port,
- Arduino Standard Servo Read: Get position of standard servo motor shaft in degrees,
- Arduino Continuous Servo Write: Set shaft speed of continuous rotation servo motor and
- Arduino Standard Servo Write: Set shaft position of standard servo motor.

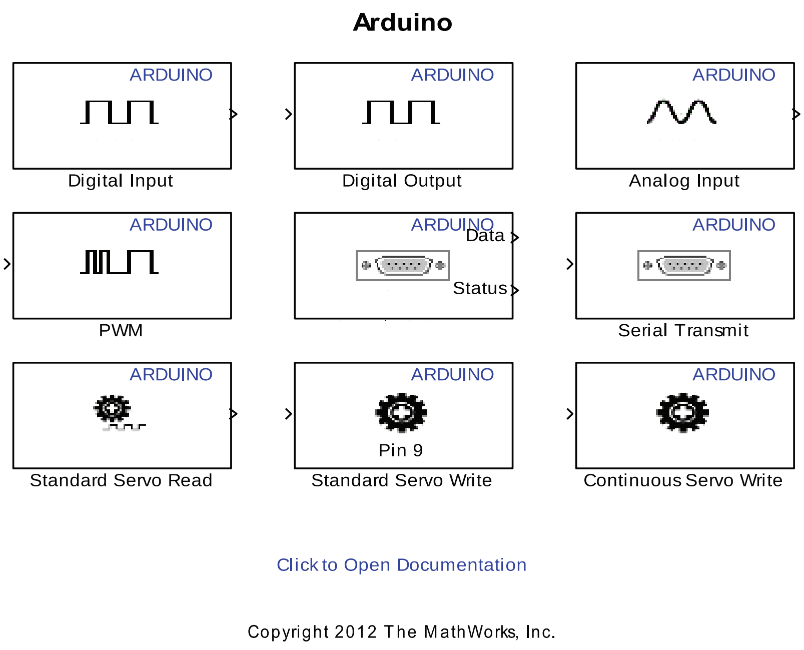


Figure 2.10: Simulink blocks of Arduino taget hardware

Beagle Board

The new Beagle Board support blocks provides the capability of:

- Beagle Board ALSA Audio Capture: Capture audio from sound card using ALSA,
- Beagle Board ALSA Audio Playback: Send audio to sound card for playback using ALSA,

- Beagle Board SDL Video Display: Display video using SDL,
- Beagle Board UDP Receive: Receive UDP packets over IP network,
- Beagle Board UDP Send: Send UDP packets over IP network,
- Beagle Board V4L2 Video Capture: Capture video from USB camera using V4L2.

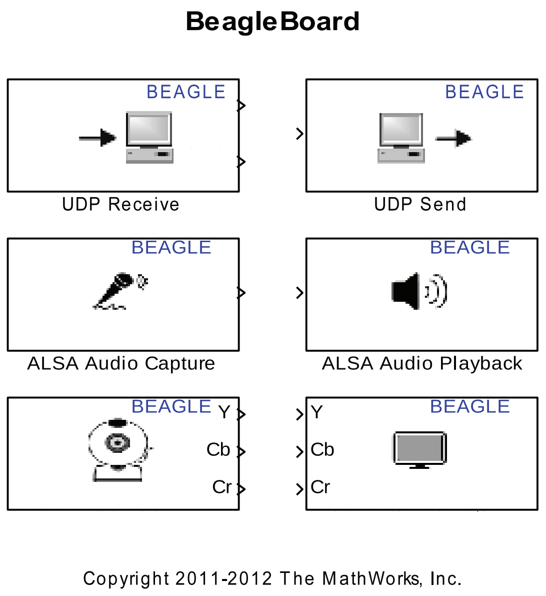


Figure 2.11: Simulink blocks of Beagle Board taget hardware

2.5.3 C-Compiler for Win 7

The C-compiler is required by stateflow and MATLAB-function blocks, if the Host PC has installed with 64 bits Win 7 OS, the error massage will be appeared by running the models with stateflow and M-func. This website: http://www.mathworks.cn/support/compilers/R2012a/win64.htm describe the complete set of supported compilers, among which the SDK is most recommended. Before install this C-compiler the Visual C++ 2010 must be uninstalled, otherwise will cause problem. After the installation the command 'Mex-setup' should be typed in MATLAB window to select this supported C-compiler.

2.6 Interaction between Host and Target

In this section 2 demos will run to test the installation and the interaction between the host and target PC. The demos are under Simulink Help in the Library Browser Help menu. Then look for the "Other Demos", which contains two submenu about the help ,with that the user can get stated with the target hardware. Since in chapter 3 an Arduino simple example model will be written and executed, it's not necessary to test the demo in this case. Later the Beagle Board will be taken to do a real example for the industrial purpose, which will be reported in chapter 4, so first a simple demo model will be introduced for getting use to its development environment and making sure the interaction between the two PC working in a proper way.

2.6.1 The Image Inversion Demo Test with Beagle Board

Before running the Simulink the camera must be connected to the USB port of the Beagle Board. The Logitech Webcam Pro 9000 is confirmed to be supported by the target package. The other connections are the same as written before in section 2.4. Paying particular attention that the SSH connection must done before running the model. The whole system looks the same as in the Fig. 2.12.

Then open the Image Inversion model as the Fig. 2.13 :

the next thing is to run on the hardware configuration, which can be selected under Tools > Run on Target Hardware > Options... The parameters on this page should be reviewed and set up. If the step 'Replace Firmware' of installation has been performed, the information of the Board will be automatically showed like the pictures (Fig. 2.14) below.

Finally select **Tools** > **Run on Target Hardware** > **Run** to run the model on the target hardware. It takes about one minute until the compiler, download etc. finish. After that a terminal window appears. The whole precess can be watched in SSH Secure Shell or the terminal of Ubuntu by typing the command 'ps -A'.

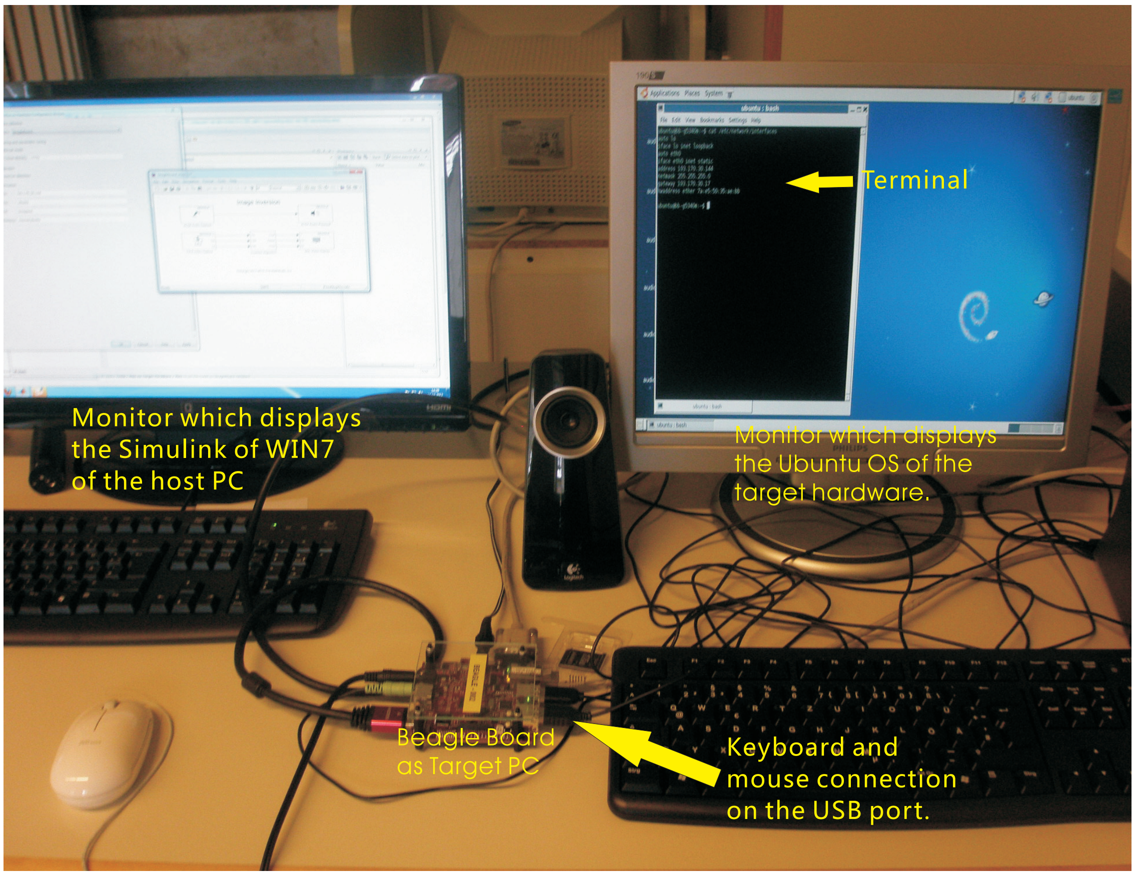

Monitor which displays the Simulink of WIN7 of the host PC
Terminal
Monitor which displays the Ubuntu OS of the target hardware.
Beagle Board as Target PC
Keyboard and mouse connection on the USB port.

Figure 2.12: The target host connection for the demo

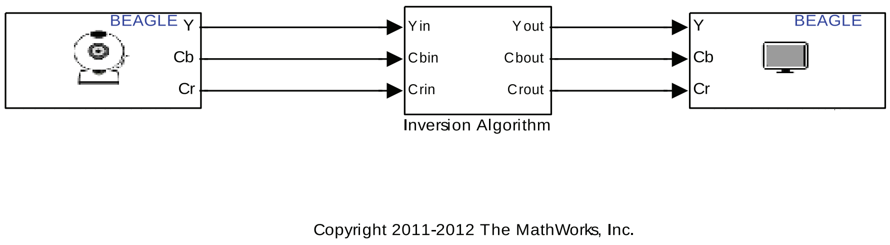
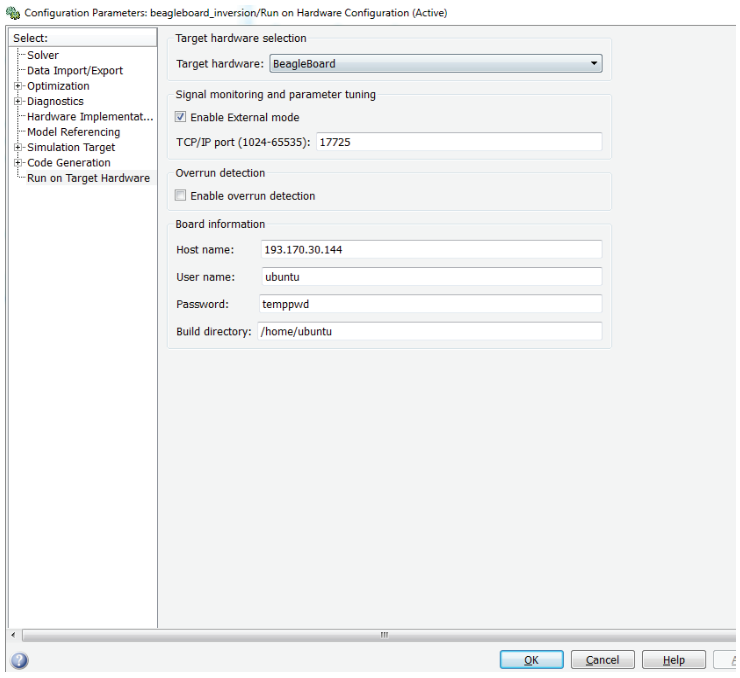


Figure 2.13: Image Inversion model which will run on Beagle Board


Configuration Parameters: beagleboard_inversion/Run on Hardware Configuration (Active)

Target hardware selection

Target hardware: BeagleBoard

Signal monitoring and parameter tuning

☑ Enable External mode

TCP/IP port (1024-65535): 17725

Overrun detection

☐ Enable overrun detection

Board information

Host name: 193.170.30.144

User name: ubuntu

Password: temppwd

Build directory: /home/ubuntu

Figure 2.14: The configuration before running the model

Chapter 3

Implementation of Arduino as Platform

Arduino is a hardware platform, which has many inputs and outputs pocities as mentioned before. In this chapter an example is presented from connection of the sensor and actuator with the board to control the RT system.

3.1 Peripherals

In this example a servo motor is going to attend. Servo is defined as an electrical device in which an electrical input determines the angular position of the shaft as an output.

Standard servos allow the shaft to be positioned at various angles, usually between 0°and 180°. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.

3.1.1 Servo Control Principle

Servos are comprised of an electric motor mechanically linked to a potentiometer. Arduino controls the servo using the Servo Library. Pulse-width modulation (PWM) signals sent to the servo are translated into position commands by electronics inside the servo. When the servo is commanded to rotate, the motor is powered until the potentiometer reaches the value corresponding to the commanded position.

A standard servo will move based on pulses sent over the control wire. The control pulses set the angle of the servo axis. The servo expects a pulse every 20 ms in order to gain correct information about the angle. The pulse width maps directly to the servo angle. The servo will rotate 180°and expect pulse widths between 1-2 ms or so [14].

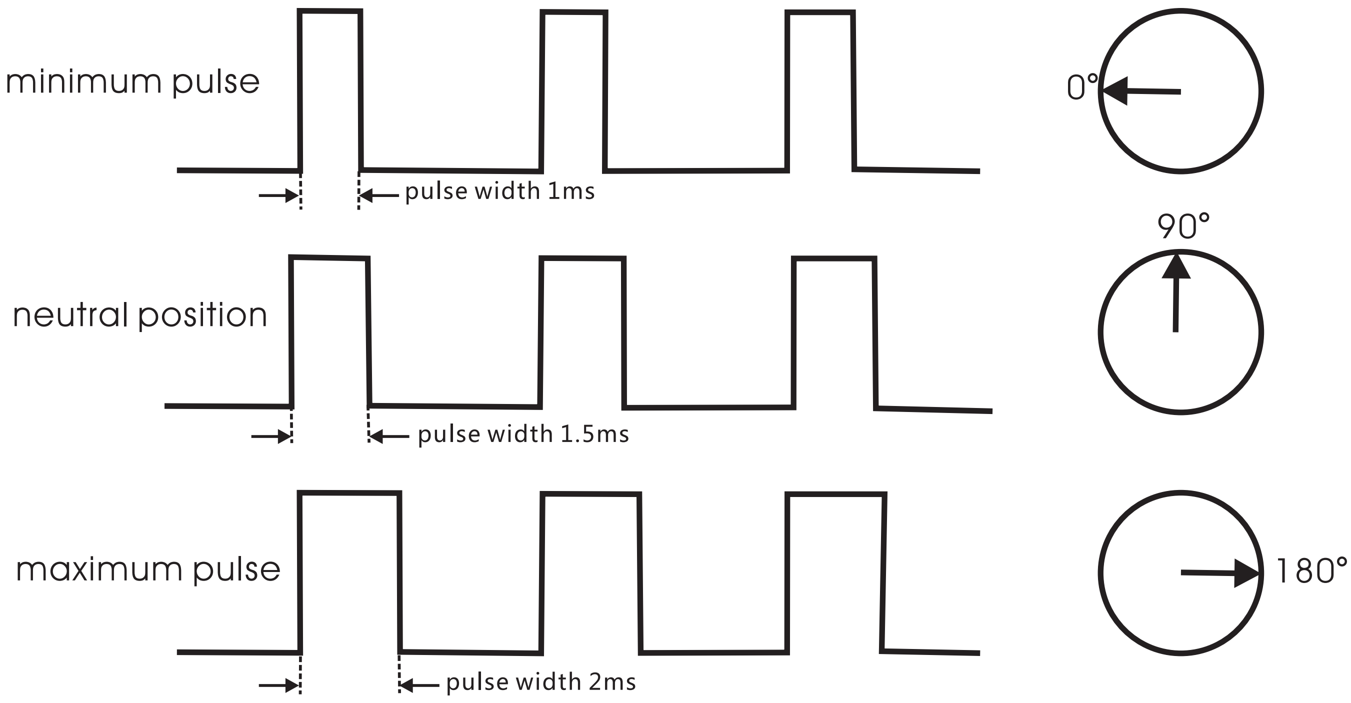


Figure 3.1: Control Principle of a servo motor

3.1.2 Connection of the Peripherals with the Arduino

The other peripheral is the joystick, whose funciton here is as a sensor to provide some analog input signals. Both the peripherals are sold as modules and can easily be mounted on the TinkerKit shield. All the wire connection work can be saved, the only thing to do is plugging the 3pin connector to the sensor shield. The joystick's connectors are outputs which must be connected to the input connectors on the Thinkerkit shield. It has 12 standard TinkeKit 3 pin connectors. The 6 labeled I0 through I5 are Analog Inputs. The ones labeled O0 through 05 are Analog Outputs connected to the PWM capable outputs of the Arduino Board [24]. Fig. 3.2 shows the completely system connected system.

3.2 Model implementation on Arduino

The idea in this example is that the servo will read the analog values from the joystick. The block 'Analog Input' should be connected to the 'Servo Write'. One thing to notice is setting

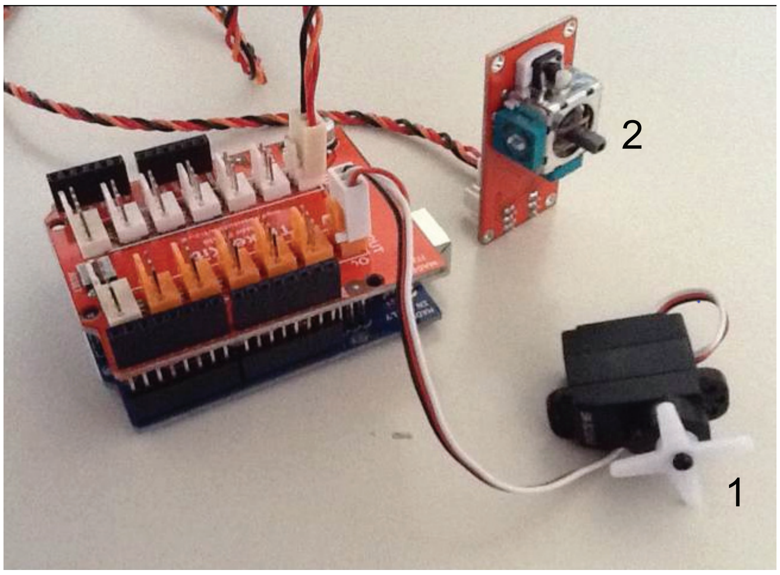


Figure 3.2: Servo motor (1) and joystick (2) with Arduino

pins on Arduino board	pins on the shield
Pin 11	O0
Pin 10	O1
Pin 9	O2
Pin 6	O3
Pin 5	O4
Pin 3	O5

Table 3.1: Relationship between pins on the Arduino and pins on the shield

put the pin number of both servo and joystick. The pins have a relation with the pins on the shield:

So the simple model looks like the one showed in Fig. 3.3.

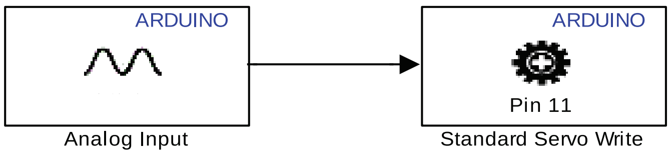


Figure 3.3: The program for joystick controlling the servo

The other option set-up is same as what has been done in the Beagle Board image test before. Then Run on the Target Hardware > Run. After it finished, the servo may be controlled with the joystick.

3.3 Implementation Using Simulink in Contrast With Using Arduino 1.0.1

Instead of analog signals the servo can also accept the digital ones, which can be configured on the PC.

The Arduino 1.0.1 is the actual software version, which is provided by Arduino itself and can be downloaded from its website. There is a program module in this software for the digital signals to make the shaft angle of the servo motor vary between 0 and 180 degrees. The programm will be found in the appendix. Now the same result will be obtained by creating a Simulink model.

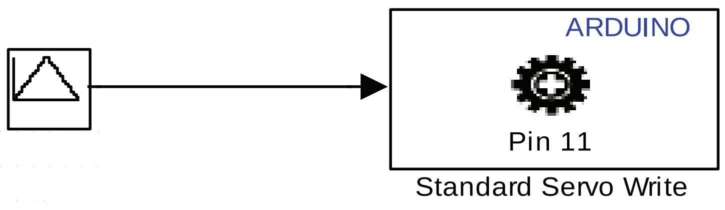


Figure 3.4: Program for controlling the servo with signal sources

When the model starts running on the Arduino Uno, the motor shaft position sweeping 0-180 degrees can be observed.

If someone want to stop the servo, he only need to drag the manual switch block between the signal and servo blocks.

With the external mode it can be halted by double clicking the switch.

conclusion In this example it is obviously to know the advantage of the model based design for Arduino:

- Design directly points at the requirements.

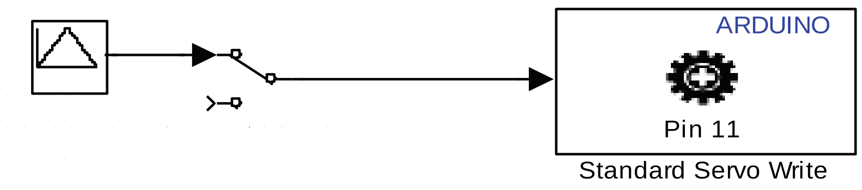


Figure 3.5: Model built to stop the servo

- The workflow is obvious at a glance.
- It takes little time to modify the model and to test continuously identify and correct errors. Even with this simple example, it can be imagined that we benefit from the MBD without manually coded errors.
- This kind of development can save the time to learn the programming language.

Chapter 4

A Machine Condition Monitoring Example on Beagle Board

4.1 Wireless Monitoring of Injection Molding

4.1.1 Introduction

Injection molding is an important manufacturing process for producing parts from plastics. In-mold sensors are usually used to monitor injection molding process, so that high quality parts could be produced. Wired sensors predominate at present. Since wires make the design and maintenance of the mold more difficult, a new sensor concept is presented for wireless monitoring filling of molds [7]. A mechanical resonator is excited and generate a characteristic structure borne sound, when the melt front is passing by. This temporal resolution of the passing melt front is of special interest for detecting the switch over point or for hot runner balancing [12].

4.1.2 Principle of Operation

Fig. 4.1 shows the principle of the operation: The dark area represents the melt filling the cavity (C). When the melt reaches the pin (B), it will be depressed and excites the resonator (A), which injects a characteristic sound into the metal mass of the mold [18].

Mechanical Resonator

For the experimental purposes two different resonators were designed [18]:

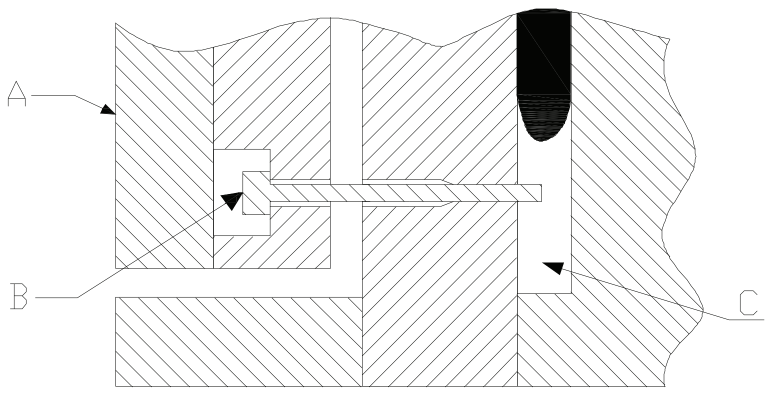


Figure 4.1: The principle of operation

- A plate resonator with a primary resonant mode at 12 kHz and
- a tongue resonator with a primary resonant mode at 3.8 kHz.

Both designs are only differentiated by the resonant body. The two different oscillation characteristics can be separated with those two resonators. Both two resonators generate sounds, which will be acquired by using a single accelerometer.

Acoustic Detection

The accelerometer is mounted on the outside of the mold and detects the structure borne sounds from both of the resonators. The accelerometer has a wide frequency range in order to improve performance of the system with the presence of acoustic disturbances and a high sampling frequency which can be used in the pattern recognition to detect the desired oscillations and the wider range of noise is uncorrelated.

4.1.3 Signal Processing

The purpose is to find out an optimal detector for the resonators. A new algebraic approach [18] has been explored to implement the frequency domain pattern matching. In this work the classical digital signal processing method will be used. A bandpass filter is going to be designed to identify one of the signature sound, meanwhile the exactly occurrence time of it can also be found out. In this manner the two characteristic damped oscillatory behavior could be separated.

4.2 Principle of Digital Signal processing

4.2.1 Introduction

The signals in the real world, such as voice, audio, video, temperature, pressure or position are called analog signals. These signals should be digitized, so that the computer can mathematically manipulate them. While an analog signal is continuous in both time and amplitude, a digital signal is discrete in both time and amplitude. To convert a signal from continuous time to discrete time, a process called sampling is used. The value of the signal is measured at certain intervals in time. Each measurement is referred to as a sample.

Digital processing of analog signals proceeds in three stages [21]:

1. The analog signal is digitized, that is, it is sampled and each sample quantized to a finite number of bits. This process called A/D conversion.
2. The digitized samples are processed by a digital signal processor.
3. The resulting output samples may be converted back into analog from by an analog reconstructor (D/A conversion).

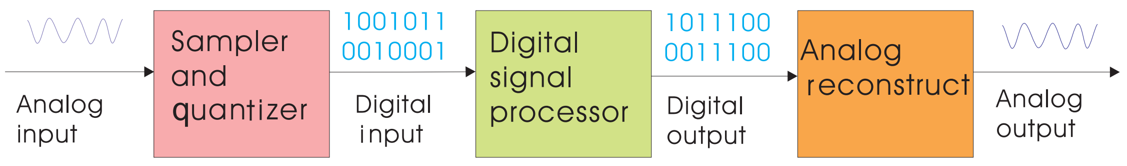


Figure 4.2: A typical digital signal processing system

The digital signal processor can be programmed to perform a variety of operations, such as filtering, spectrum estimation, and other DSP algorithm.

4.2.2 Sampling

The continuous signal varies over time and the sampling process is performed by measuring the continuous signal's value every T units of time, which is called the sampling interval. Sampling results in a sequence of numbers, called samples, to represent the original signal. The reciprocal of the sampling interval $(1/T)$ is the sampling frequency f_s.

For system design purposes, two questions must be answered [21]:

1. What is the effect of sampling on the original frequency spectrum?

2. How should one choose the sampling interval T?

Aliasing

Sinusoids are an important periodic function, because realistic signals are often modeled as the summation of many sinusoids with different frequencies and different amplitudes. In fig. 4.3 a set of samples were plotted with a sample-interval 1, as you can see, two different sinusoids can reproduce the sample, whose sample rate $f_s = 1$. Suppose the interval is 1 second, it means the rate is 1 sample per second. Nine period of the red sinusoid span an interval of 10, that is to say, its frequency is 0.9. In this way the frequency of the blue sinusoid can also be obtained as 0.1.

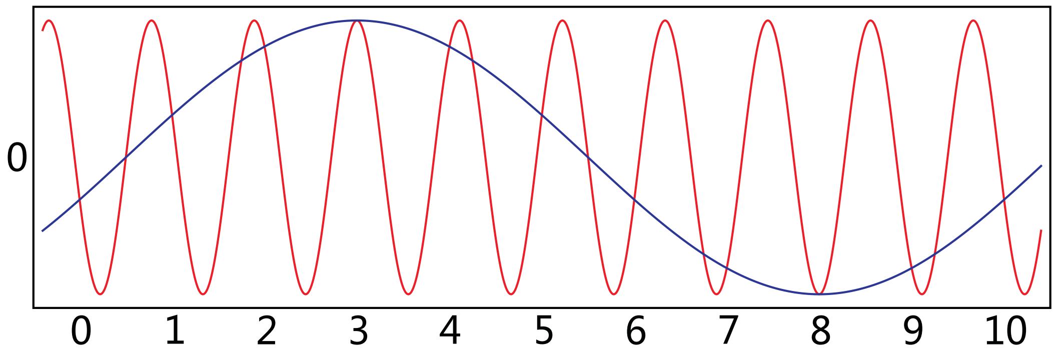


Figure 4.3: Two different sinusoids that fit the same set of samples

When a sinusoid of frequency f is sampled with frequency f_s, the resulting samples are indistinguishable from those of another sinusoid of frequency $(f - N \times f_s\ )$ for any integer N. The values corresponding to $N\neg 0$ [20]. When $N = 1$ for the red sinusoid, the aliased frequency is $0.9 - 1 \times 1 = -0.1$. A negative frequency is equivalent to its positive value, because

$$\sin(-\omega t + \theta) = \sin(\omega t - \theta + \pi) \tag{4.1}$$

$$\cos(-\omega t + \theta) = \cos(\omega t - \theta) \tag{4.2}$$

Then the $N = 1$ alias of fred is f_{blue}. This aliasing will result in the reconstructed signal not matching the original signal.

Nyquist sampling theorem

The Nyquist sampling theorem provides a prescription for the nominal sampling interval required to avoid aliasing. It may be stated simply as follows: The sampling frequency should be at least twice the highest frequency contained in the signal [20].

Or in mathematical terms:

$$f_s \geq 2f_{\max} \tag{4.3}$$

where f_s is the sampling frequency, and f_{max} is the highest frequency contained in the signal.

Hardware Limits

In RT applications, each input must be sampled, quantized and processed by the digital signal processor, and the output will be converted back into analog format. This operation is continuous in order to reduce the time. Anyhow there is a total processing time T_{proc} required for each sample. The time interval T between input samples must b e greater than T_{proc}; otherwise, the processor would not be able to keep up with the incoming samples [21]. Thus,

$$T \geq T_{proc} \tag{4.4}$$

In other words, processing rate f_{proc}=1/T_{proc}. We have already known by Nyquist theorem the lower bound of f_s and now we obtain the upper limit for, so the range of f_s, which can be chosen is:

$$2f_{\max} \leq f_s \leq f_{proc} \tag{4.5}$$

Filtering

In Fig. 4.1 the filtering has been mentioned as a device or process, which actually removes from a signal some unwanted component or feature.

The frequency response can be classified into a number of different forms, for example:

- Low pass filter low frequencies are passed, high frequencies are attenuated.
- High pass filter high frequencies are passed, low frequencies are attenuated.
- Band pass filter only frequencies in a frequency band are passed.

- Band stop filter only frequencies in a frequency band are attenuated.

Linear Time Invariant System

Before starting to look at the filter, some concepts must be introduced first.

Linear vs. Nonlinear *A **linear system** is any system that obeys the properties of scaling (homogeneity) and superposition (additivity), while **a nonlinear system** is any system that does not obey at least one of these.* and can be described so in mathematics:

$$H\left(k_1 f_1\left(t\right) + k_1 f_1\left(t\right)\right) = k_1 H\left(f_1\left(t\right)\right) + k_2 H\left(f_2\left(t\right)\right). \tag{4.6}$$

Time Invariant ***Time invariant** system is one that does not depend on when it occurs: the shape of the output does not change with a delay of the input.* That is to say that for a system H where $H(f(t)) = y(t)$, H is time invariant if for all T

$$H\left(f\left(t-T\right)\right) = y\left(t-T\right). \tag{4.7}$$

When a system has both linear and time invariant characteristics, we call it a linear time invariant (LTI) system. and the Fig. 4.4 illustrates the superposition in Linear Time-Invariant Systems.

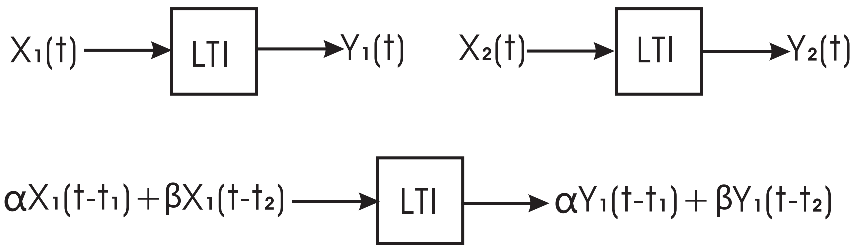


Figure 4.4: The principle of superposition applied to LTI systems

Impulse Response and Convolution

LTI systems are characterized uniquely by their impulse response sequence $h(n)$, which is defined as the response of the system to a unit impulse $\delta(n)$,as shown in Fig. 4.5 [21]. The unit impulse is the discrete-time analog of the Dirac delta-function $\delta(t)$ and is defined as

$$\delta\left(n\right)=\begin{cases}1 & n=0\\ 0 & n\neq 0\end{cases}\tag{4.8}$$

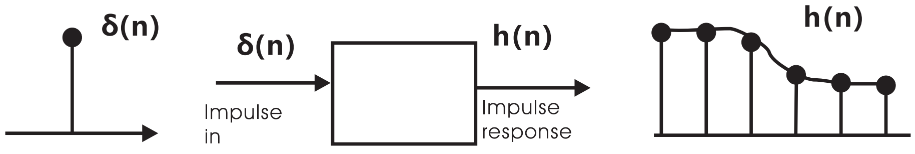


Figure 4.5: Response to linear combination of inputs

Time invariance implies that

$$\delta\left(n-D\right)\xrightarrow{H}h\left(n-D\right)\tag{4.9}$$

and linearity can prove that

$$\delta\left(n\right)+\delta\left(n-1\right)+\delta\left(n-2\right)\xrightarrow{H}h\left(n\right)+h\left(n-1\right)+h\left(n-2\right).\tag{4.10}$$

The weighted linear combination of three inputs will cause the same weighted combination of three outputs:

$$x\left(0\right)\delta\left(n\right)+x\left(1\right)\delta\left(n-1\right)+x\left(2\right)\delta\left(n-2\right))\xrightarrow{H}h\left(0\right)\delta\left(n\right)+h\left(1\right)\delta\left(n-1\right)+h\left(2\right)\delta\left(n-2\right).\tag{4.11}$$

An arbitrary input sequence $x(0)$, $x(1)$, $x(2)$, ... can be usually described as the linear combination of shifted and weighted unit impulses:

$$x\left(n\right)=x\left(0\right)\delta\left(n\right)+x\left(1\right)\delta\left(n-1\right)+x\left(2\right)\delta\left(n-2\right)...\tag{4.12}$$

and the output sequence will be obtained by replacing each delayed unit impulse by the corresponding delayed impulse response, that is:

$$y\left(n\right)=x\left(0\right)h\left(n\right)+x\left(1\right)h\left(n-1\right)+x\left(2\right)h\left(n-2\right)...\tag{4.13}$$

or written in this form:

$$y\left(n\right)=\sum_{m}x\left(n\right)h\left(n-k\right)\tag{4.14}$$

or

$$y\left(n\right)=\sum_{m}h\left(n\right)x\left(n-k\right). \tag{4.15}$$

This is the ***discrete-time convolution*** of the input sequence $x(n)$ with the filter sequence $h(n)$. Hence, LTI systems are convolves.

FIR and IIR Systems

Both FIR and IIR filters are LTI systems and are defined as:

Finite Impulse Response (FIR) *have an impulse response that is zero outside some finite time interval* or

$$h\left(n\right)=0 \qquad for\; n<0\; and\; n\geq M \tag{4.16}$$

M is referred to as the filter order. The output at any time n is simply a weighted linear combination of the input signal samples $x[n]$ to $x[n-M+1]$, where the weights are the values of the impulse response $h[k]$ for $k=0$ to $M-1$.

The system acts as a window that views only the most recent M input signal samples in forming the output [16]. The Fig. 4.6 explains an FIR, which has a finite memory of length 4 samples. z^{-1} represents the digital memory elements. So it's not difficult to find out, that in this FIR example,

$$y\left(n\right)=\sum_{k=0}^{4}h_k x\left(n-k\right). \tag{4.17}$$

Consider that we change the numb er 4 to ∞ . It means this system has infinite memory, which doesn't exist in the real world, how could an infinite impulse response (IIR) system implement? In some cases it is better to define the output of a system in terms of its current input value and past output values instead of just the resent and past values of its inputs[16]. A general form of the IIR system is defined by,

$$y\left(n\right)=-\sum_{k=1}^{N}a_k y\left(n-k\right)+\sum_{k=0}^{M}b_k y\left(n-k\right). \tag{4.18}$$

a_k and b_k represent a time-invariant constant.

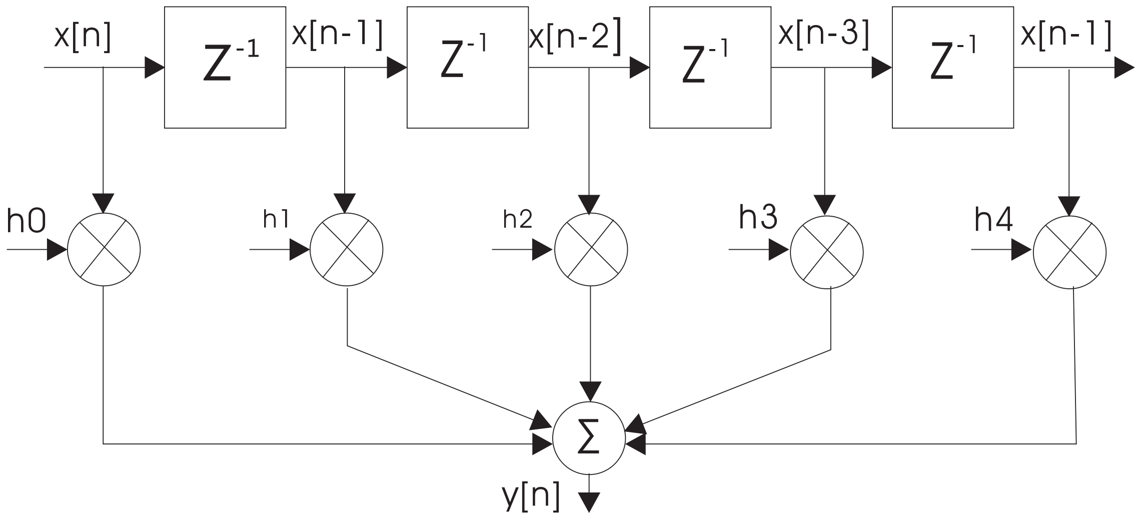


Figure 4.6: example of FIR

4.2.3 FIR Filter Design for a Bandpass Filter

We discuss the basic principles of FIR filter design and concentrate mostly on bandpass filters, which is the core of the algorithms for the experiment later. There are two main reasons why FIR filters are very attractive for design:

- Stable
- Easy to attain linear phase

Window Method

Ideal Filter The window method is one of the simplest methods of designing FIR digital filters. An ideal bandpass filter shapes can be designed is shown in Fig. 4.7. It blocks signals with a frequency range of $-\pi$ to $-\omega_b$, between $-\omega_a$ and ω_a and from above $-\omega_b$. The signals operating at frequencies between $-\omega_b$ and $-\omega_a$, also ω_a and ω_b.

A given desired ideal frequency response, say $D(\omega)$, being periodic in (ω) with period $2(\pi)$, need only be specified over one complete Nyquist interval $-\pi \leq \omega \leq \pi$. The corresponding impulse response, say $d(k)$, is related to $D(\omega)$ by the Discrete-Time Fourier Trans-form (DTFT) and inverse DTFT relationships, which will not be told into details and just taken this formula:

$$D\left(\omega\right) = \sum_{k=-\infty}^{\infty} d\left(k\right) e^{-j\omega k} \Leftrightarrow d\left(k\right) = \int_{-\pi}^{\pi} D\left(\omega\right) e^{j\omega k} \frac{d\omega}{2\pi} \tag{4.19}$$

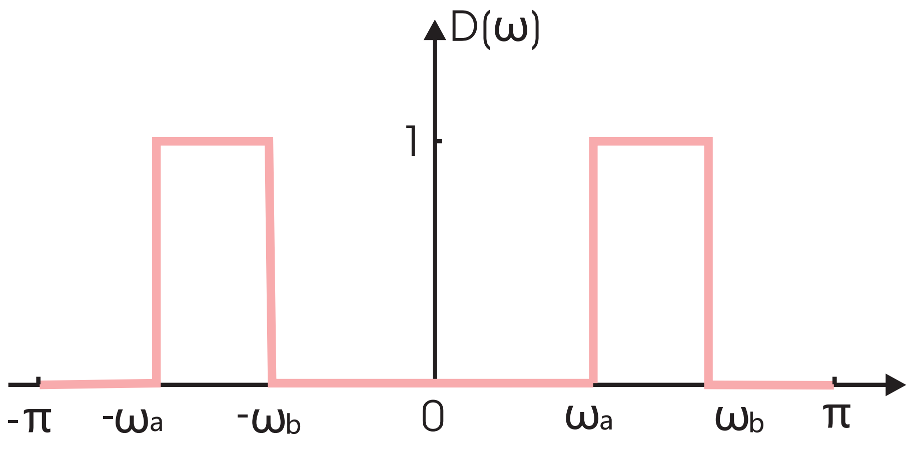


Figure 4.7: Ideal bandpass filter

and frequency response $D(\omega)$ in this example satisfies

$$D\left(\omega\right)=\begin{cases}1, & if\ -\omega_b<\omega<-\omega_a,\ or\ \omega_a<\omega<\omega_b;\\ 0, & if\ -\pi<\omega<-\omega_b,\ or\ -\omega_a<\omega<\omega_a,\ or\ \omega_b<\omega<\pi.\end{cases} \tag{4.20}$$

Hence, equation 4.18 can be written as the impulse response of the ideal band pass filter:

$$d\left(k\right)=\frac{\sin(k\omega_b)-\sin(k\omega_a)}{k\pi}\quad -\infty<k<\infty \tag{4.21}$$

Window Functions Because the impulse response required to implement the ideal bandpass filter is infinitely long, it is impossible to design an ideal FIR bandpass filter. Although $d(k)$ decreases with large k it is in theory infinite. Therefore this is not a FIR. One Solution to this problem is to truncate the impulse response for $n > M$. However finite length approximations to the ideal impulse response introduce undesirable ripples and overshoots(the Gibbs phenomenon) in the frequency response of the filter - ***The Windowing Method*** . A practical approach is to multiply the ideal impulse response by a suitable window function $\omega[n]$ whose duration is finite. This will ensure that the resulting impulse response decays smoothly to zero.

FIR Filter Design Specifications Both the passband/stopband ripples and the transition width are undesirable but unavoidable, which disobey the response of an ideal bandpass filter. Practical FIR designs typically consist of filters that meet certain design specifications,which describes a transition width and maximum passband/stopband ripples that do not exceed allowable values or a filter order M. Fig. 4.8 illustrates a typical deviations from the ideal bandpass filter when approximating with an FIR filter.

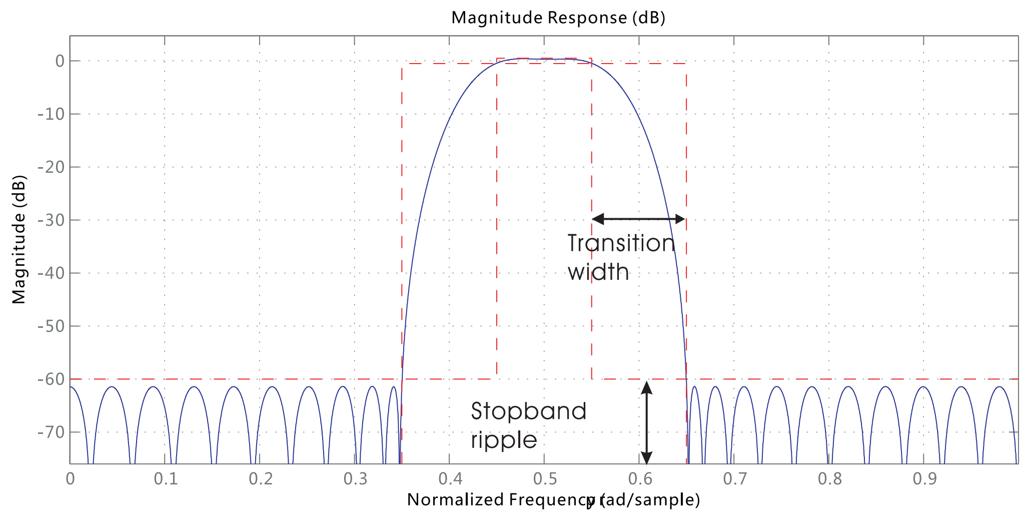


Figure 4.8: The typical deviations from the ideal bandpass filter

4.3 Model Construction for Simulation

4.3.1 Basic concept

As what mentioned in section 4.1 the purpose of this incidence is seeking for each signature sound of the two resonators and the exact occurrence time of them. Since the resonators have totally different frequencies, the first thing supposed to do is using a bandpass filter, which can pass only one of the both resonators' frequencies. Two optimal frequency range can be observed, in which only one of the structure borne sound exists. So it is necessary to design a bandpass filter at first.

The system's resonant frequencies are known as the frequencies at which the response amplitude is a relative maximum. so the next thing is going to find out the maximum value of the signals' amplitude.

Finally on the basis of the two maximum values the occurrence time of the signature sound should figure out.

4.3.2 Design of the Bandpass Filter

The first step begins with the bandpass filter design.

Preprocess the Audio Data

The format of the sampled audio data from the injection molding machine is .mat file, which can be loaded by MATLAB into workspace and its value is read as 1200000×4 double. Only the first channel of the data is needed, so the command :

plot(data(:,1))

helps to express graphicly like Fig.4.9.

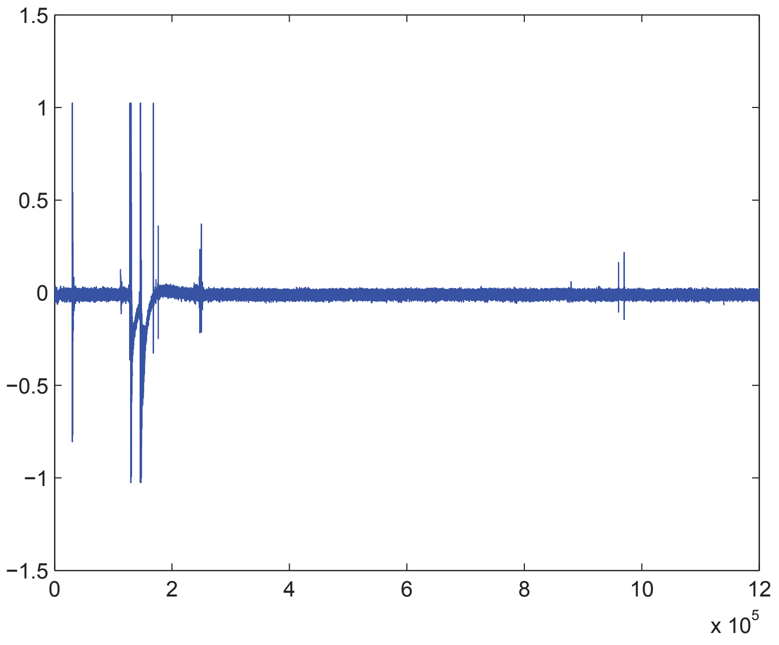


Figure 4.9: The original audio data channel 1

The signature sound happens in the data range from 2×10^5 to 3×10^5. So the command

data_new = data(200000:300000,1);

is used to cut out the required data. After that the data looks like the figure below:

The standard audio sampling rate used by professional digital video equipment is 48kHz, which was chosen because the human ear can respond to minute pressure variations in the air if they are in the audible frequency range, roughly 20-22kHz and according to the Nyquist sampling theorem the sample frequency of the audio device must be more than twice. The audio sampling rate of Beagle Board is from 4000 to 48000Hz, which means the 120kHz is not acceptable. This problem can be solved with the MATLAB function **Resample**, which resamples object data at P/Q times original sample rate. A new .m file is generated for that aim, which include only two command:

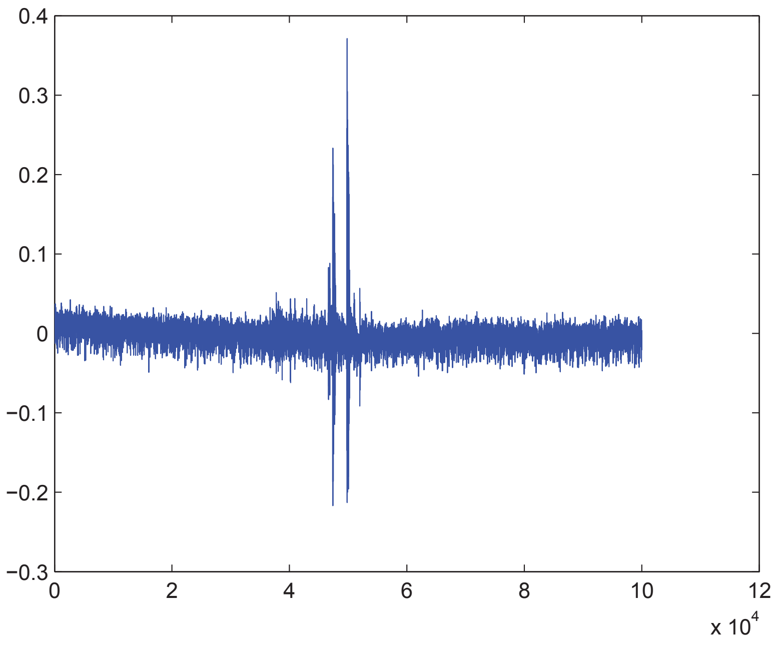


Figure 4.10: The required audio data which contains the structure borne sound

load data_new;

data_new_resample=resample(data_new,2,5);

Assumed that the data_new has been saved under the current folder of MATLAB. In this way the audio data with sample rate 48kHz is produced. In the workspace the data_new_resample can be seen as 40001×1 double. For simulation purpose this data must be written into sound file with the MATLAB command wavwrite :

wavwrite(data_new_resample,48000,'Schuss14_resample');

Now a .wav file with sampling rate of 48kHz is generated, which can be taken to do the audio signal process.

Parameters Tuning

The bandpass filter is planed to be the kernel algorithms for distinguishing those two signature elements. In the previous section the mathematically design of the filter design is described. It's not easy work to program or write this into MATLAB language, still there is one in the appendix. Simulink provides a fairly simple approach to design one's own filter: ***The DSP System Toolbox*** in the Library Browser. In the submenu exist the Filtering, in that a Bandpass Filter block can be found out. With this block the identification of one sound is able to be done. In the DSP Toolbox there is a 'From Multimedia File' block, which allowed wav file to be loaded as signal inputs. In order to observe the outputs which comes after the signals processed by the bandpass filter, a time scope is connected to the filter.

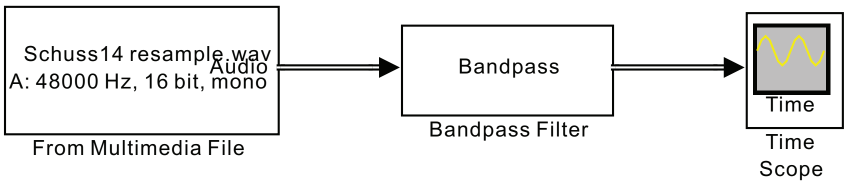
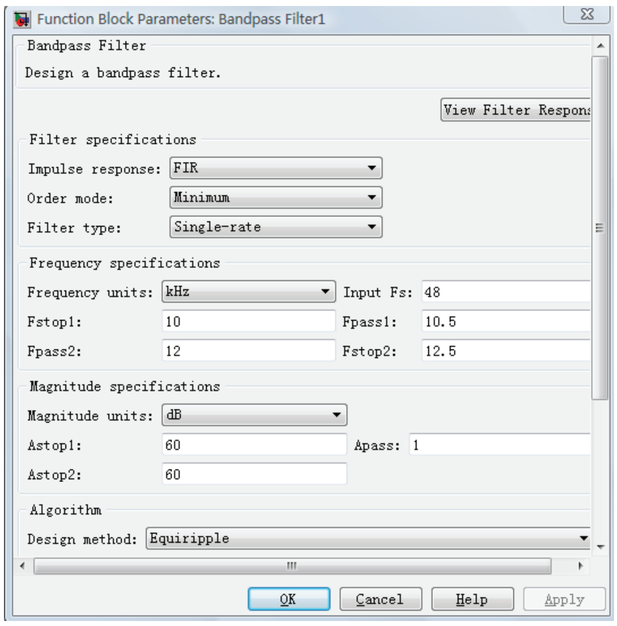


Figure 4.11: The bandpass filter model, which is designed to identify the signature sound

After structuring the model various kinds of frequencies should be tested to obtained the optimal two, under each of that 2 frequency ranges exists only one structure borne sound or at least one sound is obviously more characteristic than the other one. For purpose of get such frequency range the parameters of the bandpass filter blocks must be set up like Fig. 4.12 and one clicks 'View Filter Response' the graphical details of the filter can be displayed.


Function Block Parameters: Bandpass Filter1

Bandpass Filter

Design a bandpass filter.

View Filter Respons

Filter specifications

Impulse response: FIR

Order mode: Minimum

Filter type: Single-rate

Frequency specifications

Frequency units: kHz   Input Fs: 48

Fstop1: 10   Fpass1: 10.5

Fpass2: 12   Fstop2: 12.5

Magnitude specifications

Magnitude units: dB

Astop1: 60   Apass: 1

Astop2: 60

Algorithm

Design method: Equiripple

Figure 4.12: The parameter set-up of the bandpassfilter

Different frequency ranges of the pass band have been tested. One can see that in the range about 22kHz just exists the second signature element, the first one disappears (Fig. 4.13).

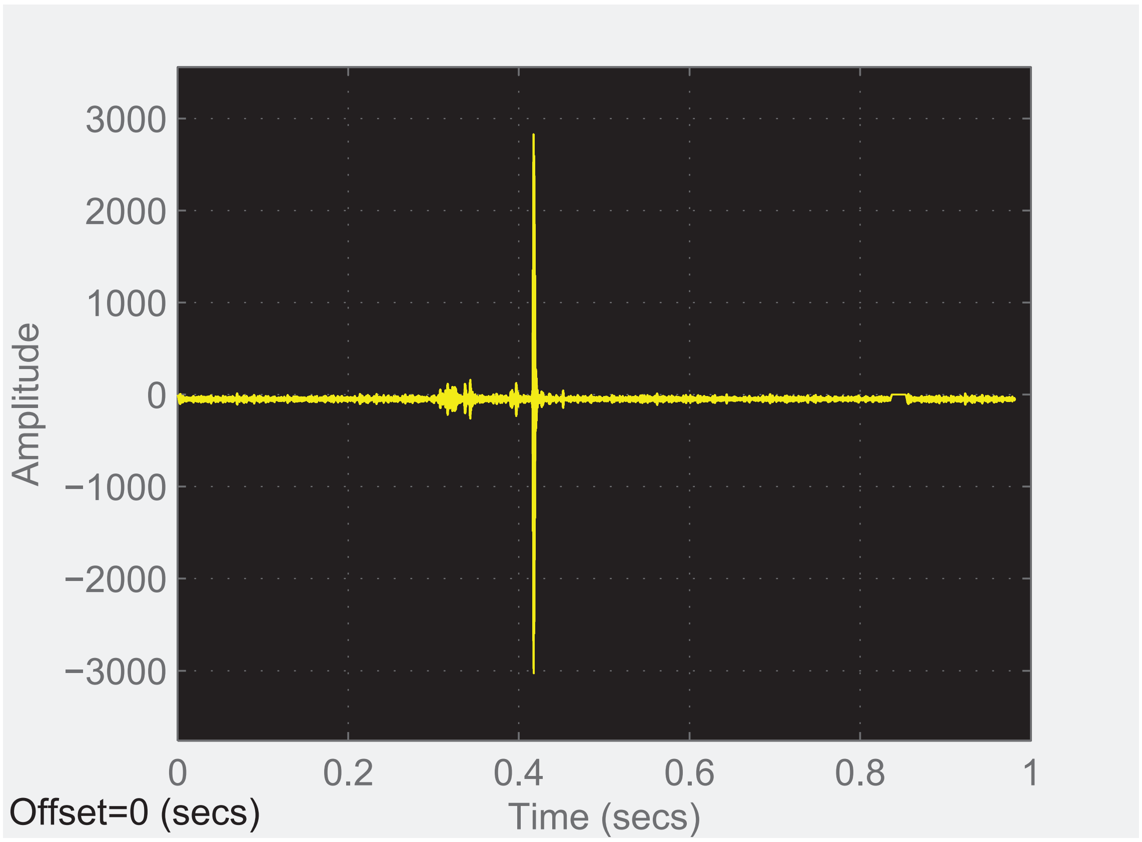


Figure 4.13: The signals after processing by the bandpass filter of pass band about 22kHz

The other result is not just as desired, in the test frequency range from 5000Hz to 25kHz the other optimal pass band for the first signature element can't be found, which means the second exists in the entire range. So the other way to identify the first one must be tried, that in some range the first is clearly different from the second sound. In the zone of 11kHz this idea works. The first one has a much larger magnitude as the second one(Fig. 4.14).

Thus far,the design for the filter part is finished.

4.3.3 Peak detector with the MATLAB-Function

For the sake of finding the maximal value of the filtered signal inputs, peak detectors are in need. Simulink provides some blocks of that functions such as peak finder, which displays all the local maximum of the inputs signals or MinMax block with the same purpose. But they both are not what is searched for. The idea is to obtain not only the maximum but also the response to the change of the local maximum values. So a new block in Simulink should be written for that task. Here we use the MATLAB function under user-Defined Functions submenu, so that the capability of using the blocks on the target hardware can

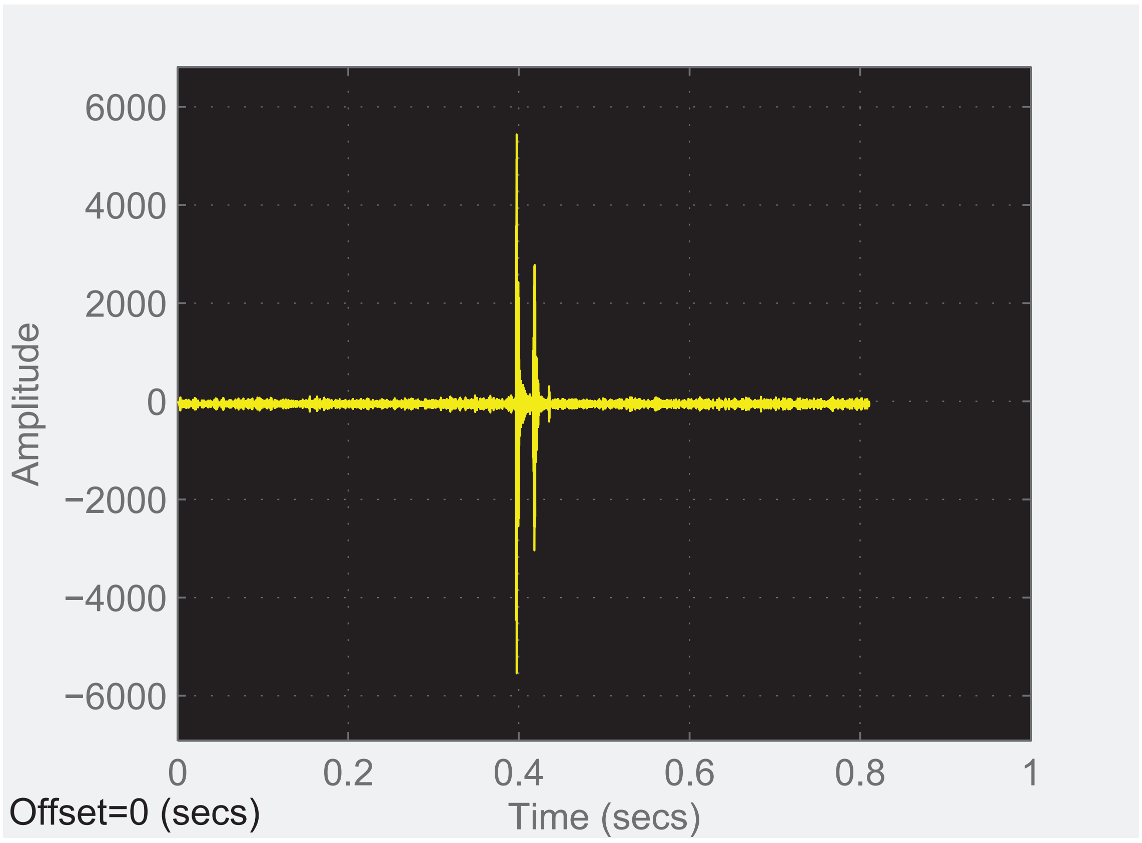


Figure 4.14: The signals after processing by the bandpass filter of pass band about 11kHz

also be tested.

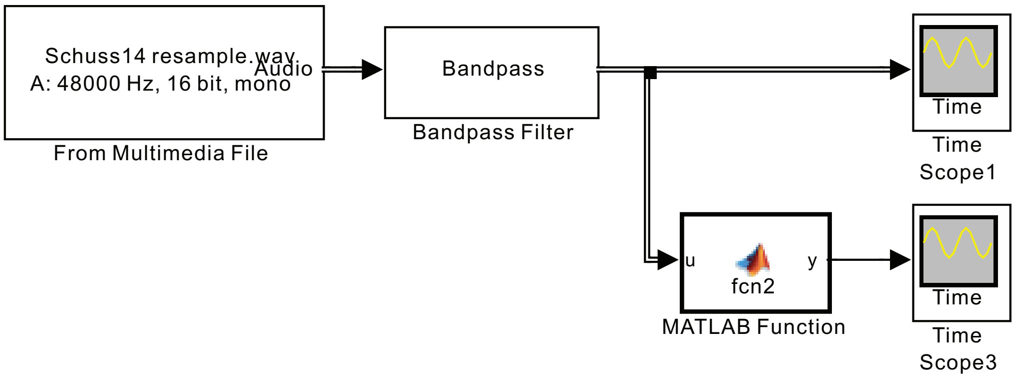


Figure 4.15: Part of the model with peak detector using MATLAB-function.

Time constant τ

The detector time constant (τ) represents the time it takes for the detector output to reach a level τ of an instantaneous change in the incoming signal, which usually is (1-1/e) = 63.2%. In order to get the optimal behavior of the local maximal values, various values for τ were tried. For 128 samples per audio channel the range of τ is acceptable from 0.0001 to 0.01. In the block the peak detector is programmed, whose details can be found in appendix. By using this block a satisfactory result is worked out.

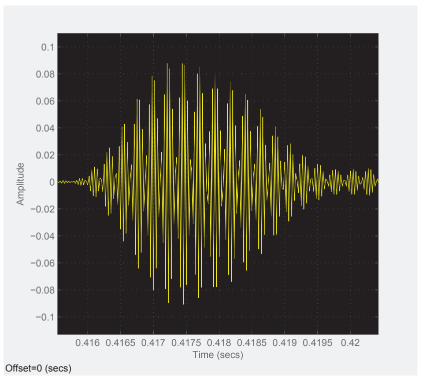


Figure 4.16: The filtered signals which contains the maximal amplitude of the signature sound

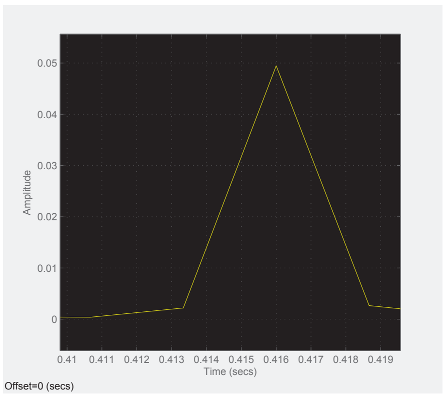


Figure 4.17: The maximal amplitude of the signature sound has been represented with the Time Scope3

Absolute Maximum

The complete result coming from the peak detector is shown in Fig. 4.18, where some local maxima can still be seen. So the next step is to achieve the absolute one and detect its position.

The solution is comparing with a threshold, so that only the values above it will be held. In Fig 4.18 it is obvious that all the other peaks are much smaller than 0.01 except the absolute one. To be sure about this observed result, a MaxMin block is used with the mode parameter set to 'running', which means the absolute maximal value holds. It has also an advantage by executing the model on the target PC, which will be discussed in next section. In this way, the threshold can be found out. Then the 'Relational Operator' will be connect to the peaks detector and the threshold. Finally the absolute maximum with its position shows in Fig. 4.19. In the same manner, the signals from the bandpass filter with pass band

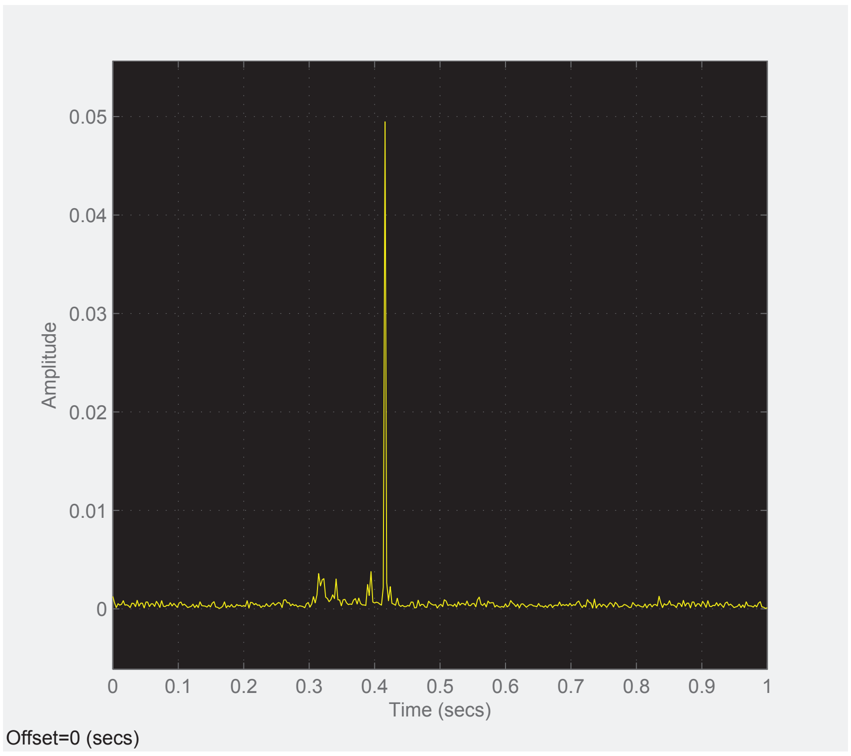


Figure 4.18: The complete result coming from the peaks detector.

of 11 kHz are processed (the result is displayed in Fig. 4.20) and the model of stage 2 is accomplished (Fig. 4.21).

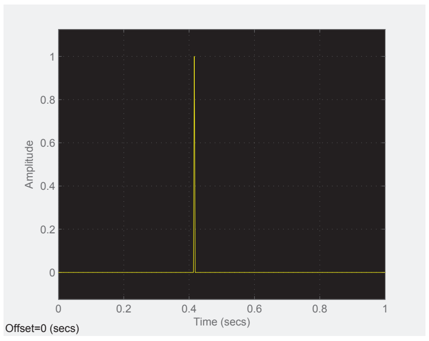


Figure 4.19: The absolute maximum with its position of the second signature element

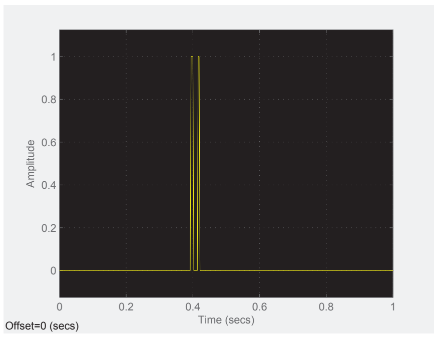


Figure 4.20: The absolute maximum with their positions of the both signature element

4.3.4 The Occurrence Time Finder Based on State Machine

The last step of the model design look for the occurrence time of the two structured borne sound. In stage 2 the positions of them have already appeared. The idea is take advantage of the positions to confirm their unique corresponding time. In this case the Stateflow block set of Simulink is going to be applied.

Stateflow

The Stateflow is a powerful graphical design and development tool based on state machines and flow for complex control and logic problems charts [31].

A finite state machine is a model, which describes the behavior of a finite number of states, the transitions between those states, and actions. A state represents an operating mode of a machine [11]. The activity or inactivity of the states dynamically changes based on on events and conditions. An action describes the activity that is to be performed. Like Simulink, objects can be dragged and dropped to create state transition charts in which a series of transitions directs a flow of logic from one state to another [10].

In this case there are two basic states labeled 'executing the time detector' and 'time counter'. During the activity of the first state the detector implements to find the appearance of the two signature elements. By the transition the second state will be activated to record the time of the result of the searching. The condition on the transition is showing up the position of the held maximum which explained in last subsection. One more point is the Default transitions, which specify the exclusive state is to be active when there is ambiguity between two or more exclusive states at the same level in the hierarchy [11]. so the chart of the Stateflow is obtained in Fig. 4.22.

In the chart the whole process can be understood as a visual logical work flow. The first action in 'detecting' state is a during action, that is executed while this state is active and no transition to another state occurs. The variable *eltime*, which represents the time of the first signature element, is updated with the actual time *tim*, as long as the state is active. When the state is left, el1time holds the time of the event. In the same manner the event time for the second acoustic element *el2time* is obtained (Fig. 4.22).

Next the input and output events should be added in Simulink and the data type must be setup under Tools Explores, like illustrated in Fig. 4.23.

The events sig11 and sig22 come from the 'Relational Operators' in Fig 4.22 and a clock must be put into the model for calculating the time. The blocks 'Clock' and 'Digital Clock' have been tested and both didn't work. So a 'Discrete-Time Integrator' is used to take place of the clock. The output is the time for the appearance of both the signature signals and the time difference is also needed, which will be discussed in next section. After connection of the inputs and 'Display' blocks as outputs to the Stateflow chart the entire model for the project's simulation is finished (Fig. 4.24).

4.3.5 Result from the Simulation

As what displayed after running the model, the first structure borne sound occurs at 392ms and the second appears at 418.7ms.

4.4 Implement the Model on the Target Beagle Board

The results from simulation is as the wishes, so its algorithms can be accepted and inherited by the Beagle Board.

4.4.1 Model for the Target Hardware

For the Beagle Board the input audio signals is no longer the wave file but the audio from the sound card using its ALSA driver framework. By using the 'ALSA Audio Capture' block the signal source can be accessed. This kind of source is stereo, which means the output's dimensions are $[N \times 2]$. The audio sampling frequency is also set to 48kHz as in the simulation. The wave file signals are mono, so one block must be added extra to select only one of the columns of the outputs for processing by using the 'Multiport Selector' block. First the model should be built almost like in the Simulation except the input signal part, the selector block and the 'ALSA Audio Playback' instead of 'To Audio Device' block. Since the scale is different from the simulation running on the host PC, the threshold value can not be set to the same value. On the other hand time Scopes cause the running error, the MaxiMin block is used to estimate the threshold as mentioned before. After running the model once, the threshold value will be obtained. Then the comparator level for getting the time incident is modified and the model started again. The program (Fig. 4.25) for the board looks similar to the model for simulation.

4.4.2 Run the Model on Beagle Board

To run this acoustic machine condition monitoring program the following hardware is required:

- Headphones or speakers,
- audio input source and
- audio cable.

The connection is the same as what described in section 2.4. Additionally the speaker must be plugged into the audio in of the board and one side of the audio cable to the audio out of the target and the other side to the audio out of the host PC. Run the model under external mode and play the wave file on the host with the software like media player. Due to the time running at the very beginning before playing the wave file, the time for detecting the position of the signature elements can not be told, but the difference between then is showed as 91.38ms, which is a little bit different from the one from simulation (26.70ms). Different audio software is applied to play the wave file, but the results of the time-difference are identical. The result is shown in Fig. 4.26.

4.5 Conclusion

The development of a machine condition monitoring system based on acoustic signals points out a way to develop software for a real-time hardware platform with help of Simulink target support. First the program is implemented as a model on the PC for testing and simulation, which takes much less time than to run the model on the target system. Then only a few blocks are modified to adjust the development environment to the target hardware. An advantage is that many functions from Simulink and Matlab libraries can be used directly for the real-time application. For example from block-sets for digital signal processing, Stateflow, data acquisition and computer vision. The so-called external mode allows the access to process variables and makes them visible in the Matlab workspace. Communication blocks, such as UDP and TCP, or scope functions are available. Some of them, which are not especially designed for the target system, may cause problems. Consequently it is important to test such functions carefully before deployment.

Another aspect of the design process turned out during this work. Using this target support, the software development for embedded systems is possible requiring less knowledge about the target hardware details. Furthermore, engineers need not to be experts for languages and programming techniques, but can concentrate on algorithms and their specific processes.

Its also easy to tune the parameters and monitor the signals, which brings great convenience and saves time. The real-time implementation on the target hardware shows a satisfying result compared to the simulation of the host computer.

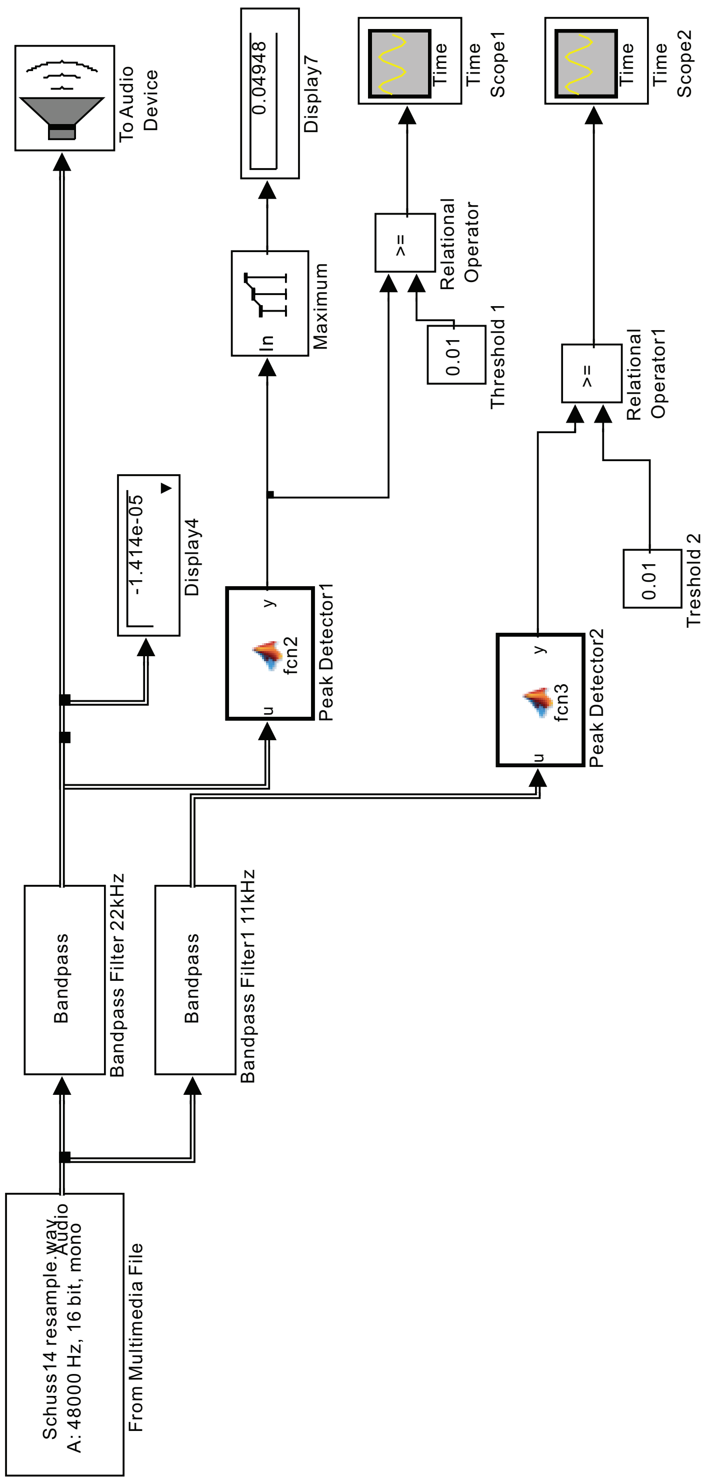


Figure 4.21: The complete result coming from the peaks detector.

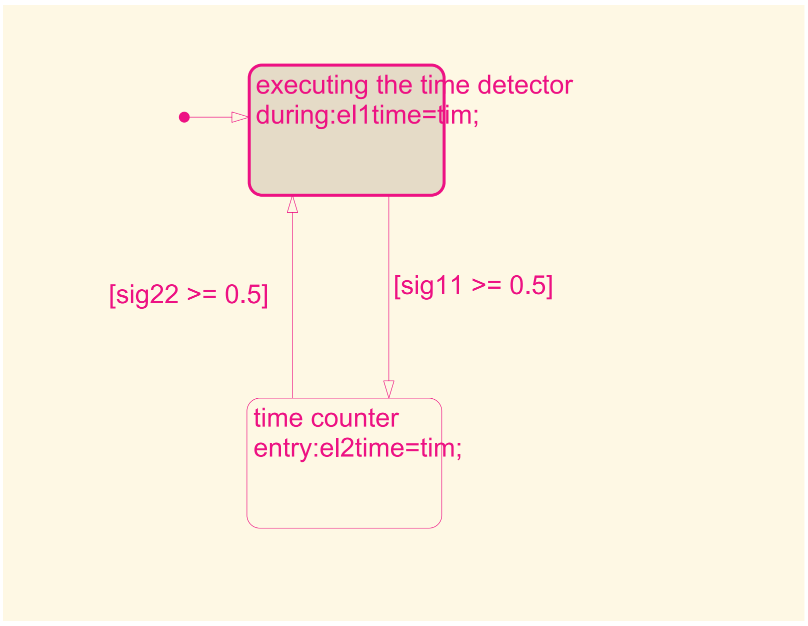
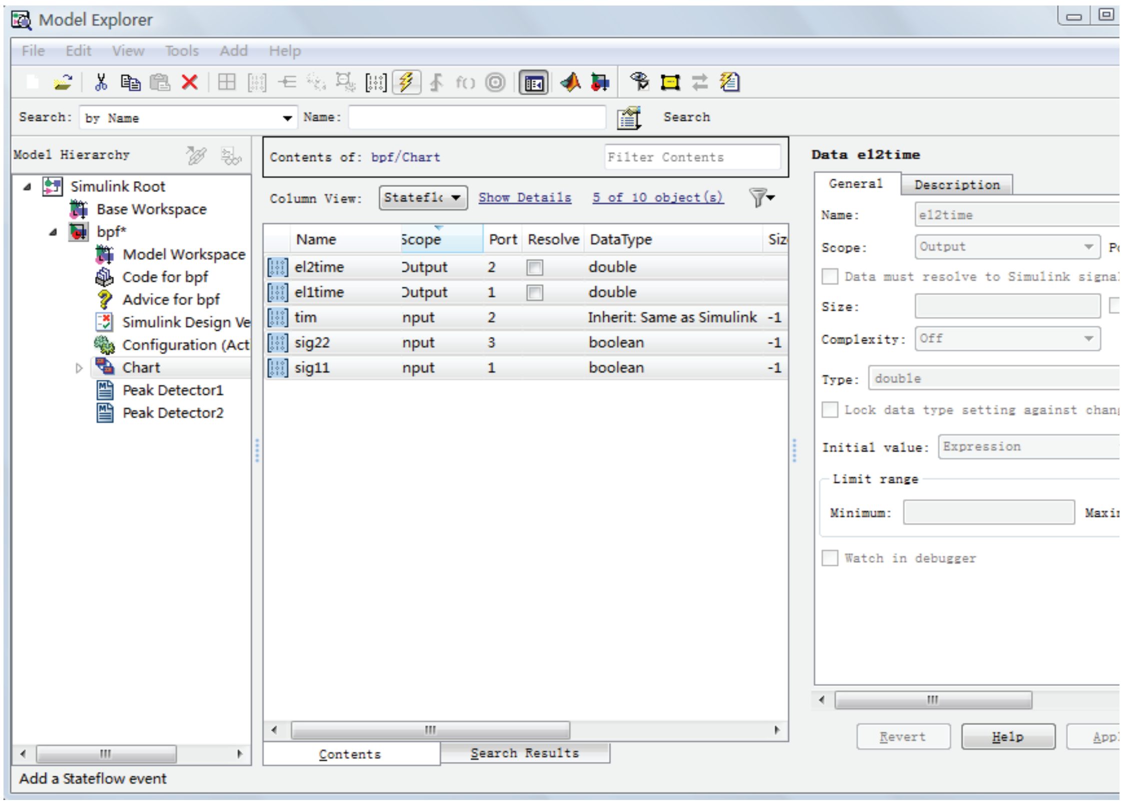


Figure 4.22: The Chart of the Stateflow.



Name	Scope	Port	Resolve	DataType	Siz
el2time	Output	2	☐	double	
el1time	Output	1	☐	double	
tim	nput	2		Inherit: Same as Simulink	-1
sig22	nput	3		boolean	-1
sig11	nput	1		boolean	-1

Name: el2time

Scope: Output

☐ Data must resolve to Simulink signal

Size:

Complexity: Off

Type: double

☐ Lock data type setting against chan

Initial value: Expression

Limit range

Minimum:

Maxi

☐ Watch in debugger

Figure 4.23: Input and output events for the stateflow.

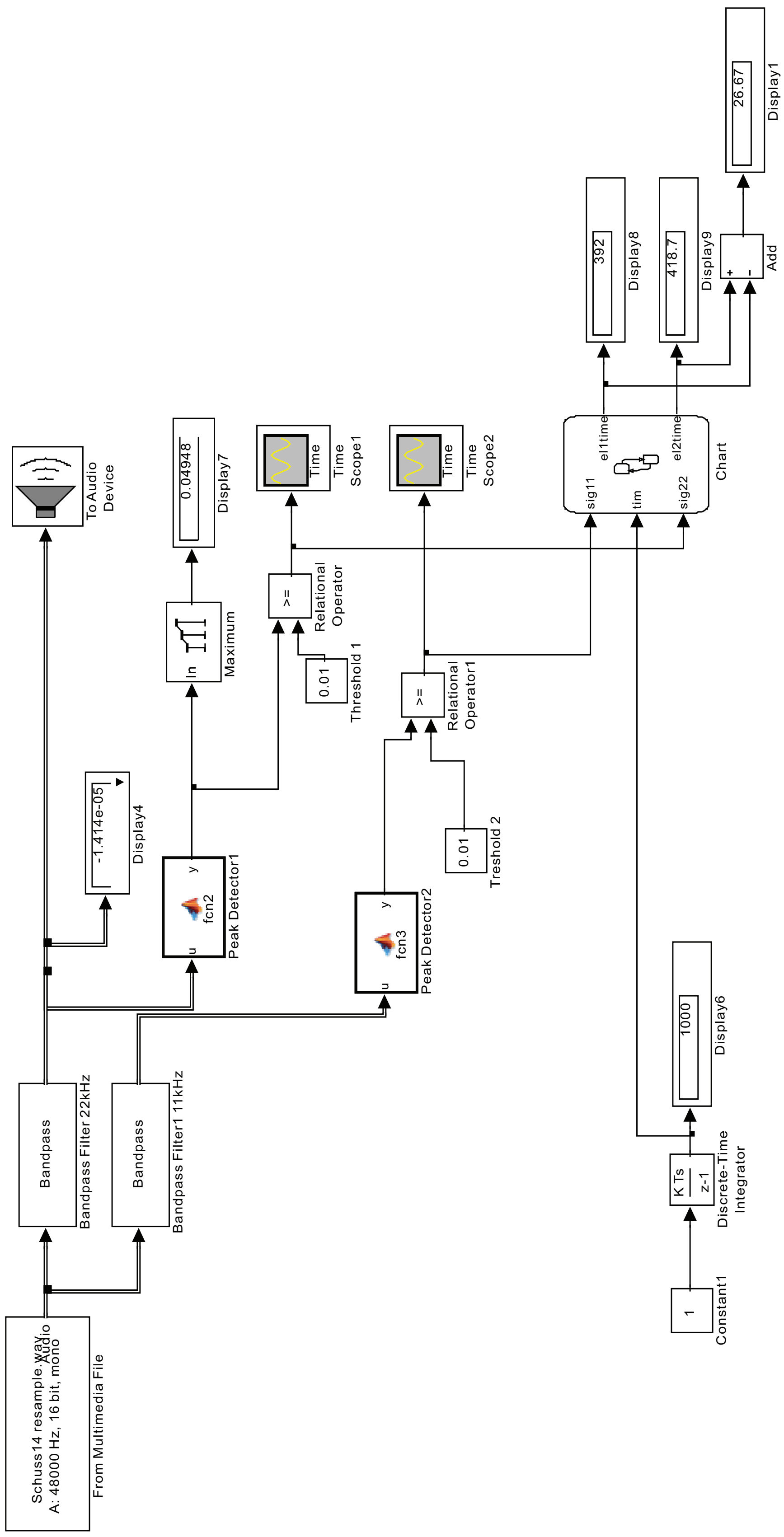


Figure 4.24: The entire model for the project’s simulation

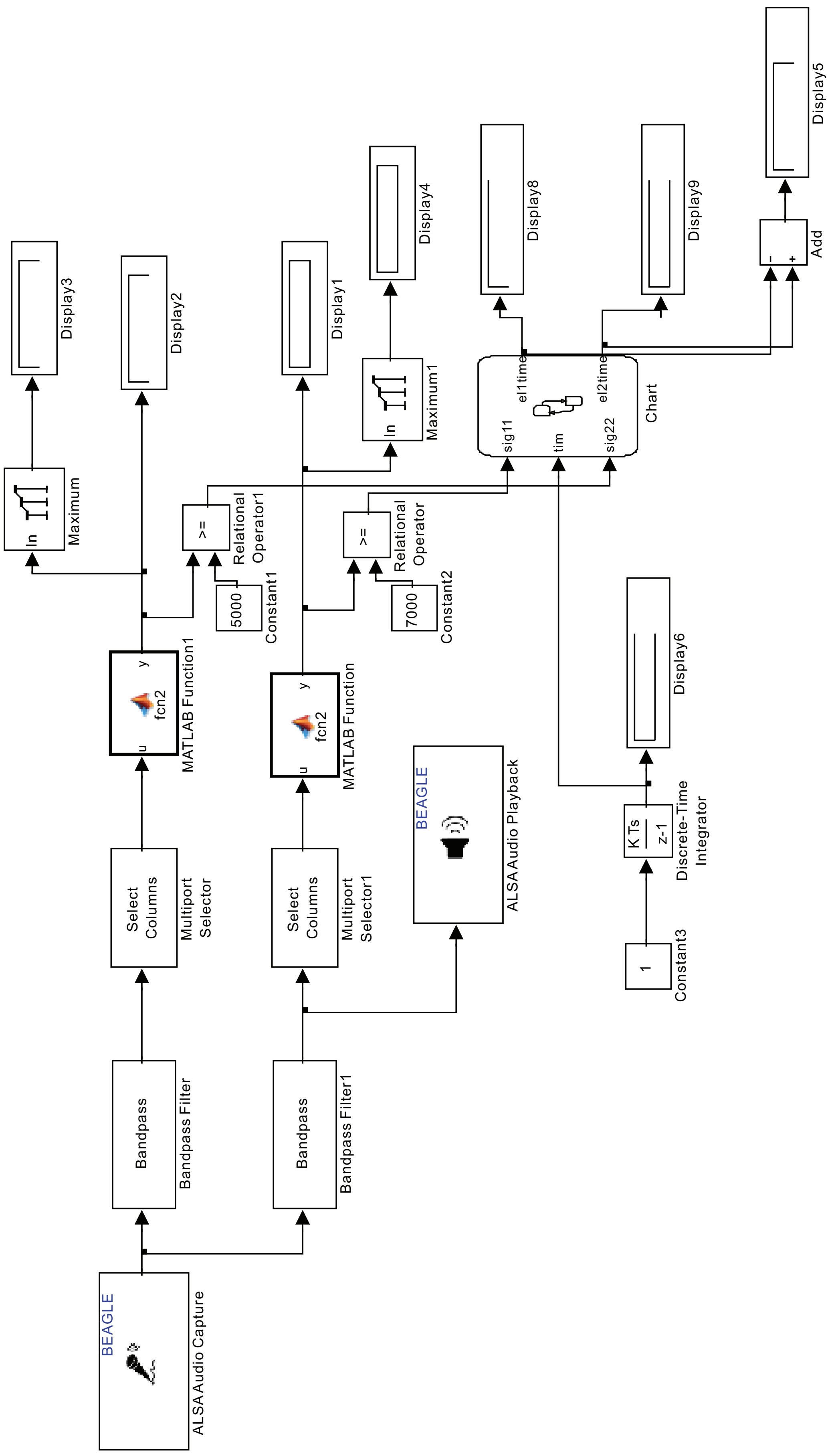


Figure 4.25: The entire model prepared for the project's implementation on the target Beagle Board.

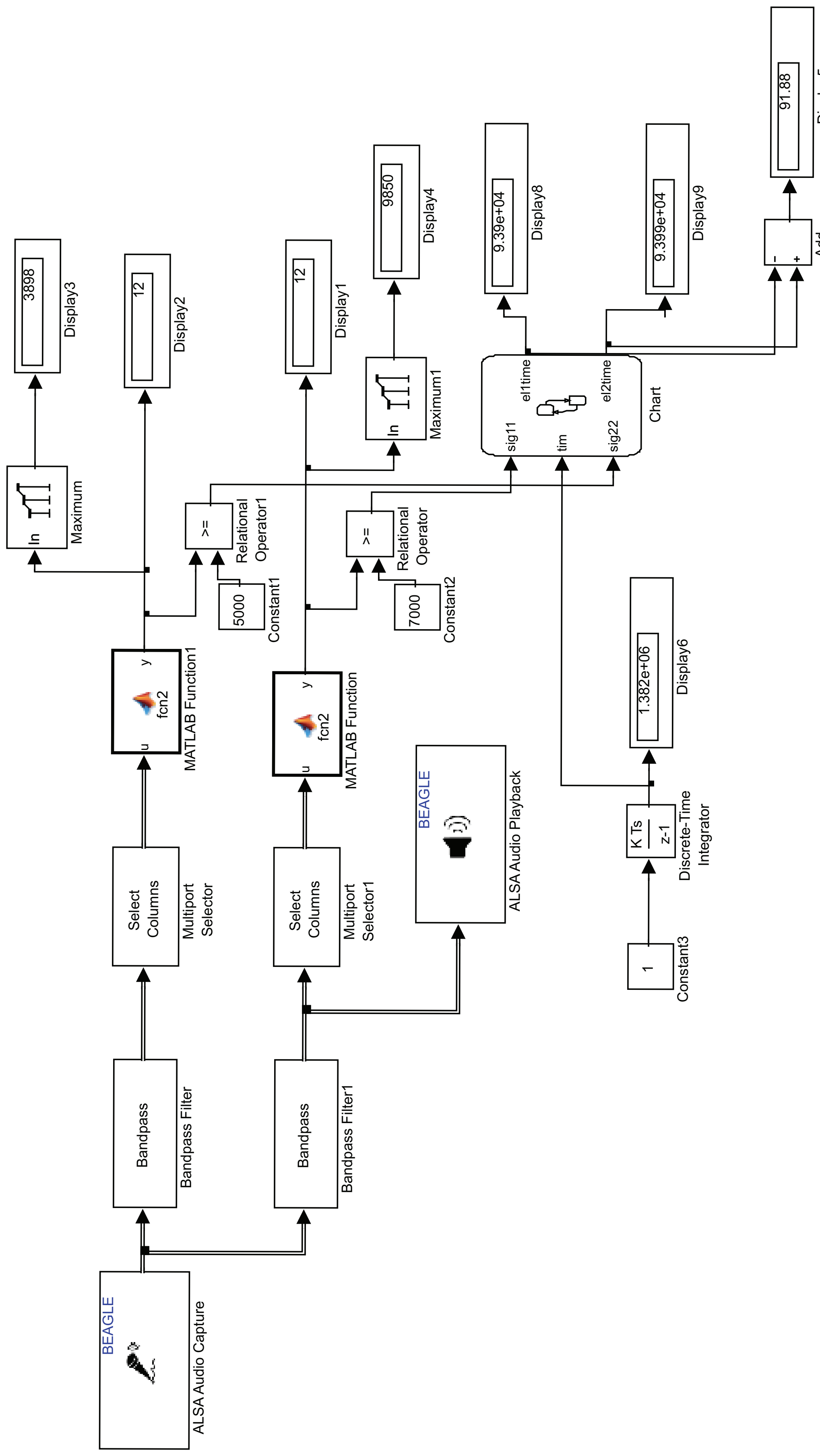


Figure 4.26: The result coming from the Beagle Board.

Chapter 5

Summary and Outlook

The goal of this thesis was to test the cross-development environment provided by Matlab®/ Simulink® for modern processors designed for embedded systems. Two different popular hardware platforms, which are supported, were used for this purpose, Arduino™ and BeagleBoard. Simulink supports model based design for software development on embedded real-time systems.

In Chapter 1 the basic concepts of real-time, embedded systems and model based design are introduced. A feature of the environment is that engineers can use the same suite for modelling and testing on the computer and for programming the target system. Consequently in the next chapter this new tool has been discussed in detail. It allows the designer to save time and focus on the algorithms and the rest will be all done with a click by support of Simulink.

In the third chapter a small design example was presented on the Arduino target to operate a servo motor by a joystick. This example shows the possibility of real-time control of this hardware. Then a digital signal processing application was developed for an injection molding machine. In this project the software development process was described. First the idea of the algorithms was abstracted to a graphical model. With the help of learning some DSP knowledge this model is accomplished step by step and was simulated on the host PC to test its possibility and the correctness of the algorithms. By simulation there is also the advantage to tune the parameters and monitor the signals, so that the model can be used to prepare the implementation on the target. After this the model only needed slight modification based on the simulation to run on the Beagle Board as a target system. The result shows that the system successfully responds in real-time to acoustic signals. This means that such development methods are applicable for industrial automation.

The number of hardware platforms that are supported directly is presently increasing. So it is expected that in the future more and more embedded hardware which are applied in every field of industrial automation can be developed in an efficient way, saving time and letting the engineers concentrate on the model construction.

References

[1] BeagleBoard.org. Beagleboard-xm rev c system reference manual, 2010.

[2] J. Bélanger and C. Dufour. Modern methodology of electrical system design using rapid-control prototyping and hardware-in-the-loop. In K. Popovici and P.J. Mosterman, editors, *Real-time Simulation Technologies: Principles, Methodologies, and Applications*. CRC Press, Boca Raton, FL, 2012.

[3] M. Bilsky. Iterfacing beagleboard with simulink and arduino. `http://mattbilsky.com/mediawiki/index.php?title=Interfacing_BeagleBoard_with_Simulink_and_Arduino#Configuring_Matlab_2010b_Code_Generation`. accessed nov. 2012.

[4] T. Brühlmann. *Arduino Praxiseinstieg*. MITP, 2010.

[5] R. Colgren. *Basic MATLAB, Simulink, and Stateflow*. American Institute of Aeronautics and Astronautics, Inc., Reston, Virginia, USA, 2006.

[6] R. Dlugy-Hegwer. Is the linux kernel on beagleboard a ”real-time” kernelhttp://www.mathworks.com/matlabcentral/answers/46081-is-the-linux-kernel-on-beagleboard-a-real-time-kernel. `http://www.mathworks.com/matlabcentral/answers/42290`. accessed nov. 2012.

[7] J. Giboz, T. Copponnex, and P. Mele. Microinjection molding of thermoplastic polymers. In *Journal of Micromechanics and Microengineering*, Nr. 6, pages P. R96–R109, 2007.

[8] E. Green. Os on beagleboard. `http://bbfordummies.blogspot.co.at/2009/07/os-on-beagleboard.html`. accessed nov. 2012.

[9] J. O. Hamblen. Introduction to embedded systems using windows embedded ce. Technical report, School of Electrical and Computer Engineering, Georgia Institute of Technology, 2007.

[10] J. Hoffmann and U. Brunner. *Matlab und Tools Für die Simulation dynamischer Systeme*. Addison Wesley, München, Germany, 2002.

[11] S. T. Karris. *Introduction to Stateflow with Applications*. Orchard Publications, Freemont, CA, USA, 2007.

[12] D.O. Kazmer, S. Velusamy, S. Westerdale, S. Johnston, and R.X. Gao. A comparison of seven filling to packing switchover methods for injection molding. *Polymer Engineering and Science*, 50(10):2031–2043, 2010.

[13] H. Kopetz. *Real-Time Systems*. Springer, New York, USA, second edition, 2011.

[14] Principia labs. Arduino serial servo control. `http://principialabs.com/arduino-serial-servo-control/`. accessed nov. 2012.

[15] P. A. Laplante. *Real-Time Systems Design and Analysis*. IEEE Press, Piscataway, W, USA, third edition, 2004.

[16] P. Lee. An introduction to signals, systems and digital signal processing. University Lecture, 2012.

[17] P.J. Mosterman, S. Prabhu, A. Dowd, J. Glass, T. Erkkinen, J. Kluza, and R. Shenoy. Embedded real-time control via matlab, simulink, and xpc target. In *Handbook of Networked and Embedded Control Systems*, Control Engineering, pages 419–446. Springer, 2005.

[18] F. Mueller, P. O'Leary, G. Rath, M. Kukla, C. Harker, T. Lucyshyn, and C. Holzer. Resonant acoustic sensor system for the wireless monitoring of injection moulding. Sensornets 2013,accepted.

[19] N. Nissanke. *Realtime Systems*. Prientice Hall Europe, Hertfordshire, British, first edition, 1997.

[20] A. Oppenheim and R. Schafer. *Discrete-Time Signal Processing*. Prentice Hall, Englewood Ciffs, NJ, 1989.

[21] S. J. Orfanidis. *Introduction to Signal Processing*. Prentice Hall, 1995.

[22] J. M. Osier-Mixon. Boot linux on the beagle board. Technical report, Embedded Developer and Writer, MontaVista Software, Inc., 2009.

[23] P. M. Sagar. Embedded operating systems for real-time applications. Technical report, Electronic Systems Group, EE Dept, IIT Bombay, 2002.

[24] TinkerKit. T020010 sensor shield v.2. `http://tinkerkit.com/en/Modules/T020010`. accessed nov. 2012.

[25] wikipedia. Embedded system. `http://en.wikipedia.org/wiki/Embedded_system`. accessed nov. 2012.

[26] wikipedia. Model-based design. `http://en.wikipedia.org/wiki/Model-based_design`. accessed nov. 2012.

[27] The Math Works. Configure network connection with beagleboard hardware. `http://www.mathworks.cn/cn/help/simulink/ug/getting-the-beagleboard-ip-address.html`. accessed nov. 2012.

[28] The Math Works. Connect to serial port on beagleboard hardware. `http://www.mathworks.cn/cn/help/simulink/ug/connect-to-serial-port-on-beagleboard-hardware.html`.

[29] The Math Works. Mathworks announces built-in simulink support for arduino, beagleboard, and lego mindstorms nxt. `http://www.mathworks.cn/company/newsroom/MathWorks-Announces-Built-in-Simulink-Support-for-Arduino-BeagleBoard` `html`. accessed nov. 2012.

[30] The Math Works. Replace firmware on beagleboard hardware. `http://www.mathworks.cn/cn/help/simulink/ug/update-firmware-on-the-beagleboard-hardware.html`. accessed nov. 2012.

[31] The Math Works. Stateflow users guide, 2001.

List of Figures

Appendix A

Install Support for Arduino Hardware

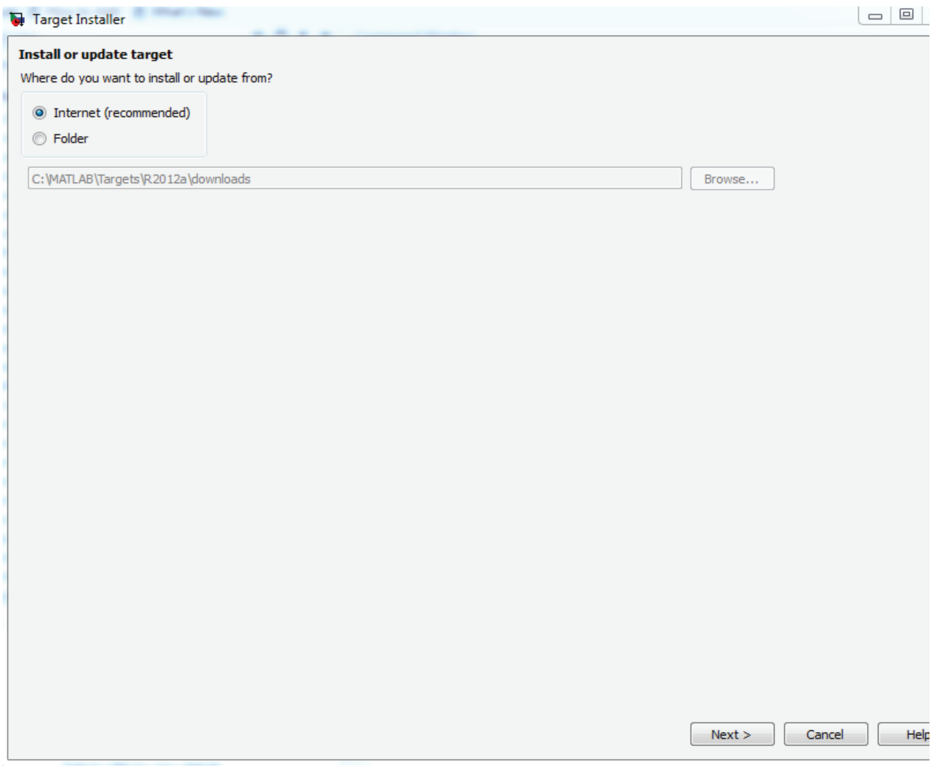
Step 1 Type targetinstaller in a MATLAB Command Window. or click the Tools menu > Run on Target Hardware > Install/Update Support Package.

Step 2 Select getting the support package from the Internet or from a folder,to that the package has been downloaded (Figure A.1).

Step 3 Select the Install check box for Arduino UNO target, and click Next (Figure A.2).

Step 4 Click Install and wait till the end of the progress (Figure A.3).

Step 5 Click Finish (Figure A.4).



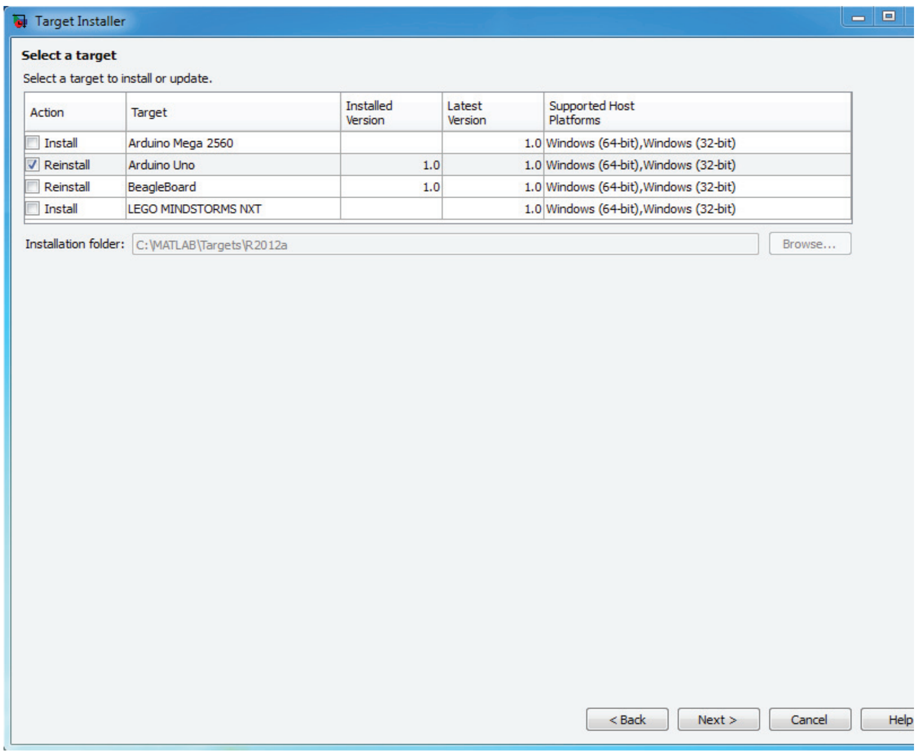
Install or update target

Where do you want to install or update from?

(● Internet (recommended)) ○ Folder

C:\MATLAB\Targets\R2012a\downloads Browse...

Figure A.1: Step 2



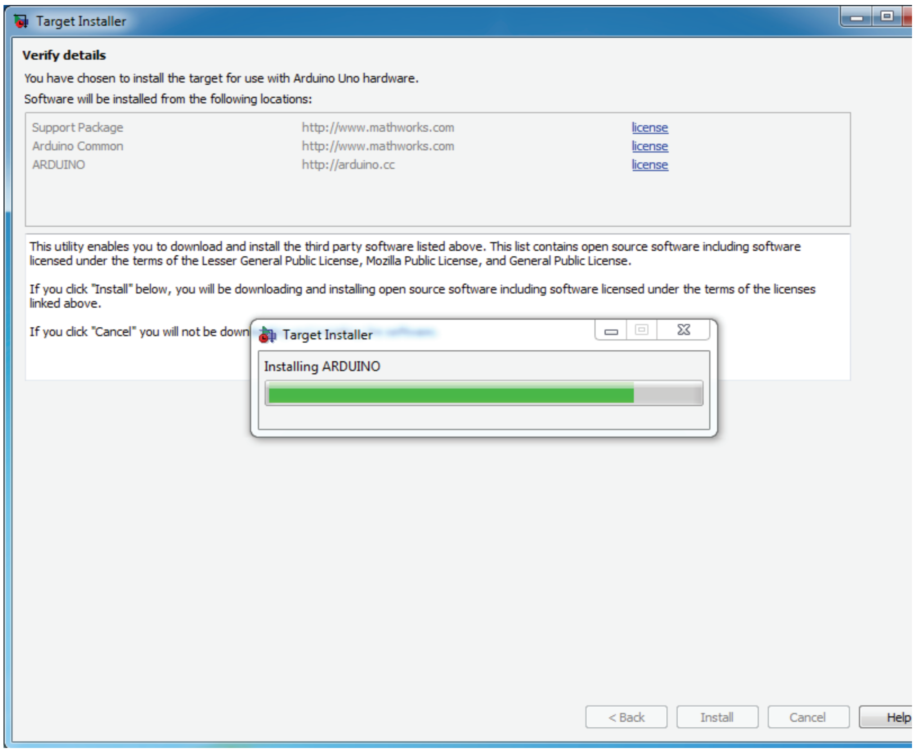
Select a target

Select a target to install or update.

Action	Target	Installed Version	Latest Version	Supported Host Platforms
☐ Install	Arduino Mega 2560		1.0	Windows (64-bit),Windows (32-bit)
✓ Reinstall	Arduino Uno	1.0	1.0	Windows (64-bit),Windows (32-bit)
☐ Reinstall	BeagleBoard	1.0	1.0	Windows (64-bit),Windows (32-bit)
☐ Install	LEGO MINDSTORMS NXT		1.0	Windows (64-bit),Windows (32-bit)

Installation folder: C:\MATLAB\Targets\R2012a Browse...

Figure A.2: Step 3


Target Installer

Verify details

You have chosen to install the target for use with Arduino Uno hardware.

Software will be installed from the following locations:

Support Package	http://www.mathworks.com	license
Arduino Common	http://www.mathworks.com	license
ARDUINO	http://arduino.cc	license

This utility enables you to download and install the third party software listed above. This list contains open source software including software licensed under the terms of the Lesser General Public License, Mozilla Public License, and General Public License.

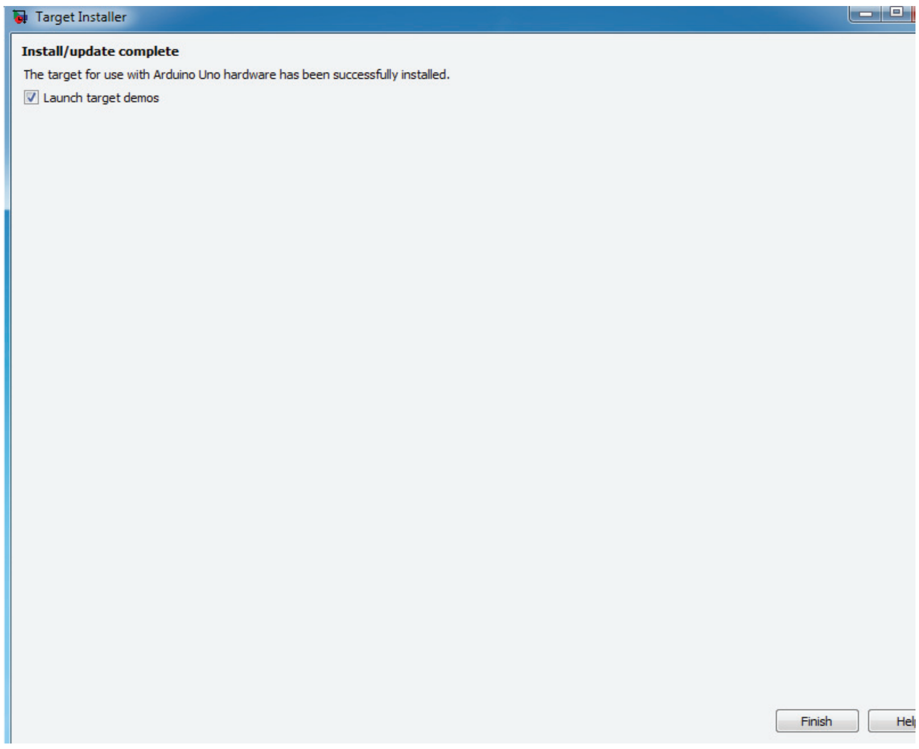
If you click "Install" below, you will be downloading and installing open source software including software licensed under the terms of the licenses linked above.

If you click "Cancel" you will not be down

Target Installer

Installing ARDUINO

Figure A.3: Step 4


Target Installer

Install/update complete

The target for use with Arduino Uno hardware has been successfully installed.

☑ Launch target demos

Figure A.4: Step 5

Appendix B

Install Support for Beagle Board Hardware

B.1 Install the Support Package

Step 1 Type targetinstaller in a MATLAB Command Window or click the Tools menu > Run on Target Hardware > Install/Update Support Package.

Step 2 Select getting the support package from the Internet or from a folder,to that the package has been downloaded (Figure B.1).

Step 3 Select the Install check box for Beagle Board target, and click Next (Figure B.2).

Step 4 Click Install and wait till the end of the progress (Figure B.3).

Step 5 Replace the firmware on the Beagle Board with Ubuntu, which described in next section (Figure B.4).

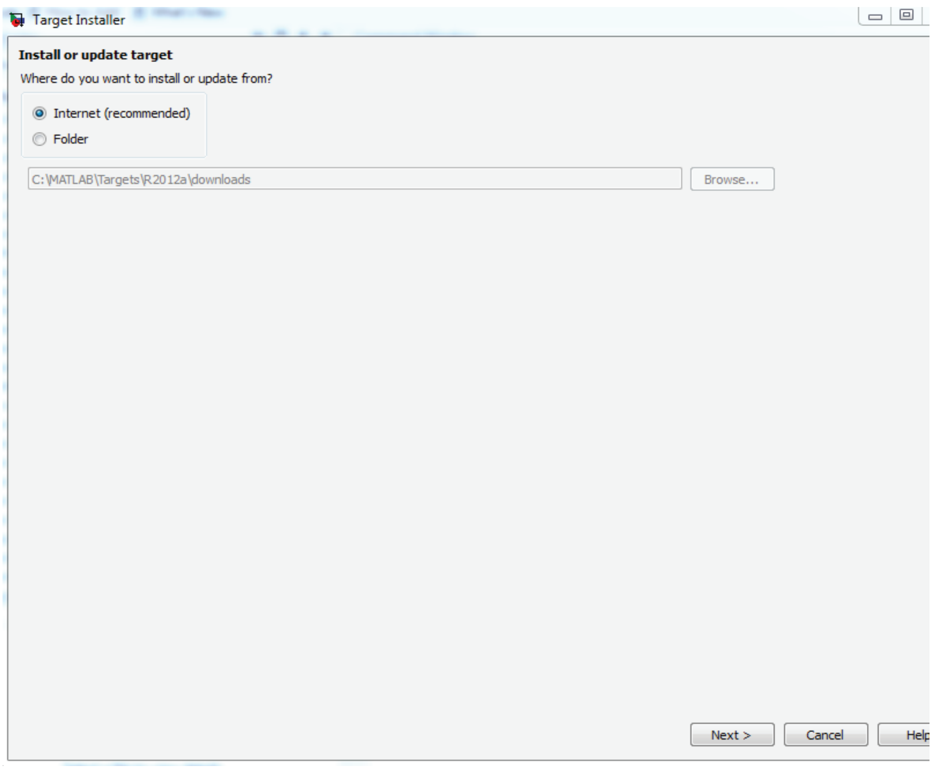
B.2 Replace the Firmware

Step 1 Choose Beagle Board and click Next (Figure B.5).

Step 2 Choose version of the board and click Next (Figure B.6).

Step 3 Connect the cables like what is shown in the Fig B.7 and click Next.

Step 4 Get the firmware image from the Internet or a folder (Figure B.8).


Target Installer

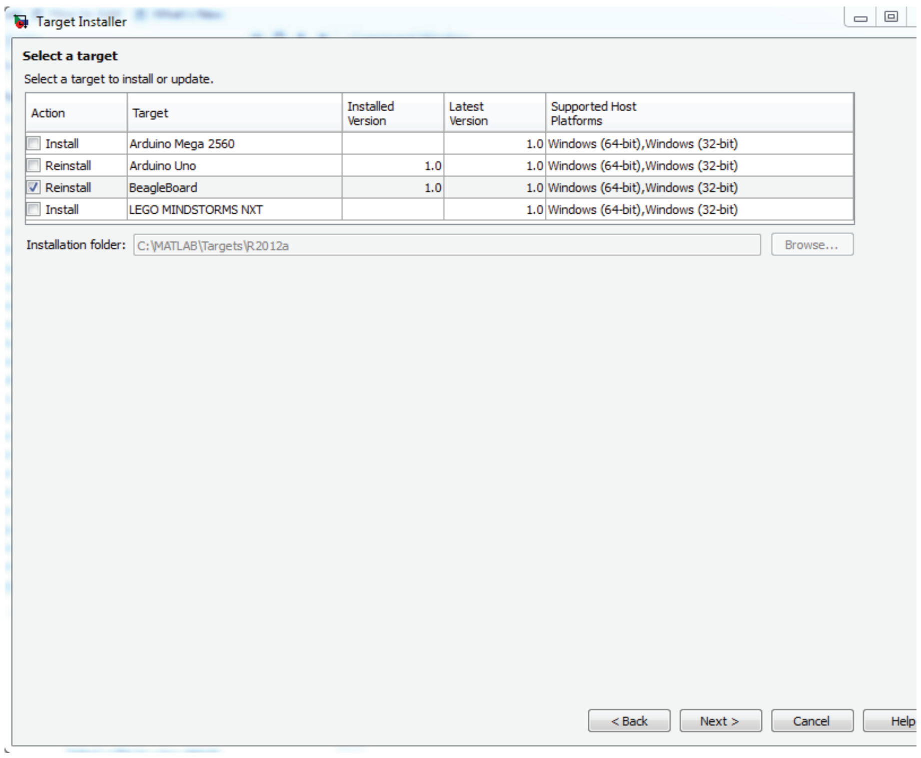
Install or update target

Where do you want to install or update from?

[✓ Internet (recommended)] Folder

C:\MATLAB\Targets\R2012a\downloads

Figure B.1: Step 2


Target Installer

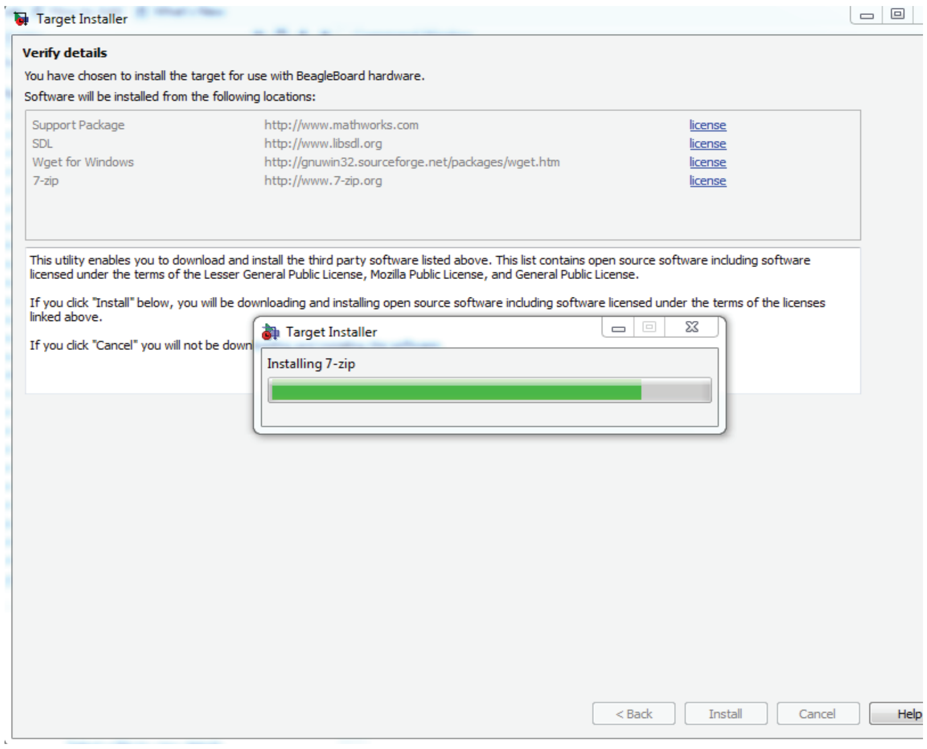
Select a target

Select a target to install or update.

Action	Target	Installed Version	Latest Version	Supported Host Platforms
Install	Arduino Mega 2560		1.0	Windows (64-bit),Windows (32-bit)
Reinstall	Arduino Uno	1.0	1.0	Windows (64-bit),Windows (32-bit)
✓ Reinstall	BeagleBoard	1.0	1.0	Windows (64-bit),Windows (32-bit)
Install	LEGO MINDSTORMS NXT		1.0	Windows (64-bit),Windows (32-bit)

Installation folder: C:\MATLAB\Targets\R2012a

Figure B.2: Step 3



You have chosen to install the target for use with BeagleBoard hardware.
Software will be installed from the following locations:

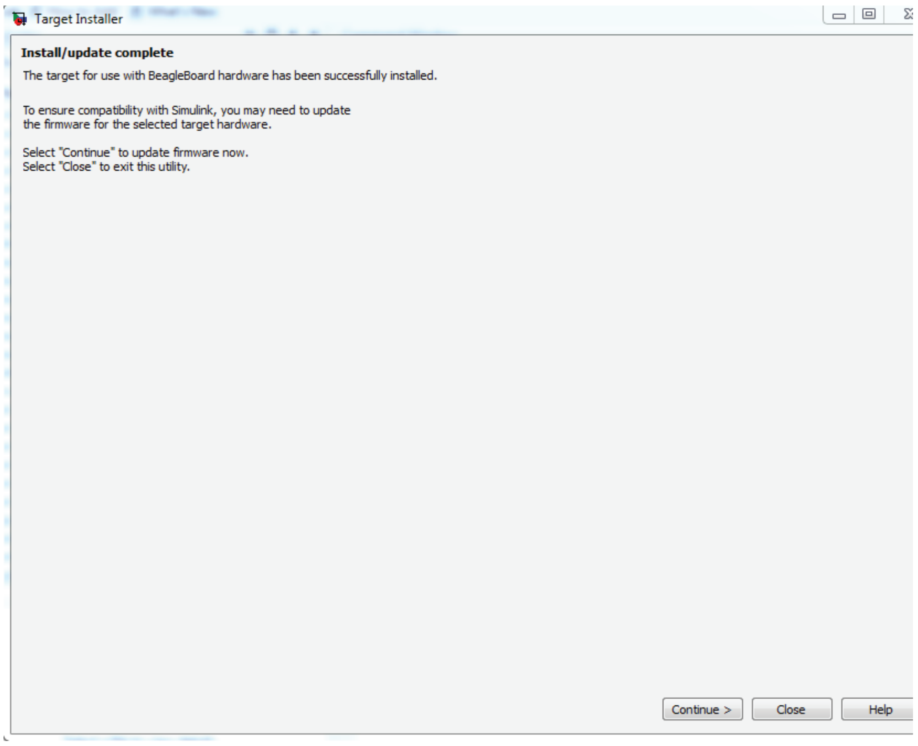
Support Package	http://www.mathworks.com	license
SDL	http://www.libsdl.org	license
Wget for Windows	http://gnuwin32.sourceforge.net/packages/wget.htm	license
7-zip	http://www.7-zip.org	license

This utility enables you to download and install the third party software listed above. This list contains open source software including software licensed under the terms of the Lesser General Public License, Mozilla Public License, and General Public License.

If you click "Install" below, you will be downloading and installing open source software including software licensed under the terms of the licenses linked above.

If you click "Cancel" you will not be down

Figure B.3: Step 4

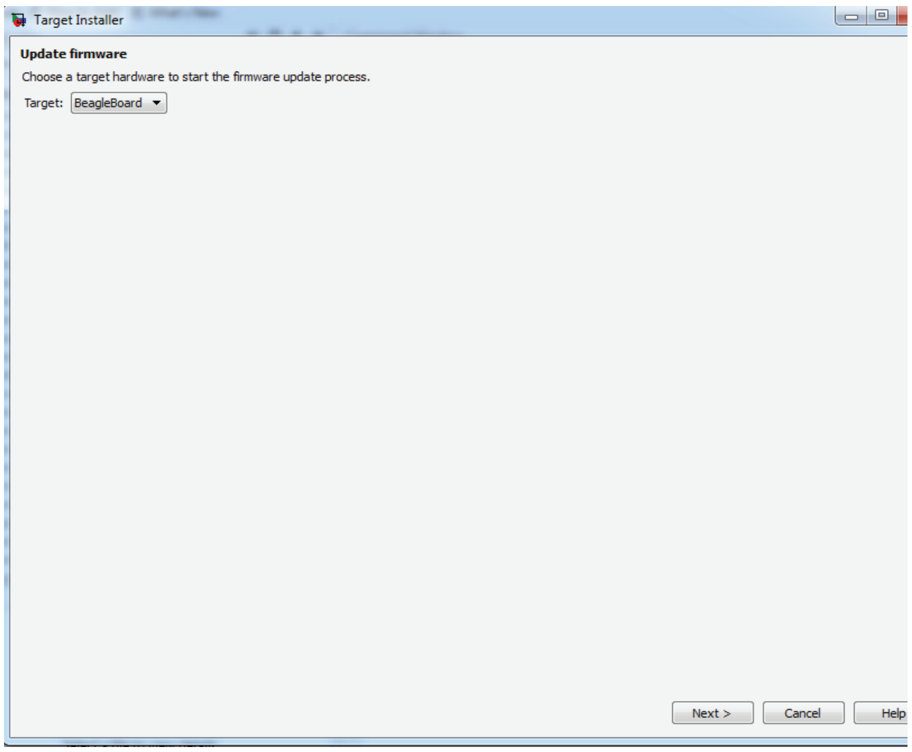


The target for use with BeagleBoard hardware has been successfully installed.

To ensure compatibility with Simulink, you may need to update the firmware for the selected target hardware.

Select "Continue" to update firmware now.
Select "Close" to exit this utility.

Figure B.4: Step 5

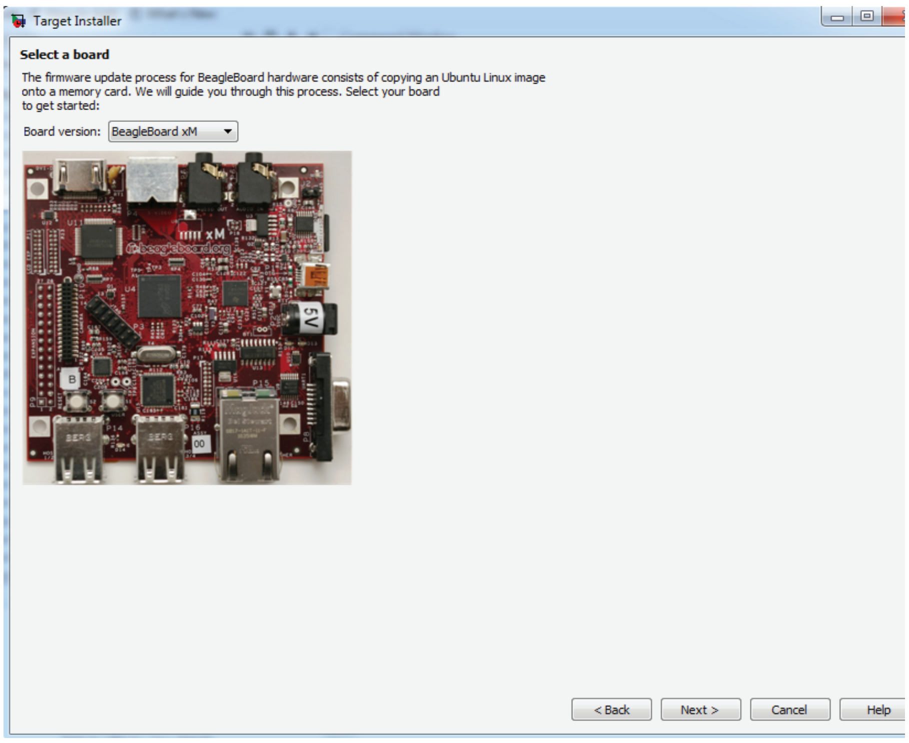


Update firmware

Choose a target hardware to start the firmware update process.

Target: BeagleBoard

Figure B.5: Replace the Firmware Step 1

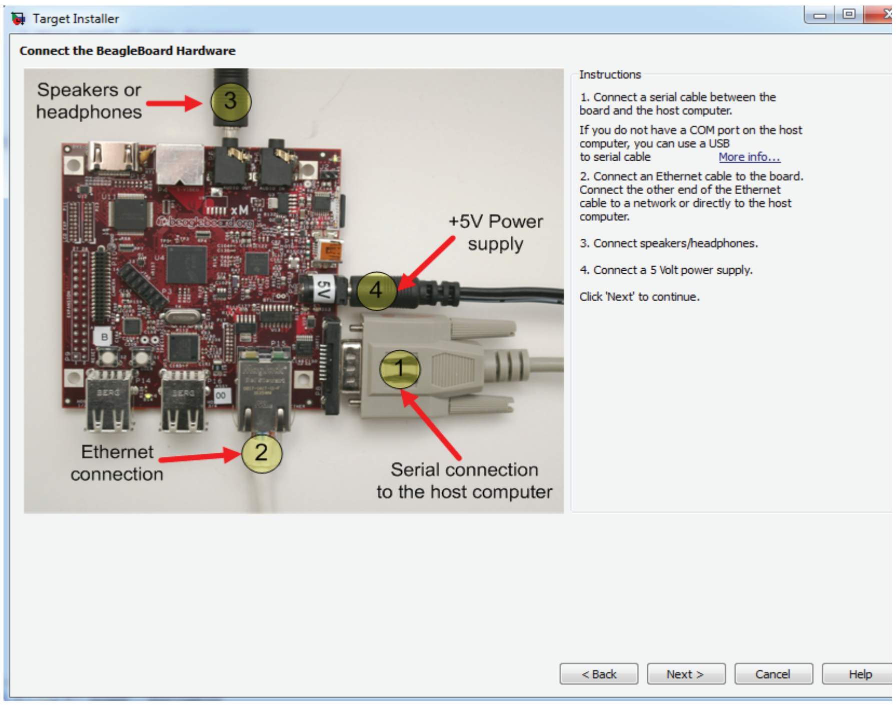


Select a board

The firmware update process for BeagleBoard hardware consists of copying an Ubuntu Linux image onto a memory card. We will guide you through this process. Select your board to get started:

Board version: BeagleBoard xM

Figure B.6: Replace the Firmware Step 2



Connect the BeagleBoard Hardware

Instructions

1. Connect a serial cable between the board and the host computer.

If you do not have a COM port on the host computer, you can use a USB to serial cable More info...

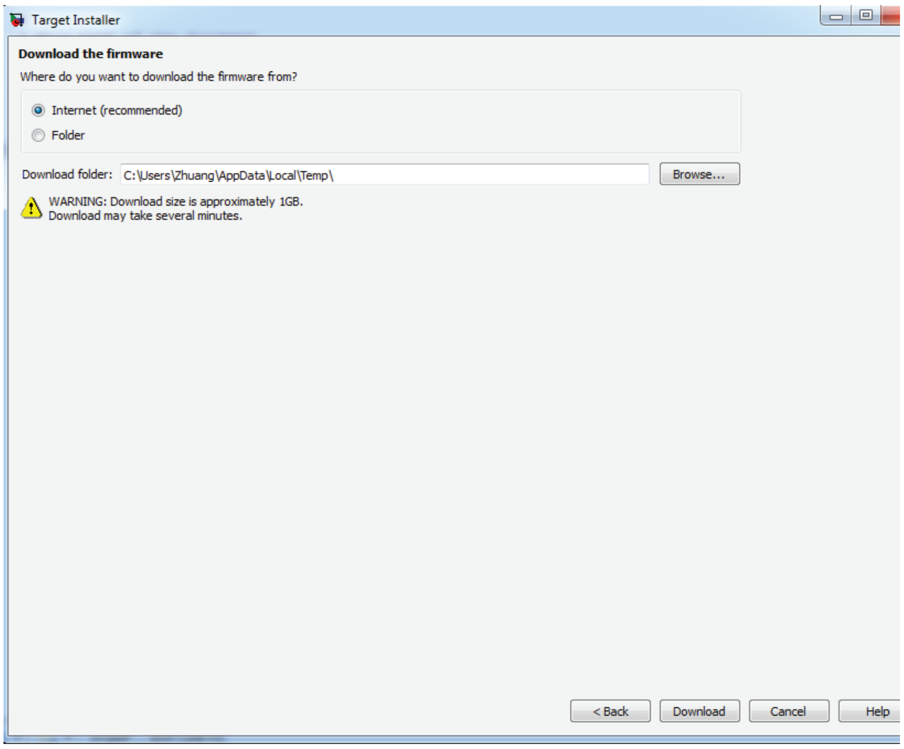
2. Connect an Ethernet cable to the board. Connect the other end of the Ethernet cable to a network or directly to the host computer.

3. Connect speakers/headphones.

4. Connect a 5 Volt power supply.

Click 'Next' to continue.

Figure B.7: Replace the Firmware Step 3



Download the firmware

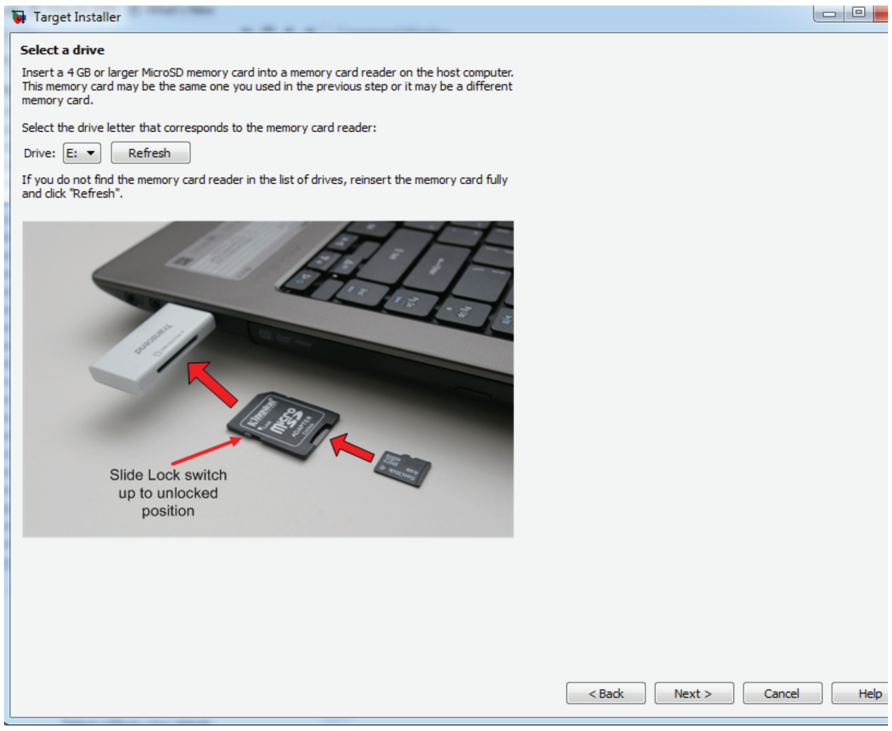
Where do you want to download the firmware from?

- [✓ Internet (recommended)]
- Folder

Download folder: C:\Users\Zhuang\AppData\Local\Temp\

WARNING: Download size is approximately 1GB.
Download may take several minutes.

Figure B.8: Replace the Firmware Step 4


Target Installer

Select a drive

Insert a 4 GB or larger MicroSD memory card into a memory card reader on the host computer. This memory card may be the same one you used in the previous step or it may be a different memory card.

Select the drive letter that corresponds to the memory card reader:

Drive: E:

If you do not find the memory card reader in the list of drives, reinsert the memory card fully and click "Refresh".

Figure B.9: Replace the Firmware Step 5

Step 5 Insert the micro SD card in the host PC with a card reader if it is necessary (Figure B.9).

Step 6 Click Write and wait for the end of the process (Figure B.10).

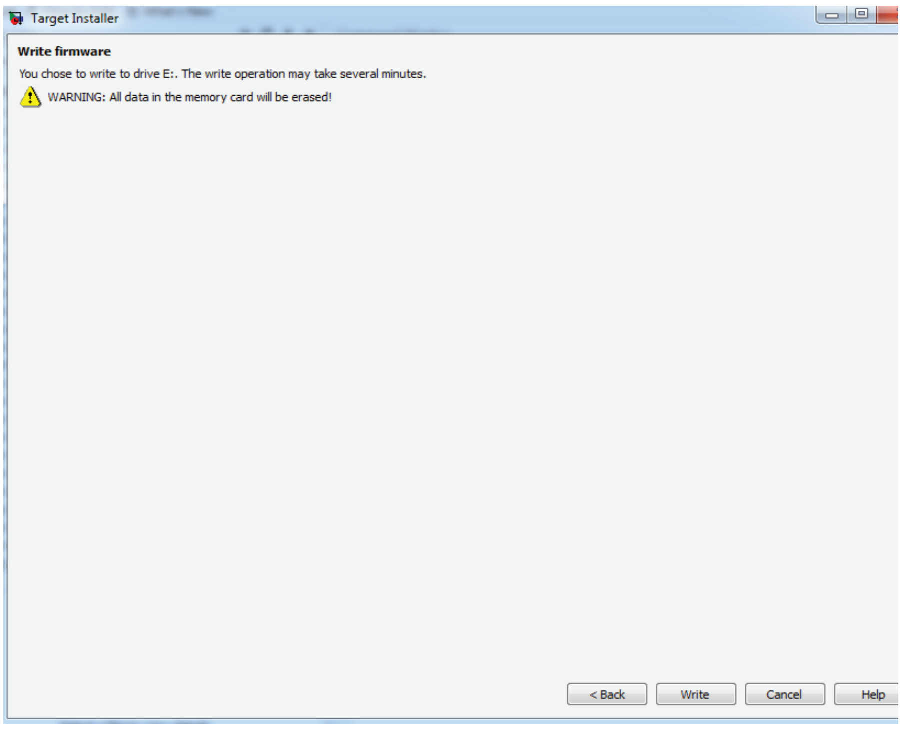
Step 7 Choose the COM port of the host and continue (Figure B.11). Pay attention that Target Installer does not automatically choose the COM port of the serial connection, before that the COM port of the serial connection in Windows must be defined.

Step 8 Insert the SD card into the Beagle Board hardware and reset (Figure B.12):

Step 9 Set the IP address as mentioned in section 2.4.3 and click Configure (Figure B.13):

Step 10 Confirm the configuration and click next (Figure B.14):

Step 11 Click Finish to end replacing the firmware (Figure B.15):

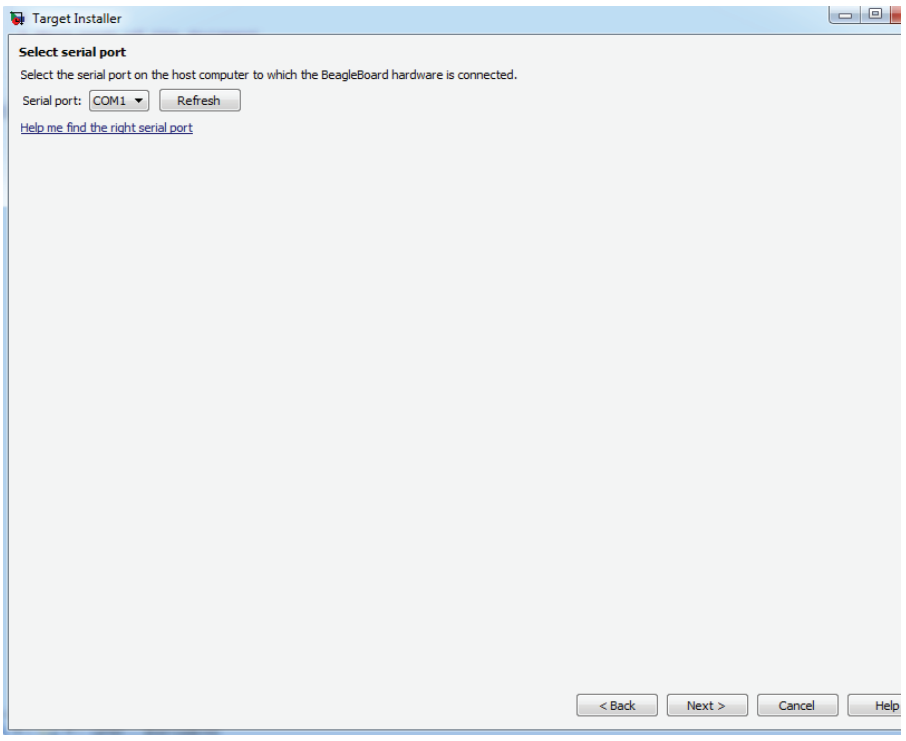


Write firmware

You chose to write to drive E:. The write operation may take several minutes.

WARNING: All data in the memory card will be erased!

Figure B.10: Replace the Firmware Step 6

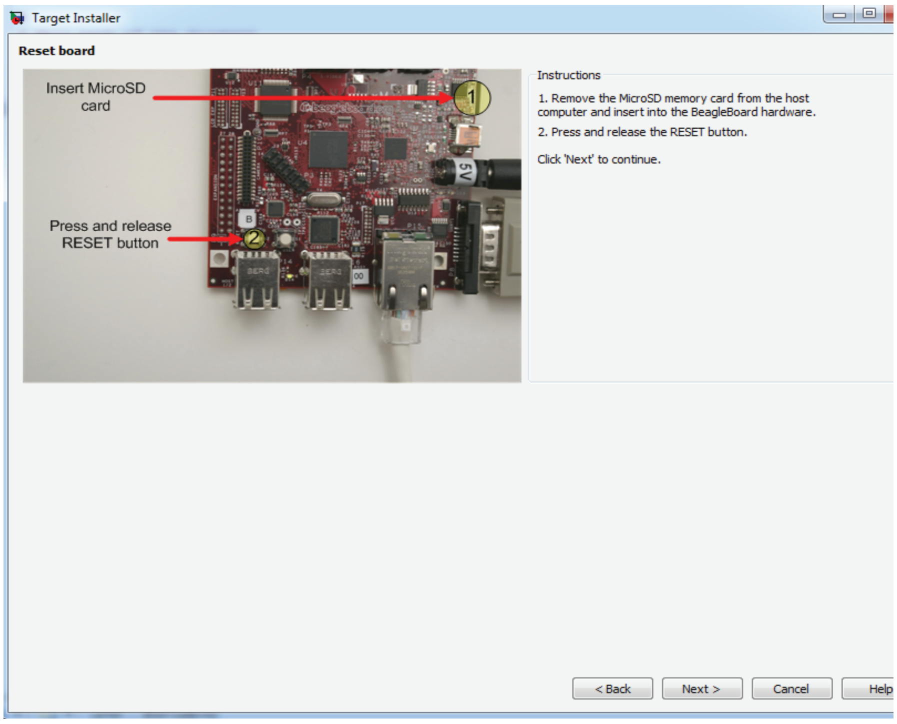


Select serial port

Select the serial port on the host computer to which the BeagleBoard hardware is connected.

Serial port: COM1

Figure B.11: Replace the Firmware Step 7


Target Installer

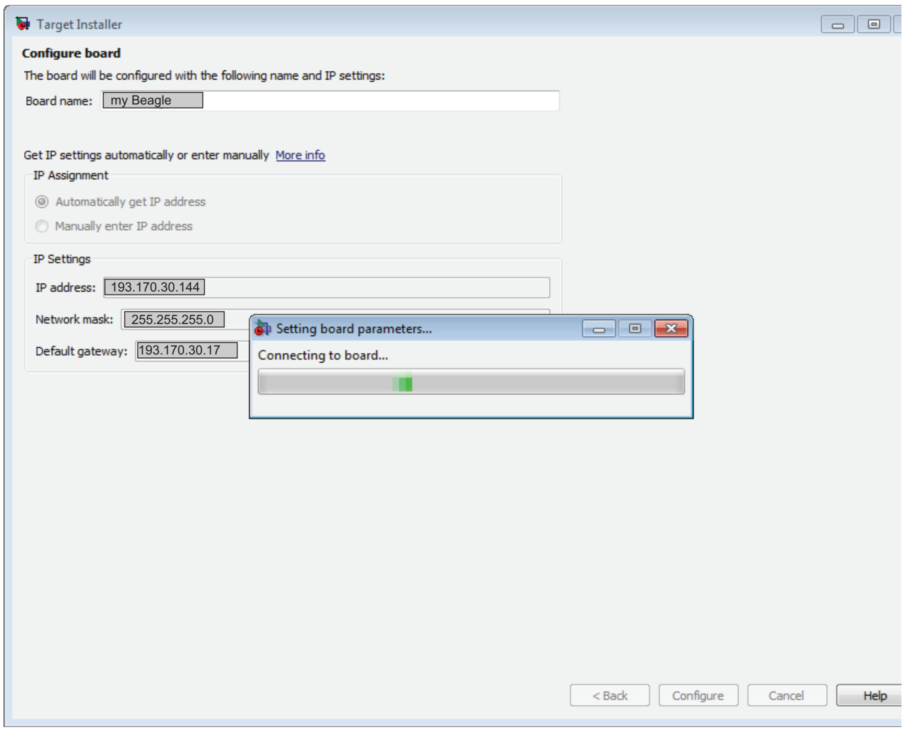
Reset board

Instructions

1. Remove the MicroSD memory card from the host computer and insert into the BeagleBoard hardware.
2. Press and release the RESET button.

Click 'Next' to continue.

Figure B.12: Replace the Firmware Step 8


Target Installer

Configure board

The board will be configured with the following name and IP settings:

Board name: my Beagle

Get IP settings automatically or enter manually More info

IP Assignment

● Automatically get IP address

○ Manually enter IP address

IP Settings

IP address: 193.170.30.144

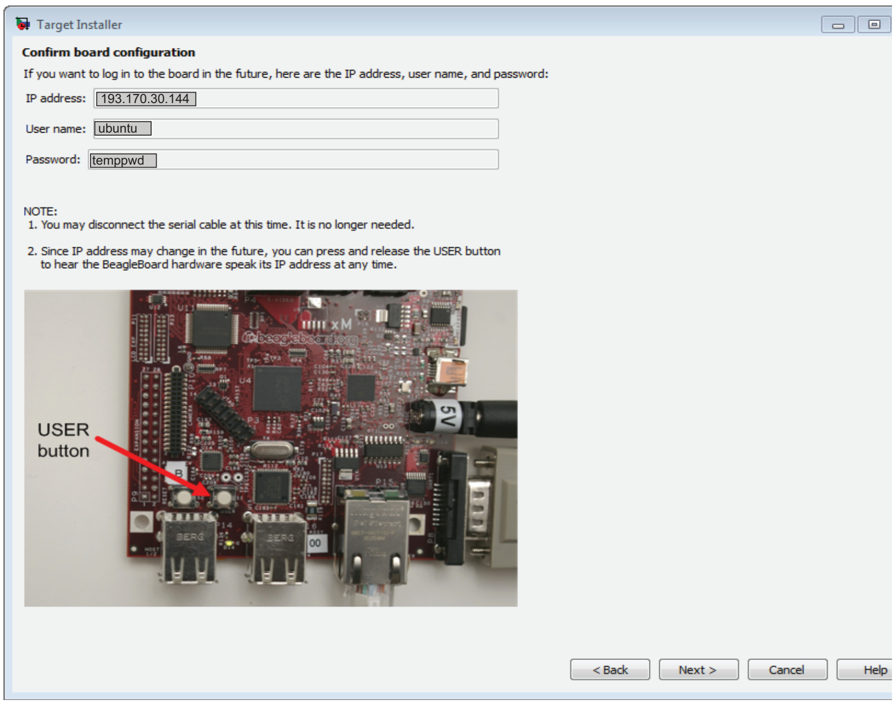
Network mask: 255.255.255.0

Default gateway: 193.170.30.17

Setting board parameters...

Connecting to board...

Figure B.13: Replace the Firmware Step 9



Confirm board configuration

If you want to log in to the board in the future, here are the IP address, user name, and password:

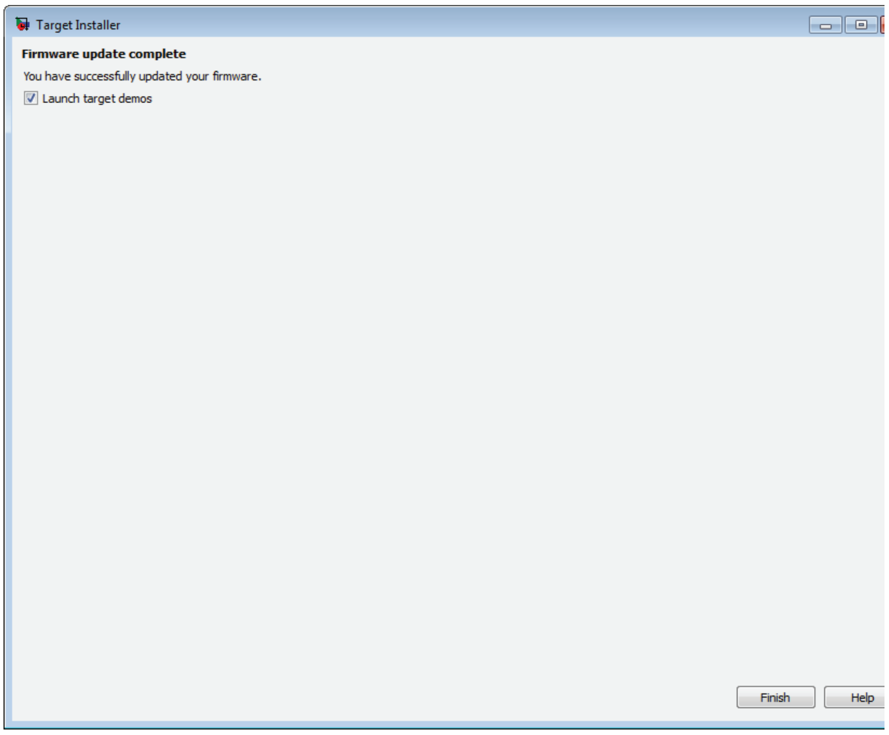
IP address: 193.170.30.144

User name: ubuntu

Password: temppwd

NOTE:
1. You may disconnect the serial cable at this time. It is no longer needed.
2. Since IP address may change in the future, you can press and release the USER button to hear the BeagleBoard hardware speak its IP address at any time.

Figure B.14: Replace the Firmware Step 10



Firmware update complete

You have successfully updated your firmware.

[✓] Launch target demos

Figure B.15: Replace the Firmware Step 11

Appendix C

A Servo Control Program Written for Arduino 1.0.1

```
#include <Servo.h>
Servo myservo;
// create servo object to control a servo
// a maximum of eight servo objects can be created
int pos = 0;
// analog pin used to connect the potentiometer
void setup()
{
myservo.attach(11);
// attaches the servo on pin 11 to the servo object
}
void loop()
{
for(pos = 0; pos < 180; pos += 1)
// goes from 0 degrees to 180 degrees
{
// in steps of 1 degree
myservo.write(pos);
// tell servo to go to position in variable 'pos'
delay(15);
// waits 15ms for the servo to reach the position
}
for(pos = 180; pos>=1; pos-=1)
// goes from 180 degrees to 0 degrees
{
myservo.write(pos);
// tell servo to go to position in variable 'pos'
delay(15) ;
// waits 15ms for the servo to reach the position
}
}
```

Appendix D

A Bandpass Filter Written with MATLAB

```
P_samplefreq = 48000;

N = length(data);

f0 = [12000 17600];
B = [0.01 0.01];
om = f0*2*pi;
nfilter = length(f0);

T = 1/P_samplefreq;

for n = 1:nfilter
        bpass(n,1) = tf([B(n)/om(n) 0], [1/(om(n)*om(n))        B(n)/om(n) 1]);
end

bpassDig = c2d (bpass, T);
```

```
impulse (bpassDig);
z = impulse (bpassDig);


figure(10);


for n = 1:nfilter
        y(n,:) = conv(data1,z(:,n),'same');
        x(n,:) = 1:N;
        subplot(nfilter+1,1,n+1)
        plot(x(n,:),y(n,:))
end
subplot(nfilter+1,1,1);
plot(data)
```

Appendix E

MATLAB Function for Peak Detector

```
function y = fcn2(u)
%#codegen
        T = 1/48000;
        tau = 0.0005;
        max = u(1);


        for i=2:128;
               if u(i)>max
                      max=u(i);


end


        max = max*(1-T/tau);
        end


        y = max;
```