Summer Fellowship Report

On

Developing a Generic Purpose OpenModelica Package For Embedded Applications

Submitted by

Mayank Mangla
Pragya Jha
Ritu Kanwar Shekhawat
Sumeet Koli

Under the guidance of

Mentor: Mr. Manas Ranjan Das

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

July 7, 2018

Acknowledgment

We are grateful to FOSSEE, IIT Bombay and our mentor Mr. Manas Ranjan Das for giving us the opportunity to work on the project Implementing an embedded package for OpenModelica. We express our sincere gratitude to our mentor Mr. Manas Ranjan Das for his support, help and encouragement throughout the project. We would also like to thank Professor Kannan M.Moudgalya who introduced us to FOSSEE, its projects and the significance of its work.we are grateful for we would not have worked on the open source software at this level of exposure without your guidance and support.

Also we would like to recognize the support and contribution of all the team members.

Contents

1 Introduction **5**

2 Implementation **6**
2.1 Algorithm 6
2.2 Making changes in source code 7
2.2.1 How to make changes to source code and make libraries 7
2.2.2 Working with Arduino UNO [Atmega328p] 7
2.2.3 Working with Tiva C [TM4C123G] 8

3 Download and Installation **9**
3.1 OpenModelica 9
3.2 Arduino IDE 9
3.3 Energia IDE 9

4 About OpenModelicaEmbedded package **10**
4.1 SynchronizeRealTime Block 10
4.2 Pins 11
4.3 Boards 14
4.4 Examples 15
4.5 ArduinoExamples 15
4.6 TivaC_Examples 16
4.7 Internal 16

5 Hardware In Loop Simulation **17**
5.1 Implementation 17

6 PID Controller **18**
6.1 Implementation 18
6.2 Example for PID 19

7 Working with Arduino UNO **20**
7.1 Connecting and Configuring the Board 21
7.2 Interfacing with OpenModelica 21
7.3 Examples for Arduino 21
7.3.1 LED Examples 22
7.3.2 Push Button Examples 22
7.3.3 LDR Examples 23

7.3.4 DC Motor Examples 24
7.3.5 Potentiometer Examples 25
7.3.6 Thermistor Examples 26
7.3.7 Servo Motor Examples 27

8 Working with Tiva C Launchpad **29**
8.1 Connecting and Configuring the Board 29
8.2 Interfacing with OpenModelica 30
8.3 Examples for Tiva C 30

9 Conclusion **32**

List of Figures

4.1 Structure of OpenModelicaEmbedded package 10
4.2 SymchronizeRealTime block 11
4.3 AnalogInput block 12
4.4 AnalogOutput block 12
4.5 DigitalInput block 13
4.6 DigitalOutput block 13
4.7 Servo block 14
4.8 Arduino block 14
4.9 ArduinoLeonardo block 14
4.10 StandardFirmata block 15
4.11 CustomFirmata block 15
4.12 customBoard block 15

6.1 Firmata to work with PID Controller 18
6.2 Model for PID Controller with DC Motor 19
6.3 Plot for PID Controller with DC Motor 19

7.1 Pin Diagram of Arduino UNO 20
7.2 Arduino Led Example 22
7.3 Arduino Push Button Example 23
7.4 Print Statement 23
7.5 Arduino LDR Example 24
7.6 Arduino DC Motor Example 25
7.7 Arduino Potentiometer Example 26
7.8 Arduino Thermistor Example 27
7.9 Data Sheet for Servo Motor SG90 28
7.10 Arduino Servo Motor Example 28

8.1 Pin Diagram of Tiva C Launchpad 29
8.2 Tiva C Led Example 31

Chapter 1

Introduction

OpenModelica is a free and open source environment based on the Modelica modeling language for simulating, optimizing and analyzing complex dynamic systems. OpenModelica is used in academic and industrial environments. Industrial applications include the use of OpenModelica along with proprietary software in the fields of power plant optimization, automotive and water treatment. Models are either built through line by line code or graphical code in OpenModelica. OpenModelica can interact with C, Python languages and can call C, Python functions from within its models. OpenModelica is a powerful tool that can be used to design and simulate complete control systems.

Our project was to implement model based design i.e. creating models for different embedded applications and generating C code that can be ported to the respective family of microcontrollers.We also worked towards Improving the Hardware In Loop (HIL) implementation on OpenModelica and microcontrollers like arduino and TivaC.The current implementation which is based on Inter Process Communication (IPC) needs to be made robust. Hence there is a need to come up with a less cumbersome IPC for HIL.

Chapter 2

Implementation

2.1 Algorithm

1. Once you have installed OpenModelica, launch OMEdit and open the OpenModelicaEmbedded package.

2. To use the above package you will also need to load Modelica_DeviceDrivers package. The synchronizeRealtime block present in this package is used to make the simulation of models real-time. All it does is that, it maps the time interval provided by you before simulation with clock your PC.

3. The components provided in this package are:

   (a) Pins: It contains Analog input, Analog output, Digital input, Digital output and Servo pins to perform corresponding function in model.

   (b) Boards: Any of the provided board can be used depending the one you are using, else use the customBoard provided and vary its parameters to match the configuration of the development board you are using.

4. Take a look at the examples provided along with the package to understand the basics structure of a model. Each model has Board block which represents the development board used. This block when added to a model, on simulation calls a couple of functions present in Internal ¿ ExternalFunctions which set the initialisation parameters for communication like PORT, BAUD rate, etc.

5. These modelica functions present in ExternalFunctions then call external C functions which perform the actual task the function is supposed to do.

6. These external C functions are bundled together and provided in the form of libraries. The Libraries used will be *.dll in case of WindowsOS and *.so in case of Linux.

7. After adding a board to your model add pins using blocks provided for the same. If you want to send some data from OpenModelica to connected microcontroller the use Analog/Digital Output Pin, and vice versa. Use Analog Pin while working with real data and Digital pin while working with Boolean.

8. These pin blocks again call functions in similar manner to either send or receive data.

9. Once your model is ready and check is successful, upload appropriate Firmware on microcontroller board connected.

10. The Firmwares for Arduino and Tiva C borads have been provided along with the package. Open Arduino IDE if using Arduino board and Energia IDE if using Tiva C board and upload corresponding firmware on board.

11. The Firmware implements Firmata protocol to establish communication with OpenModelica.

12. Now go back to OpenModelica and simulate your model.

2.2 Making changes in source code

2.2.1 How to make changes to source code and make libraries

Linux Operating System

Open the source codes by browsing to this location : OpenModelicaEmbedded →Source.After making changes to the source code files open Terminal.Browse to OpenModelicaEmbedded →Source folder using cd command.Run command make to generate shared object file('*.so').

Windows Operating System

Open the source codes by browsing to this location : OpenModelicaEmbedded →Source. After making changes to the source code, open Command Prompt (cmd).Browse to OpenModelicaEmbedded →Source folder using cd command.To compile the CPP files run the command: g++ -c modelPlugFirmata.cpp serial.cpp. To create a DLL from generated object files, run the command: g++ -shared -o modelPlugFirmata.dll modelPlugFirmata.o serial.o.Then copy the generated DLL file and paste it in folders: OpenModelicaEmbedded and Resources →Library →win64.

2.2.2 Working with Arduino UNO [Atmega328p]

Setting up firmware for Arduino board
In Tools Menu, select appropriate Board (Arduino/Genuino UNO) and Port as the available serial port to which Arduino is connected.
Open pidmata3 sketch: File →Open →OpenModelicaEmbedded →Firmware →Arduino →pidmata3 →pidmata3.ino. Upload the sketch to the board.

Simulating the Modelica model

Now open OMEdit window.
Open package.mo file OpenModelicaEmbedded folder.
In OpenModelicaEmbedded package, open ArduinoExamples package which consists of examples for arduino board. Check and simulate the example models and verify the results.

2.2.3 Working with Tiva C [TM4C123G]

In Energia, open the firmware for Tiva C provided in folder through path : File →Open →OpenModelicaEmbedded →Firmware →Tiva C →StandardFirmata → StandardFirmata.ino or add zip file of this StandardFirmata as an external library in Energia from the same folder.
Select appropriate Board (Tiva C) and Port (USB port where Tiva C is connected) in Tools menu.
Then, upload the firmware on board.

Simulate a model with Tiva C.

Now open OMEdit and Open the package.mo file from OpenModelicaEmbedded package
Open an example provided in the OpenModelicaEmbedded package which includes a Tiva C board.
Check and Simulate the model and verify the results in Plotting window.

Chapter 3

Download and Installation

3.1 OpenModelica

OpenModelica can be downloaded online from https://openmodelica.org.

For Linux:
The Debian/Ubuntu package for OpenModelica can be downloaded and installed by following the instructions on https://openmodelica.org/download/download-linux. OMEdit can be launched by typing OMEdit in the Terminal.

For Windows:
The setup for OpenModelica on Windows can be downloaded from https://openmodelica.org/download/download-windows. After downloading and installation of the software, OMEdit or OpenModelica Connection Editor can be launched by clicking on its icon or by navigating through Start menu.

3.2 Arduino IDE

The open source Arduino Software is used to write codes and upload them to the board. For the setup of Arduino IDE for both Windows and Linux go to https://www.arduino.cc/en/Main/Software and follow the download and install instructions.

3.3 Energia IDE

Energia is a Arduino-like coding environment designed for Texas Instruments Launchpads. The setup for Energia can be downloaded from http://energia.nu/download/. The instructions for installation can be referred from -
For Linux: http://energia.nu/guide/guide_linux/ For Windows: http://energia.nu/guide/guide_windows/

Chapter 4

About OpenModelicaEmbedded package

OpenModelicaEmbedded package is designed to interact between embedded systems and OpenModelica models using C/C++ functions. This package can work for both Atmega series (tested for Arduino UNO - Atmega328p) and Texas Instruments Tiva C series (tested for Tiva C EK-TM4C123GXL Launchpad).

The package can be downloaded from the following link:
https://github.com/manasdas17/OpenModelicaEmbedded
This library requires Modelica_DeviceDrivers library as a supporting library which can be downloaded from the following link:
https://github.com/modelica/Modelica_DeviceDrivers

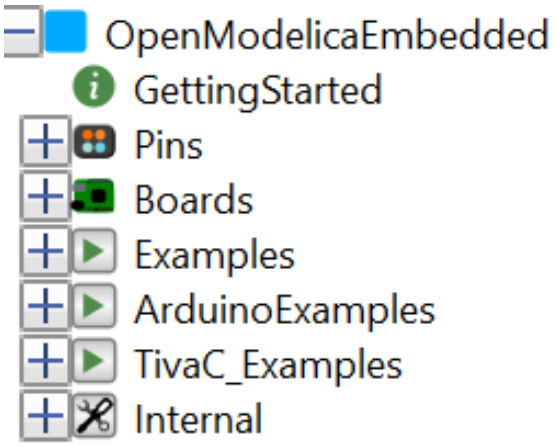


Figure 4.1: Structure of OpenModelicaEmbedded package

4.1 SynchronizeRealTime Block

This block is a part of Modelica_DeviceDrivers library used for real-time simulation of the model, i.e., this block synchronizes simulation time of the process to real-time clock of the operating system. Without this block, the models designed using this package will not be able to give proper real-time output. This block works

at five different priorty levels which can be changed in Parameters dialog box by double-clicking the block.

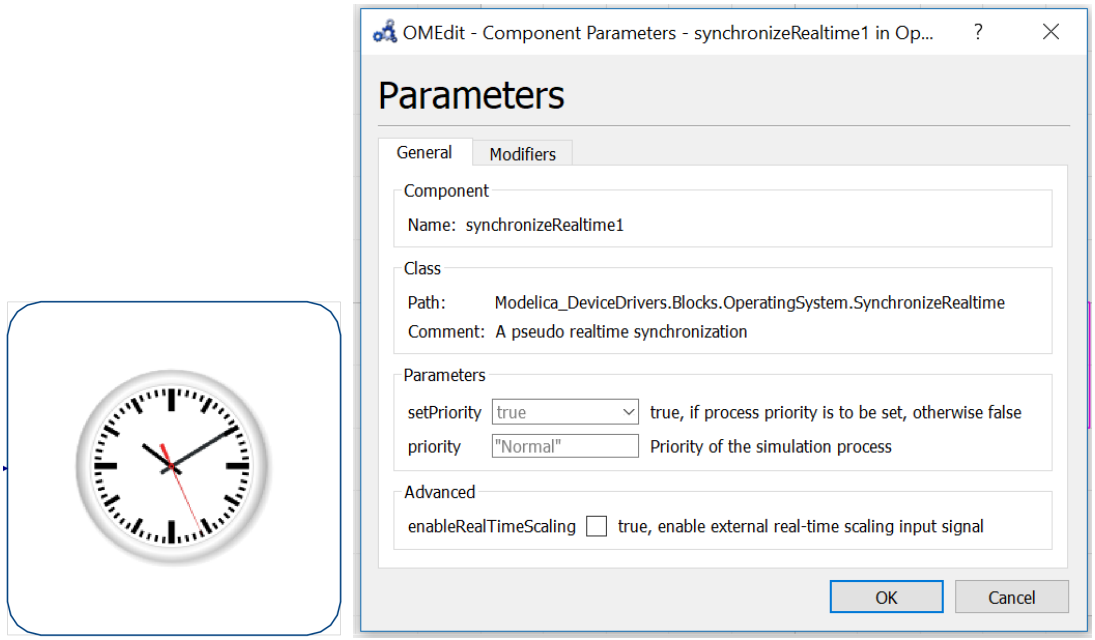


Figure 4.2: SymchronizeRealTime block

4.2 Pins

This package contains blocks which define to input and output pins of the board to which our hardware can be connected. These pin components define the properties and working of the pins used in the hardware.

- **AnalogInput:** It reads an analog signal from the specified pin. This component uses the function analogRead of Arduino. It takes minimum and maximum values of the signal as parameter (default values being 0 and 1 respectively) and gives output depending on the size of ADC (analog to digital convertor) which is 10-bit for Arduino UNO board and 12-bit for Tiva C series TM4C123G board.

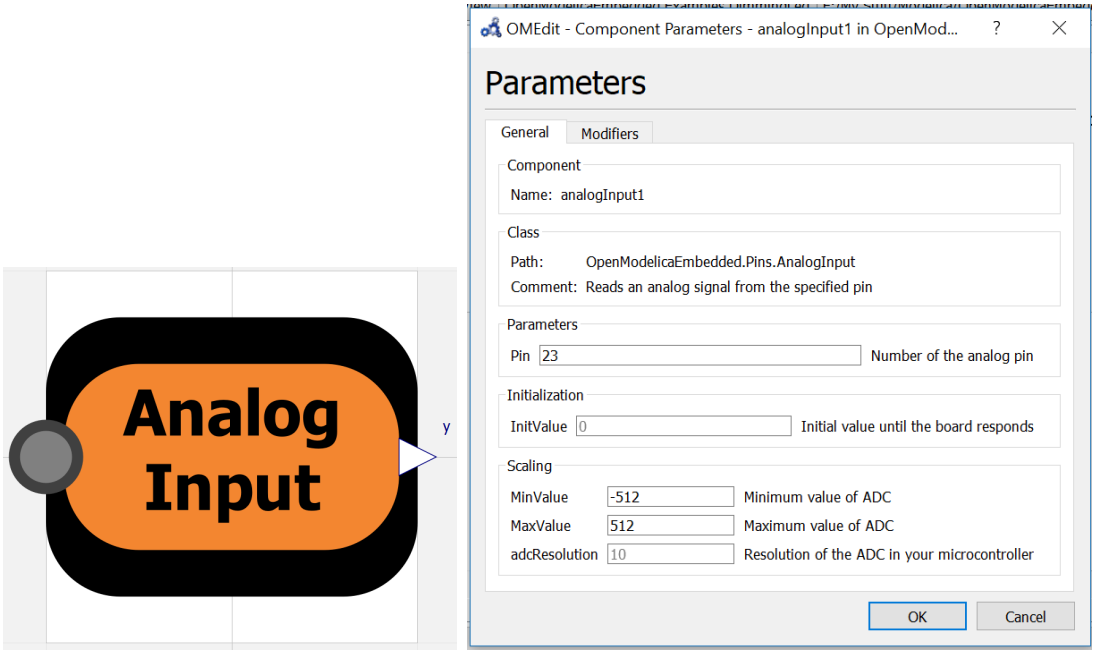


Figure 4.3: AnalogInput block

- **AnalogOutput:** It writes analog value (PWM wave) to the specified pin. This component uses the function analogWrite of Arduino. It takes minimum and maximum values of the signal as parameter (default values being 0 and 1 respectively) and gives output depending on the size of ADC.

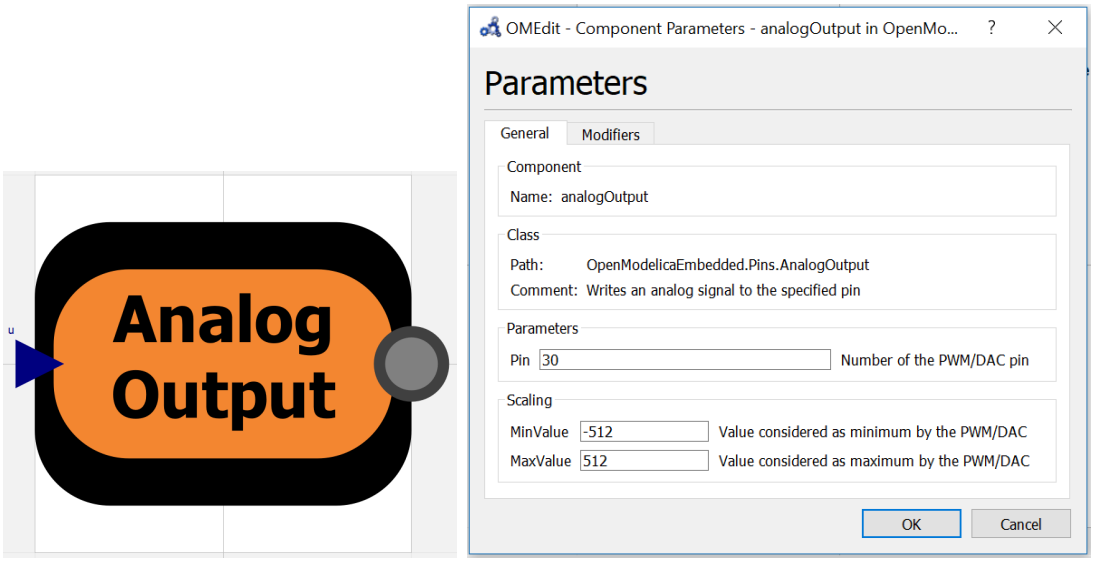


Figure 4.4: AnalogOutput block

- **DigitalInput:** It reads an digital signal from the specified pin. This component uses the function digitalRead of Arduino. It only takes boolean signals.

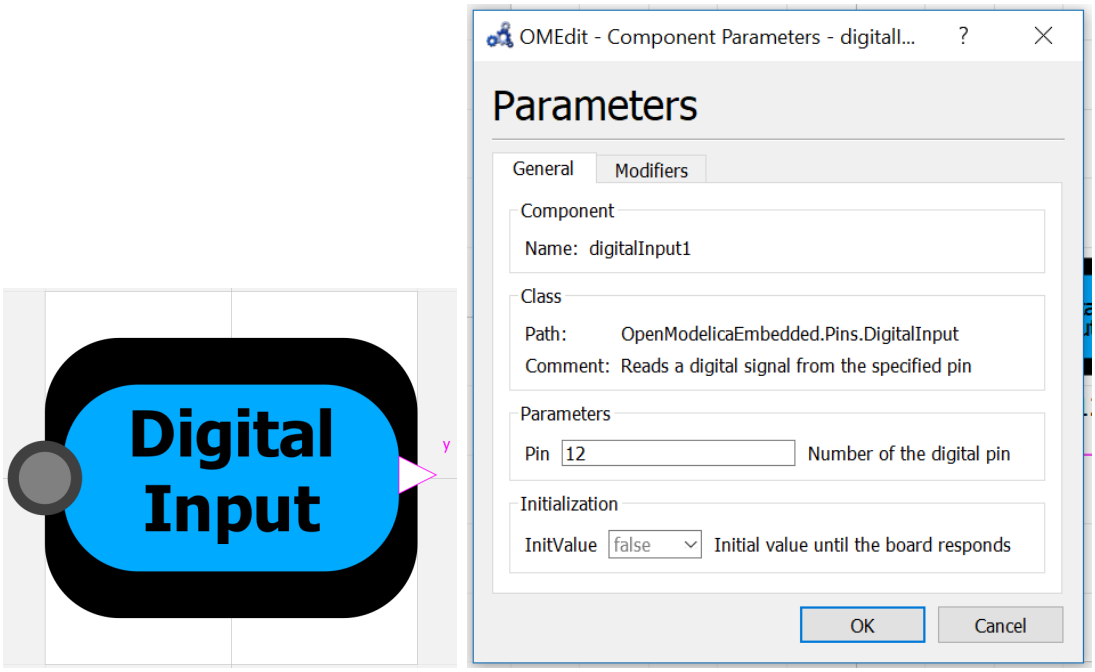


Figure 4.5: DigitalInput block

- **DigitalOutput:** It writes digital value to the specified pin. This component uses the function digitalWrite of Arduino. It only takes boolean signals.

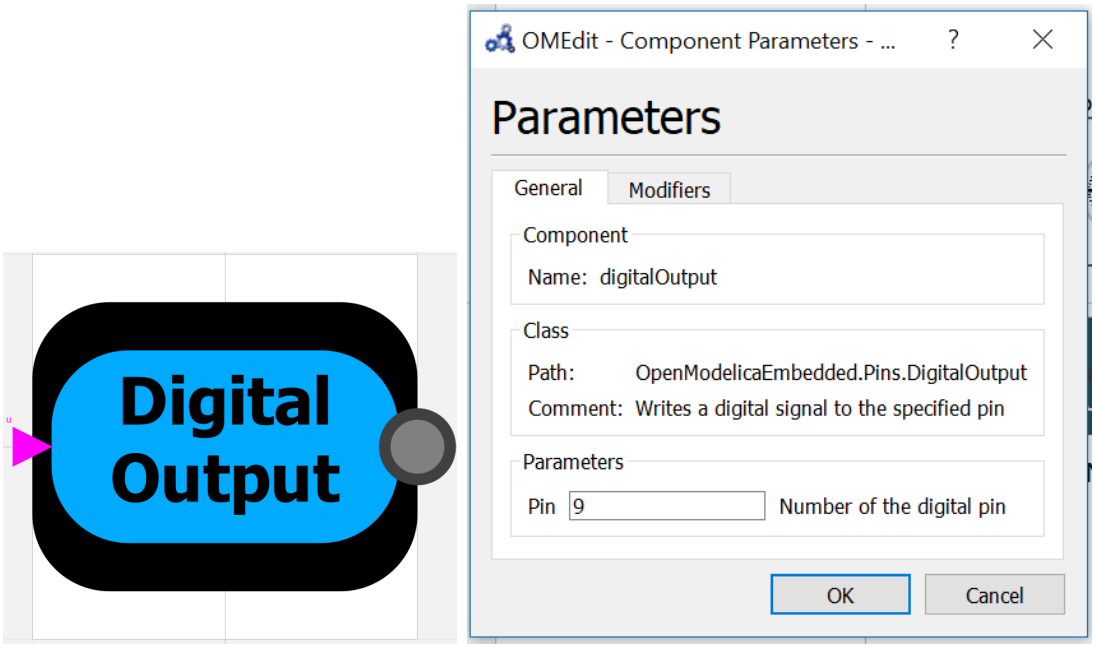


Figure 4.6: DigitalOutput block

- **Servo:** It controls a servo motor attached to the specified pin. This component uses the 'Servo' library of Arduino. By default, the range goes from 0 to 1, which corresponds to 0 to 180 degrees. If you want to input values in degrees or radians, you can change the parameter 'InputUnit' to 'Degrees' or 'Radians'.

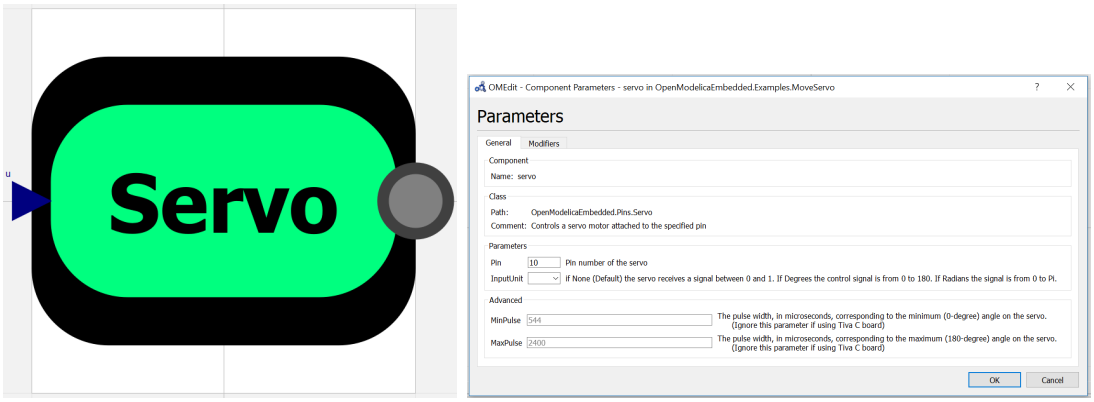


Figure 4.7: Servo block

4.3 Boards

This package contains block components which enable connection with different firmata boards. These components take serial port used for connection as parameter.

- **Arduino:** Used for connecting to Arduino boards, such as Arduino UNO, Arduino Mega 2500 and others having similar firmata.

Figure 4.8: Arduino block

- **ArduinoLeonardo:** Supports Arduino Leonardo board and other boards using native USB.

Figure 4.9: ArduinoLeonardo block

- **StandardFirmata:** Connects to Arduino compatible boards.

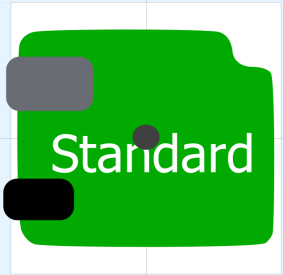


Figure 4.10: StandardFirmata block

- **CustomFirmata:** Supports any board firmata.

Figure 4.11: CustomFirmata block

- **customBoard:** Takes name of the board also as parameter and can be used to connect any board supporting firmata.

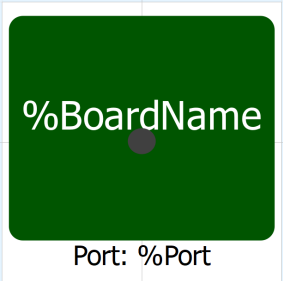


Figure 4.12: customBoard block

4.4 Examples

The package contains example models to get introduced to the working of components of the package. These models accomplish different tasks using different blocks provided in Pins and Boards package.

4.5 ArduinoExamples

The package contains example models specifc to Arduino UNO board. It can be used for other Arduino boards by change Pin Number parameter for the Pin blocks. Detailed description for the package is given in 'Working with Arduino'.

4.6 TivaC_Examples

The package contains example models specifc to Tiva C Launchpad TM4C123G board. It can be used for other similar boards by change Pin Number parameter for the Pin blocks. Detailed description for the package is given in 'Working with Tiva C'.

4.7 Internal

The components of this package cannot be used directly in the models. This package consists of icons and connectors defined and used for block designing, new types defined and used for the blocks and functions designed using external C/C++ functions at backend.

Chapter 5

Hardware In Loop Simulation

Hardware-in-the-loop (HIL) simulation, or HWIL, is a technique that is used in the development and test of complex real-time embedded systems. HIL simulation provides an effective platform by adding the complexity of the plant under control to the test platform. The complexity of the plant under control is included in test and development by adding a mathematical representation of all related dynamic systems. These mathematical representations are referred to as the plant simulation. The embedded system to be tested interacts with this plant simulation.

A HIL simulation must include electrical emulation of sensors and actuators. These electrical emulations act as the interface between the plant simulation and the embedded system under test. The value of each electrically emulated sensor is controlled by the plant simulation and is read by the embedded system under test (feedback). Likewise, the embedded system under test implements its control algorithms by outputting actuator control signals. Changes in the control signals result in changes to variable values in the plant simulation.

5.1 Implementation

The package developed (OpenModelicaEmbedded) has several components like micro-controller boards, Digital/Analog Pins, etc. which you will have to use in your model to make it interact with connected hardware device. These components make calls to external C functions present in the library provided in Library directory. Those functions using serial communication communicate with the connected device. This source file will remain same irrespective of the connected hardware device and platform used (Windows, Linux, Mac).

The connected hardware device uses Firmata protocol to communicate with OpenModelica. The source code implementing Firmata protocol on hardware will vary depending on the language/ IDE used by that hardware/microcontroller, but the underlying protocol remains the same.

Chapter 6

PID Controller

A **proportional - integral - derivative controller** (**PID controller** or **three term controller**) is a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an error value e(t) as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively) which give the controller its name.

6.1 Implementation

The PID is implemented in the Firmware present on the hardware device using AutoPID Library. To use the hardware for PID you will have to change the Firmware (Arduino source code) a bit by following the instructions mentioned in the Firmware itself. The only thing you have to do is just comment/uncomment a few macros depending on your application.

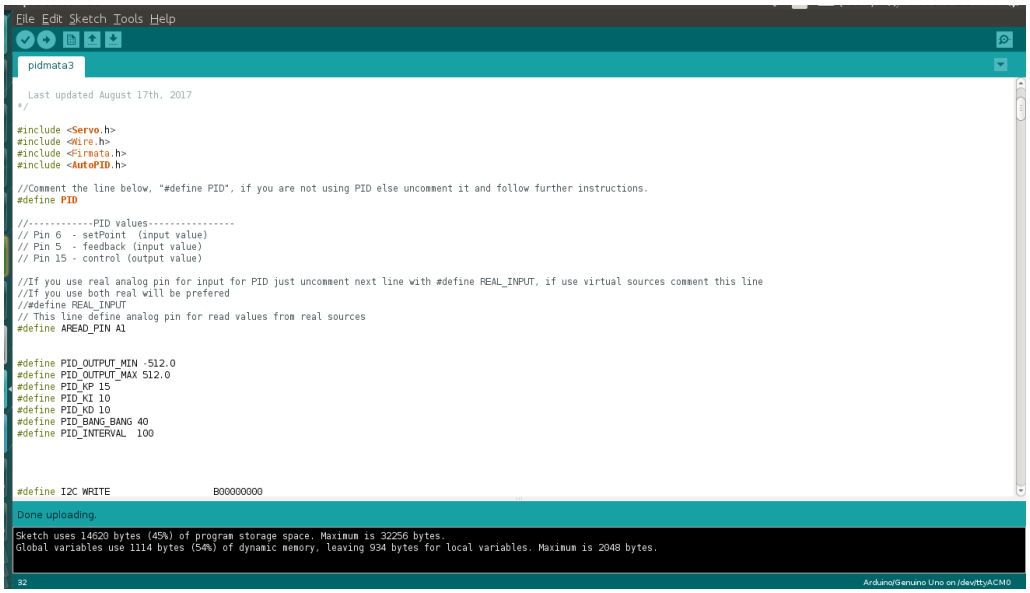


Figure 6.1: Firmata to work with PID Controller

6.2 Example for PID

Example: DCMotorWithPID (from Examples package)

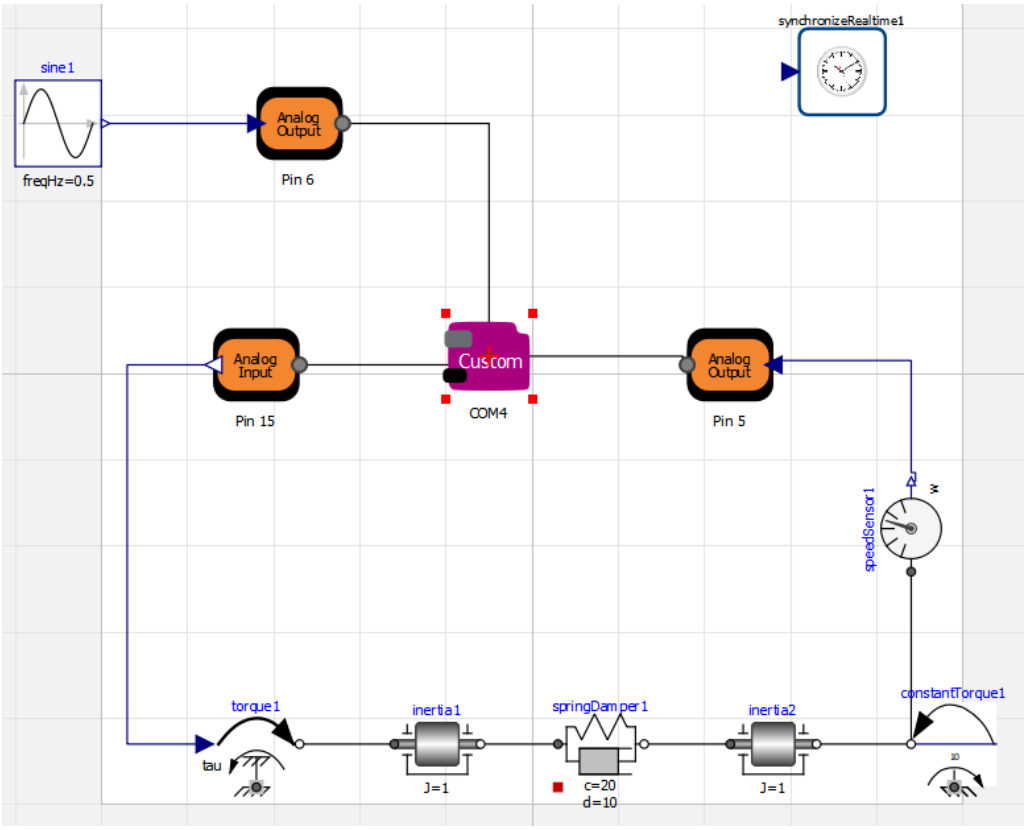


Figure 6.2: Model for PID Controller with DC Motor

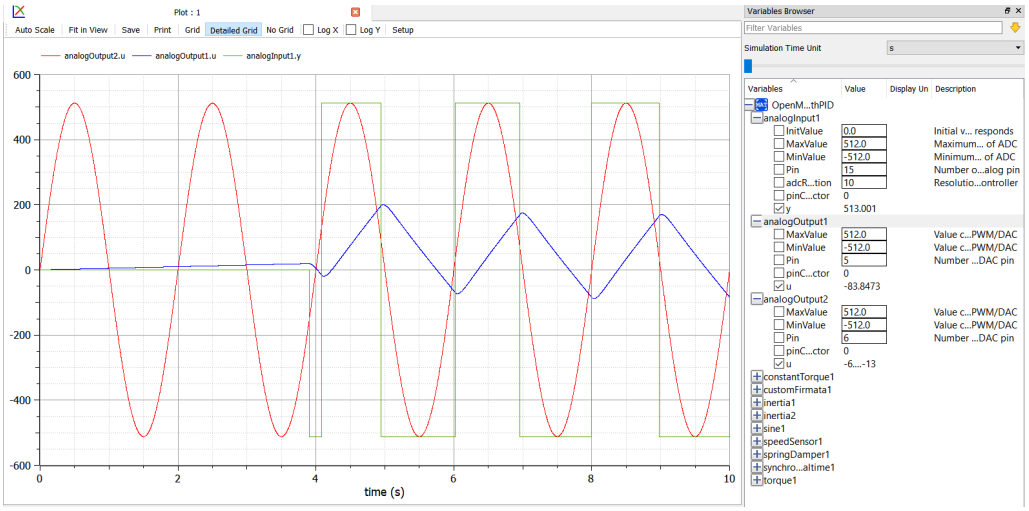


Figure 6.3: Plot for PID Controller with DC Motor

Chapter 7

Working with Arduino UNO

The Arduino UNO is a widely used microcontroller board based of ATmega328P microcontroller IC and is developed by Arduino.cc. It operates at a voltage of 5V. The board contains 14 Digital and 6 Analog input/output (I/O) pins, a 10-bit ADC (Analog to Digital Convertor), 8-bit DAC, a in-built LED connected to digital pin no. 13 and many other features.

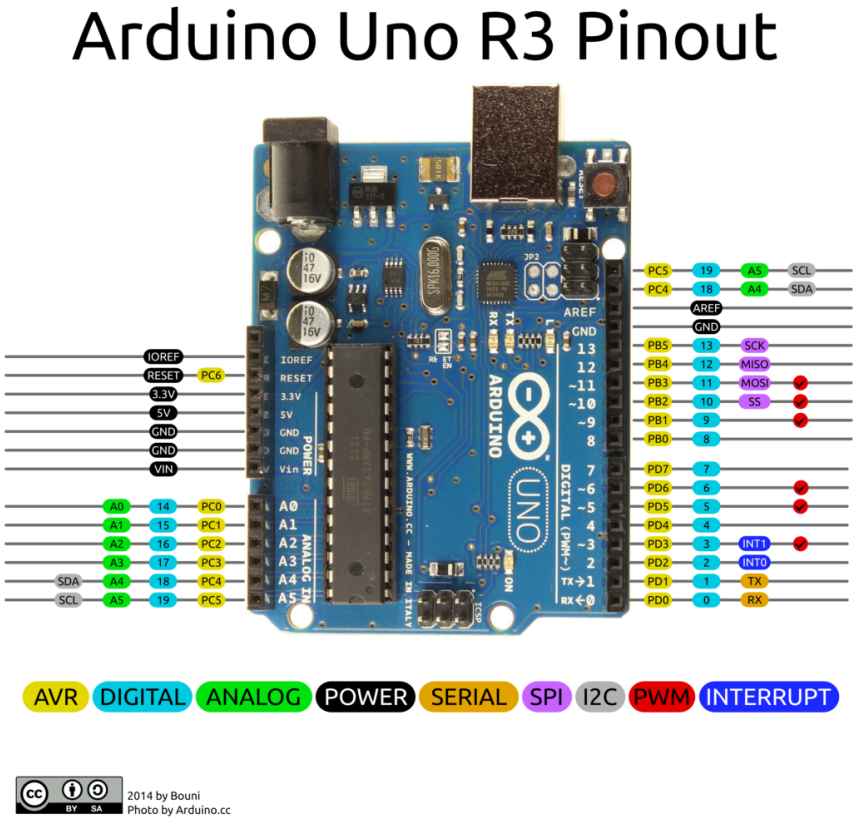


Figure 7.1: Pin Diagram of Arduino UNO

7.1 Connecting and Configuring the Board

1. Connect the Arduino UNO board to the computer using a USB cable.

2. Open Arduino IDE.

3. In Tools Menu, select Board →Arduino/Genuino UNO and Port as the available serial port to which Arduino is connected.

4. The serial port can be identified by:
   For Linux:
   Type the command **ls -l /dev/ttyACM*** in the terminal and the output it returns gives the port to which Arduino is connected, for example **/dev/ttyACM0**.
   For Windows:
   Open **Device Manager** and check for the name of connected port which will be in the form, for example, **COM5**.

5. Click Sketch →Include libraries and select Manage libraries, type AutoPID in the search bar and install the library.

6. Click File →Open and browse OpenModelicaEmbedded →Firmware →Arduino →pidmata3 and open pidmata3.ino.
   OR
   Open StandardFirmata sketch: File →Examples →StandardFirmata , if not working with PID.

7. Upload the sketch to the board.

7.2 Interfacing with OpenModelica

1. Upload StandardFirmata sketch to the Arduino board.

2. Open package.mo from OpenModelicaEmbedded package, also open package.mo file from Modelica_DeviceDrivers library.

3. In OpenModelicaEmbedded package, open ArduinoExamples package.

4. In Diagram view, change the port name for the board component to the port to which board is connected by double-clicking on it.

5. Simulate the example model.

7.3 Examples for Arduino

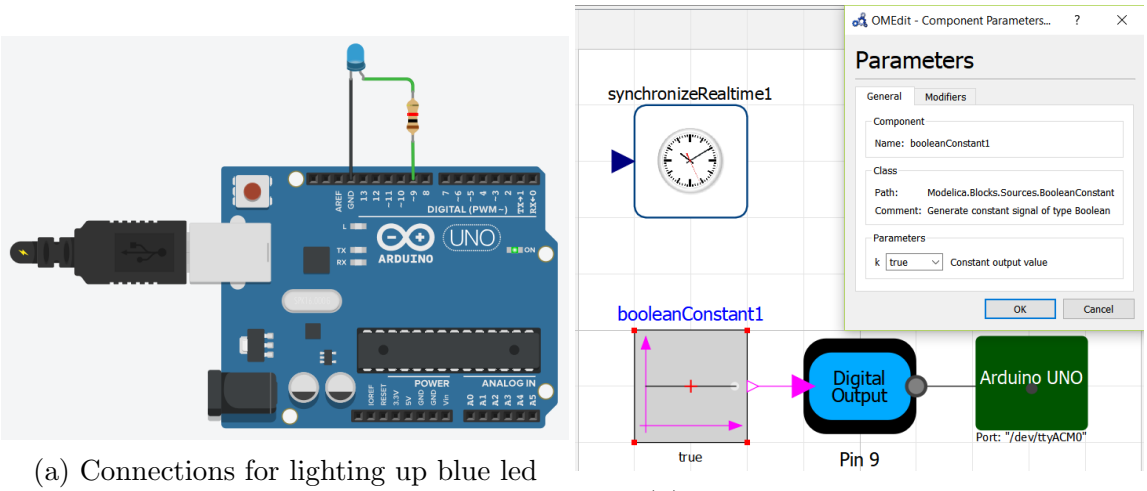
ArduinoExamples package consists of example models designed for specificly to work with Arduino UNO board. In order to work with these examples, double-click of board block in the Diagram view and change port name to the port to which board

is connected. If working with any other similar Arduino board, double-click the pin blocks and change pin number as per the used board.

7.3.1 LED Examples

Example: arduino_ex1_led_blue

The following is an example to turn on the blue led indefinitely. Double clicking each block opens the parameter window for it. Change the parameters according to the following image.



(a) Connections for lighting up blue led

(b) Model for lighting up blue led

Figure 7.2: Arduino Led Example

7.3.2 Push Button Examples

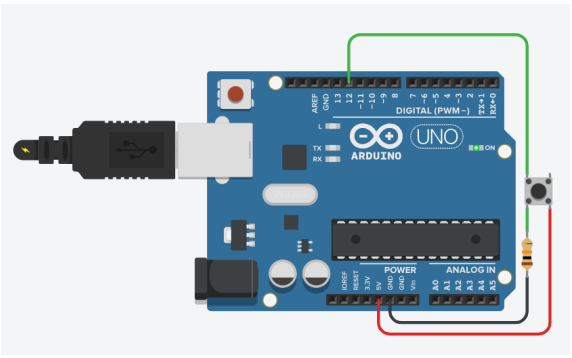
Example: arduino_ex1_push_button_status

The following example is to read the status of the pushbutton and display it on the serial monitor.

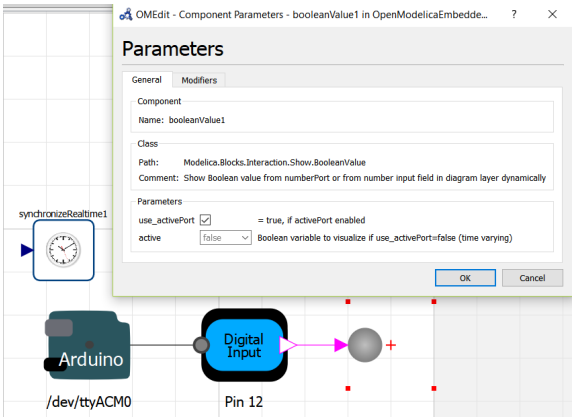
In this model, a BooleanValue block is used to show boolean value coming from the digital input pin of arduino on Simulation Output. The block BooleanValue can be found at Modelica.Blocks.Interaction.Show.BooleanValue.

Moreover, a print statement to print boolean value is written in the Text view of the model which can be seen in 7.4.

Double clicking each block opens the parameter window for it. Change the parameters according to the following image.



(a) Connections for push button



(b) Model for checking status of push button

Figure 7.3: Arduino Push Button Example

```
model arduino_ex1_push_button_status
  extends Modelica.Icons.Example;
  Modelica_DeviceDrivers.Blocks.OperatingSystem.SynchronizeRealtime synchronizeRealtime1 annotation( ...);
  OpenModelicaEmbedded.Boards.Arduino arduino1(Port = "/dev/ttyACM0") annotation( ...);
  OpenModelicaEmbedded.Pins.DigitalInput digitalInput1(Pin = 12) annotation( ...);
  Modelica.Blocks.Interaction.Show.BooleanValue booleanValue1 annotation( ...);
equation
connect(digitalInput1.y, booleanValue1.activePort) annotation( ...);
connect(arduino1.boardConnector, digitalInput1.pinConnector) annotation( ...);
  Modelica.Utilities.Streams.print(String(booleanValue1.activePort));
end arduino_ex1_push_button_status;
```

Figure 7.4: Print Statement

7.3.3 LDR Examples

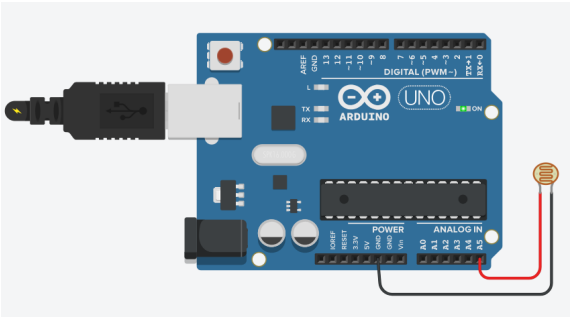
Example: arduino_ex2_ldr_read

Turning the blue LED on and off according to the values of LDR (Light Dependent Resistor).

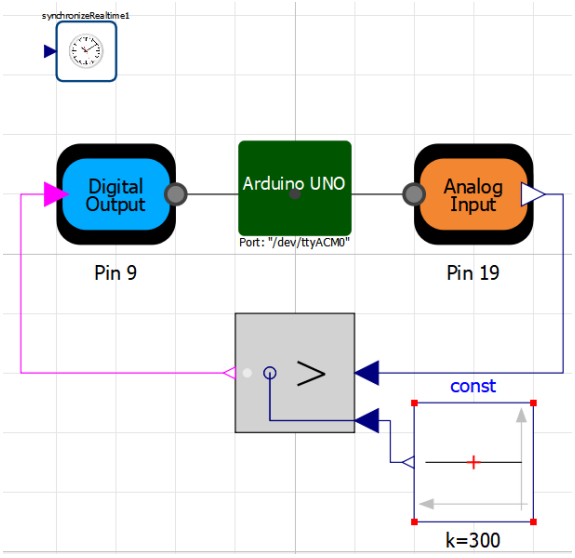
In this model, 2 blocks have been used namely Less and Constant.

Less block takes 2 input and gives 1 output according to the values of input. For example in the case below, when value from pin 19 is less than k = 300, output is true (or 1) and when value from pin 19 is greater than equal to k=300, its output is false (or 0). The block can be found at Modelica.Blocks.Logical.Less.

Constant block provides with a constant value which can be set by user. The block can be found at Modelica.Blocks.Sources.Constant.



(a) Connections for ldr



(b) Model for switching led depending on ldr value

Figure 7.5: Arduino LDR Example

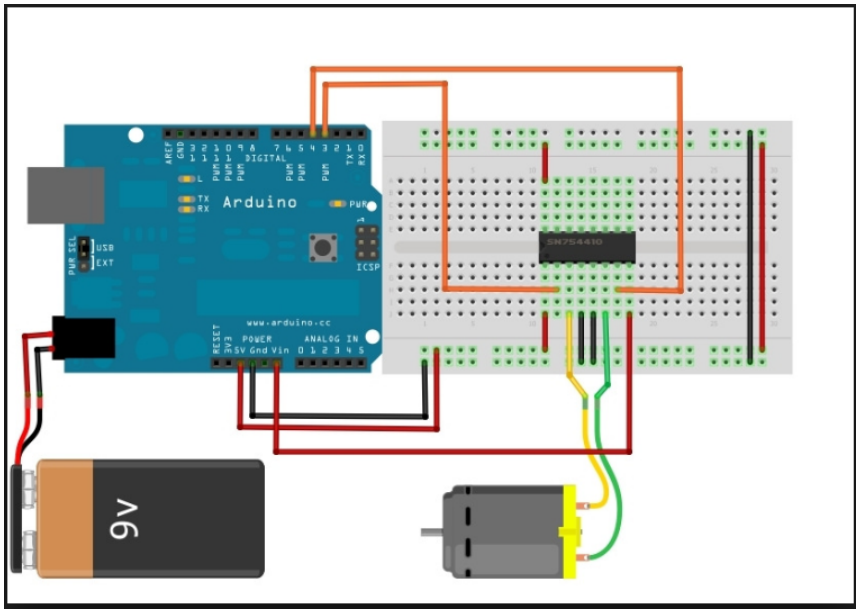
7.3.4 DC Motor Examples

Example: arduino_ex2_dcmotor_both

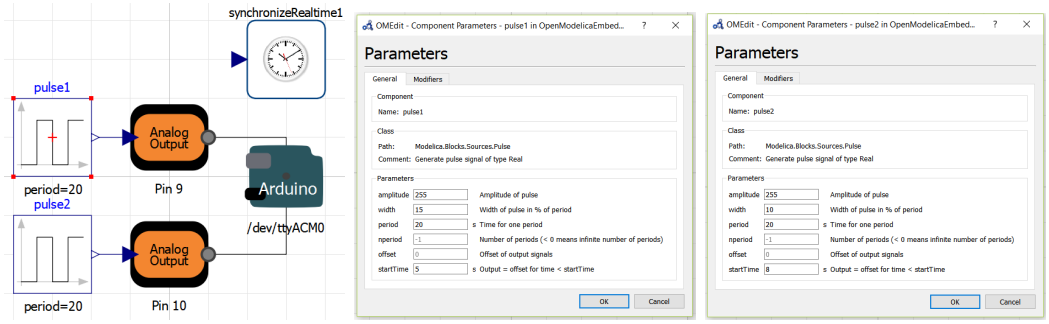
The following example is of rotating the DC motor in both directions. As visible in the model below, 2 pulse blocks are used to manage this.

A pulse block generates pulse signals of real value. Its amplitude, duty cycle, time period, start time can be varied through changing amplitude, width, period, startTime respectively in the parameter window of the pulse. The block can be found at Modelica.Blocks.Sources.Pulse.

Double clicking each block opens the parameter window for it. Change the parameters according to the following image.



(a) Connections for dc motor



(b) Model for rotating dc motor in both directions

Figure 7.6: Arduino DC Motor Example

7.3.5 Potentiometer Examples

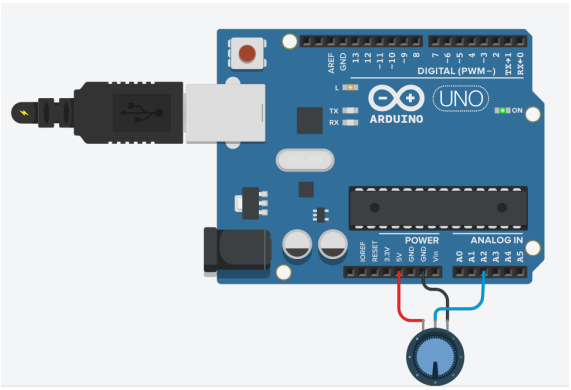
Example: arduino_ex1_pot_threshold

The following example is of turning on LEDs depending on the potentiometer threshold. As seen in the model below, weve used Xor and GreaterEqualThreshold blocks.

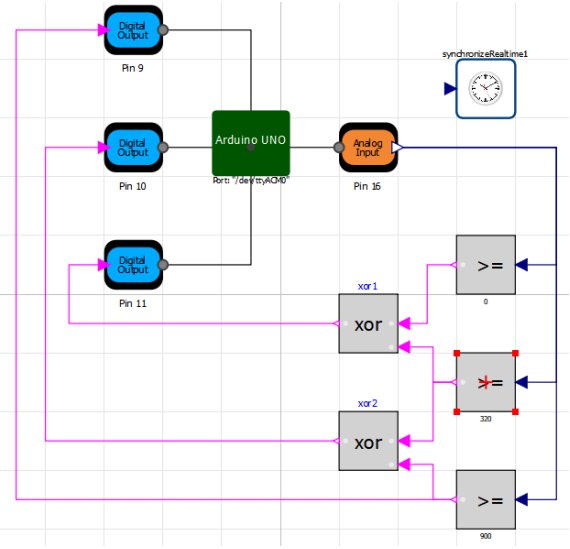
The GreaterEqualThreshold block has a parameter-threshold; if the input value to the block is greater than or equal to this threshold then the output is the same as input, otherwise 0.

This block can be found at Modelica.Blocks.Logical.GreaterEqualThreshold.

The other one is Xor block which simply xors ihe 2 input values it gets. This block can be found at Modelica.Blocks.Logical.Xor.



(a) Connections for potentiometer



(b) Model for potentiometer threshold ranges

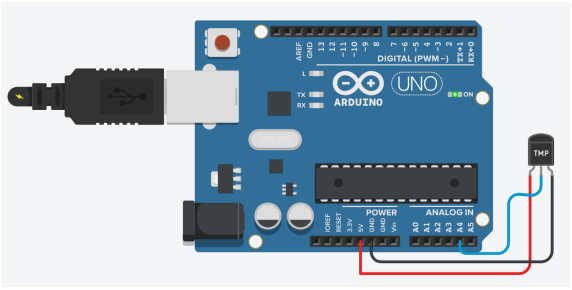
Figure 7.7: Arduino Potentiometer Example

7.3.6 Thermistor Examples

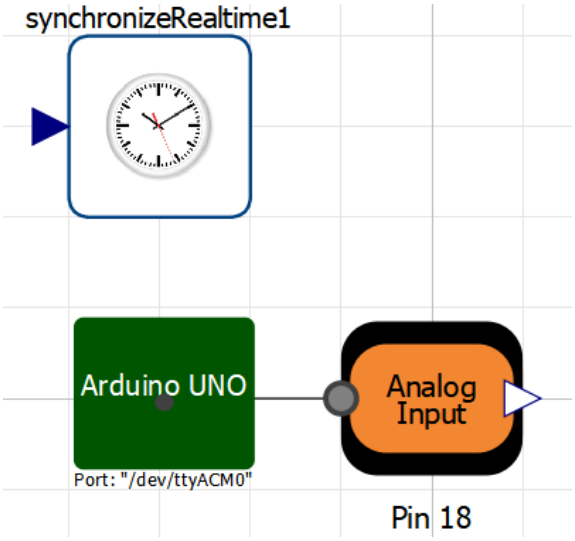
Example: arduino_ex1_therm_read

Turning the buzzer on and off using the thermistor values read by ADC.

Its model has Greater block. Greater block takes 2 input and gives 1 output according to the values of input. For example in the case below, when value from pin 18 is greater than k = 500, output is true(or 1) and when value from pin 18 is less than equal to k=500, its output is false(or 0). The block can be found at Modelica.Blocks.Logical.Greater.



(a) Connections for thermistor



(b) Model for checking value of thermistor

Figure 7.8: Arduino Thermistor Example

7.3.7 Servo Motor Examples

Example: arduino_ex3_servo_loop

Rotating the servo in increments.

The model contains blocks like Product, RealToInteger, IntegerToReal, Constant and Ramp. The Ramp block gives a strictly increasing value. On using RealToInteger block on the output, it converts it to step function. Now as the Product block accepts 2 input in real format only, there was a need to convert the value back to real using IntegerToReal block.

In Servo pin, set InputUnit to OpenModelicaEmbedded.Internal.Types.ServoUnit.None.

As can be seen from data sheet (Figure: 7.9) SG90 has a duty cycle of 5-10% where if it is 5%, the position of motor is -90 degrees and if 10%, it is +90 degrees. So as we were simulating for 10 seconds, MinPulse was 0.5 sec and MaxPulse was 1sec in Servo pin Parameters.

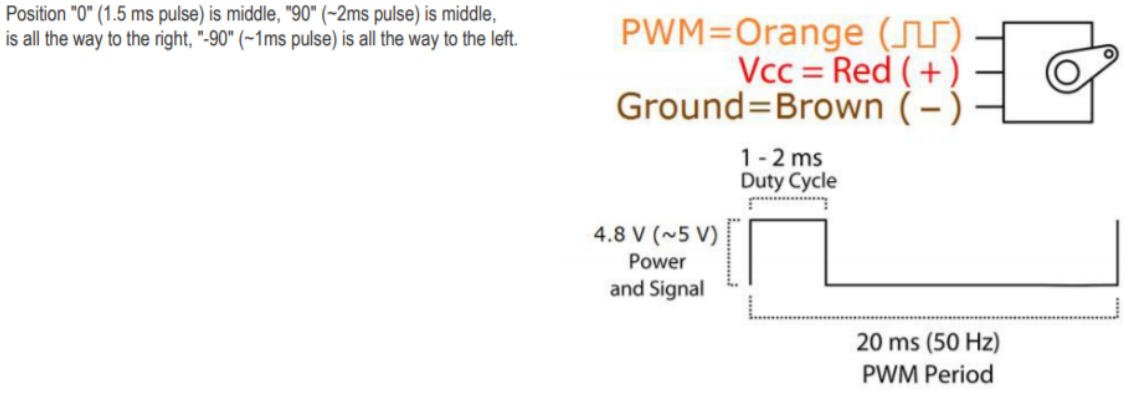
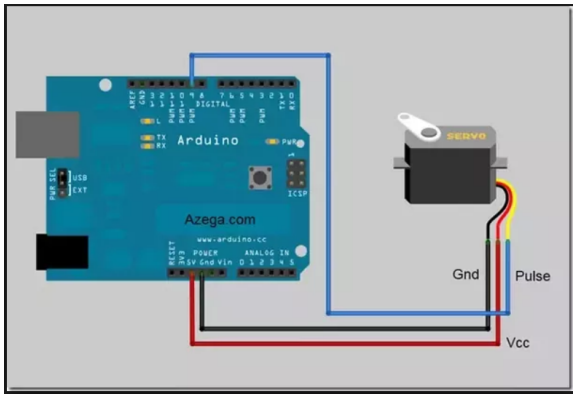
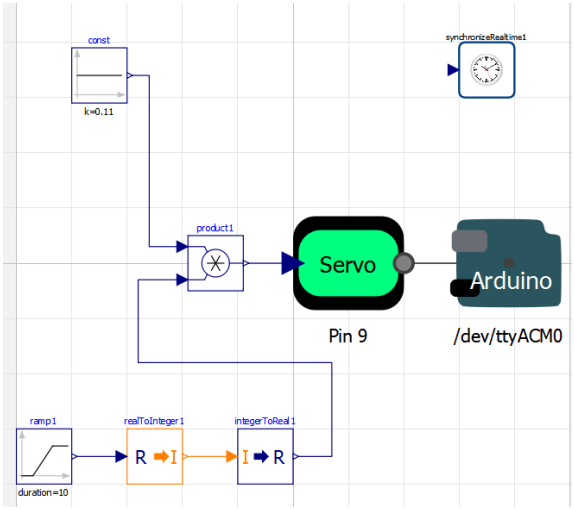


Figure 7.9: Data Sheet for Servo Motor SG90



(a) Connections for Servo Motor



(b) Model for working with Servo Motor

Figure 7.10: Arduino Servo Motor Example

Chapter 8

Working with Tiva C Launchpad

The Tiva C series Launchpad Evaluation board (EK-TM4C123GXL) is low cost ARM-Cortex-M4F based microcontroller. The board contains 40 I/O pins, two user programmable push buttons, a RGB led and many more features.

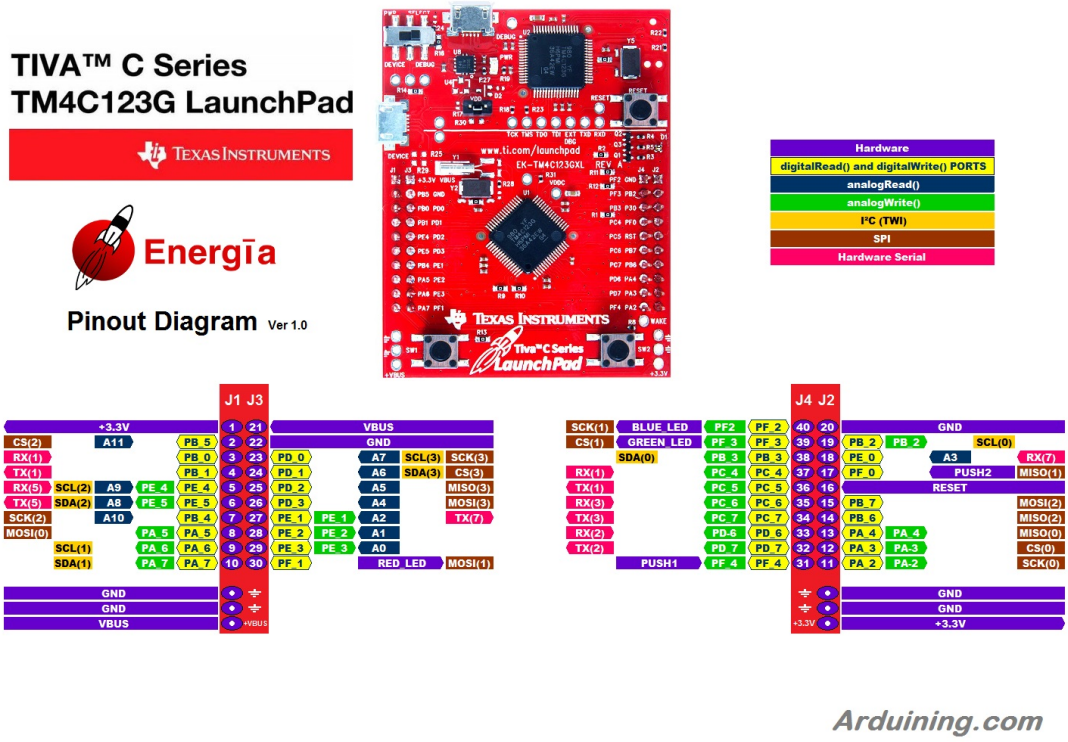


Figure 8.1: Pin Diagram of Tiva C Launchpad

8.1 Connecting and Configuring the Board

1. Connect the Tiva C board to the computer using a USB cable.
2. Open Energia IDE.

3. In Tools Menu, select Board →Tiva C and Port as the available serial port to which Arduino is connected.

4. If Tiva C board is not present, then click on Board Manager, type Tiva C in search bar and then click on Install to install board library, then apply Step 3.

5. In Sketch menu, Select Include Library →Add .zip library and add the zip file provided in the Firmware folder. Then open StandardFirmata sketch: File →Examples →StandardFirmata.
   OR
   Click File →Open and browse OpenModelicaEmbedded →Firmware →Tiva C →StandardFirmata and open StandardFiramata.ino.

6. Upload the sketch to the board.

8.2 Interfacing with OpenModelica

1. Upload StandardFirmata sketch to the Tiva C board.

2. Open package.mo from OpenModelicaEmbedded package, also open package.mo file from Modelica_DeviceDrivers library.

3. In OpenModelicaEmbedded library, open TivaC_Examples package.

4. In Diagram view, change the port name for the board component to the port to which board is connected by double-clicking on it.

5. Simulate the example model.

8.3 Examples for Tiva C

The examples explained in package for tiva-c are same as that for arduino board except that the pin configurations are different. The configuration can be seen from the pin diagram from Figure 8.1.

Example: tivac_ex1_led_blue

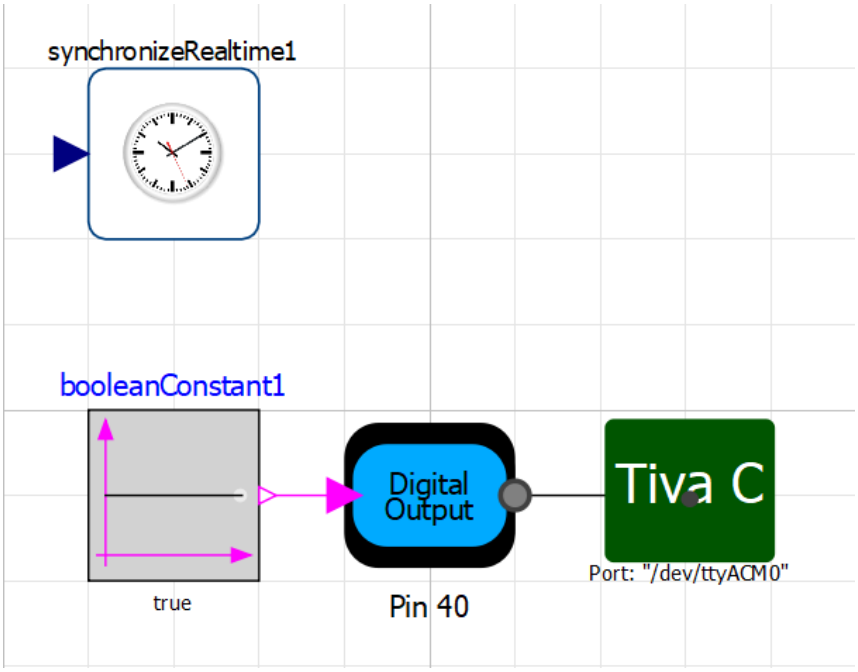


Figure 8.2: Tiva C Led Example

The example Tiva-C Led works same as the one which is explained for arduino except for the fact that the pin configuration has been changed,for arduino it was pin number 9 while for tiva-c board it is pin number 40 as can be reffered from the pin diagram of tiva-c board (Figure: 8.1). All other working remains same as already explained in case of Arduino.

For all other models also only pin number changes in case of Arduino rest they are similar to those present in ArduinoExamples package.

Chapter 9

Conclusion

The project "Interfacing of Embedded Systems with OpenModelica", is based on implementing an example package for Arduino board as well as for tiva-C board.We implemented the same set of examples on both Arduino board as well as Tiva-c board because Tiva-c board targets industries.We implemented hardware in loop simulation and PID tuning was done. Indeed the same set of codes can be used for linux as well as for windows plateform. We also explored the embedded targets package of the Modelica_DeviceDrivers library.

Although, there were many issues initially, most of them got resolved in the course of the project. While working on the project with OpenModelica we came to a conclusion that OpenModelica is an open source software based on Modelica language to design and simulate complex physical systems through code as well as graphical blocks which is also very useful for electronics prototyping and real time simulations. The main drawback of the library is its lack of appropriate documentation and various other hardware supports in the electronics hardware area. Such modules are open for modifications and can be extended by future developers. Therefore we have explored OpenModelica in detail and tried to provide a better insight in this open source software which will help developers in the future.

Reference

The following sources were referred to while working on this project:

- Peter Fritzson :Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach
- https://www.openmodelica.org/
- http://book.xogeny.com/
- https://build.openmodelica.org/Documentation/Modelica.html
- https://www.codeproject.com/Articles/84461/MinGW-Static-and-Dynamic-Libraries
- https://stackoverflow.com/questions/10039401/use-32bit-shared-library-from-64b
- https://stackoverflow.com/questions/142508/how-do-i-check-os-with-a-preprocess
- https://en.wikipedia.org/wiki/PID_controller
- https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/interop_c_python.html