FOSSEE SUMMER INTERNSHIP 2017
IIT BOMBAY

INTERNSHIP REPORT

Interfacing Arduino with OpenModelica

Developers
Souradip Pal
Vanessa Singh
Disha Agarwal

Mentor
Manas Ranjan Das

Academic Supervisor
Prof. Kannan M.Moudgalaya

Acknowledgement

We are grateful to FOSSEE, IIT Bombay and our mentor Mr. Manas Ranjan Das for giving us the opportunity to work on the project Interfacing Arduino with OpenModelica. We express our sincere gratitude to our mentor Mr. Manas Ranjan Das for his support, help and encouragement throughout the project.

I would also like to thank Professor Kannan M.Moudgalya who introduced us to FOSSEE, its projects and the significance of its work. Also I would like to recognize the support and contribution of all the team members.

TABLE OF CONTENTS

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 tries to interface it with Arduino by calling C functions from OpenModelica. Modelica functions are written in OpenModelica and they call C functions which give instructions to Arduino. These codes can be run to perform operations on dc motor, servo motor, led, ldr (light dependent resistor), thermistor and potentiometer connected externally to the board. Moreover we have also used Modelica_DeviceDrivers library which enables the use of graphical blocks in OpenModelica for graphical coding of the above operations.

In this project, we have developed the libraries 'OpenModelica-Arduino' and 'OpenModelica-Arduino-Windows' which enable the interfacing of Arduino with OpenModelica in Linux and Windows respectively. Thus using this library, we merge the functionalities of Arduino and OpenModelica for faster data processing and data visualisation of real-time simulations.

2. Abstract

Growing use of electronic products and automation increases need for softwares that can be used to program microcontrollers easily. The most used basic open source electronic development boards or prototyping platforms are Arduino platforms which are based on AVR microcontrollers (except the ones which are based on ARM microprocessor). Although, Arduino can be easily coded using Arduino software ide, python, Scilab and Julia but still softwares that enable graphical coding of Arduino operations are very few. OpenModelica is an open source software based on Modelica modeling language for complex physical systems, has support for line by line as well as graphical coding. Data visualization becomes easier and faster through the software. Also, it can interact with languages like C and python and thus, it can be easily interfaced with Arduino. Moreover, Modelica_DeviceDrivers library, which contains drag and drop blocks for arduino and can be used inside OpenModelica, enables GUI support for programing of arduino. This project interfaces OpenModelica with Arduino through C code as well as explores Modelica_DeviceDrivers library.

3. Downloading and Installing Arduino IDE

Arduino development environment is compatible with popular desktop operating systems. In this section, we will learn to set up this tool for the computers running Microsoft Windows or Linux. Later, we shall explore the important menu options in the Arduino IDE.
For both Windows and Linux go to
https://www.arduino.cc/en/Main/Software and follow the instructions to complete the setup.

4. Downloading and Installing OpenModelica

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

Windows:

The setup file can be downloaded from
https://openmodelica.org/download/download-windows.Download the latest release or the alpha version (preferred) of the software. After downloading, run the installation wizard to complete the installation.
OpenModelica Connection Editor or **OMEdit** can be launched by using the desktop shortcut or OMEdit icon.

Linux:

The Debian/Ubuntu package can be downloaded from
https://openmodelica.org/download/download-linux. Follow the installation instructions given on the site for more details. OMEdit can be launched by typing OMEdit in the terminal.

5. Connecting and Configuring Arduino UNO Board

Following two steps have to be followed whatever operating system is used:

1. To begin, we need an Arduino Uno board with a USB cable.
2. Connect it to a computer and power it up.

Windows:

Attach the Arduino UNO Board and go to Device Manager->Ports (COM & LPT) ->Arduino UNO (COM2) (Click on it) ->Port Settings -> Advanced->COM "Port Number". Change this to COM2 and click OK. (Any other port can also be set but remember to change to that port wherever COM2 is mentioned in this document.)

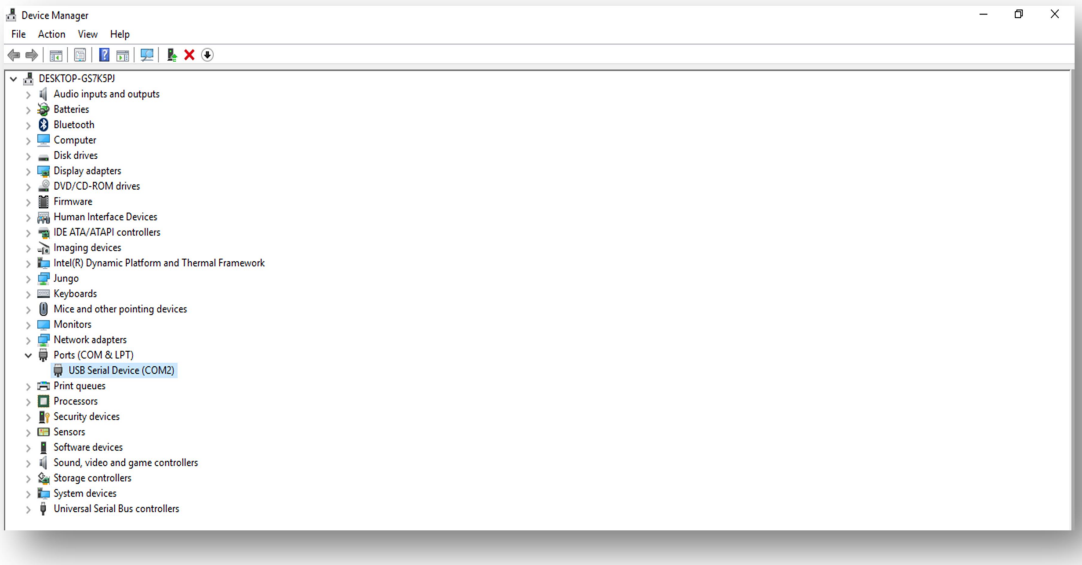


Fig 1.Device Manager in Windows

Linux:

Type the command **ls -l /dev/ttyACM*** in the terminal and if it returns ACM0 then the port to which Arduino is connected to is 0.

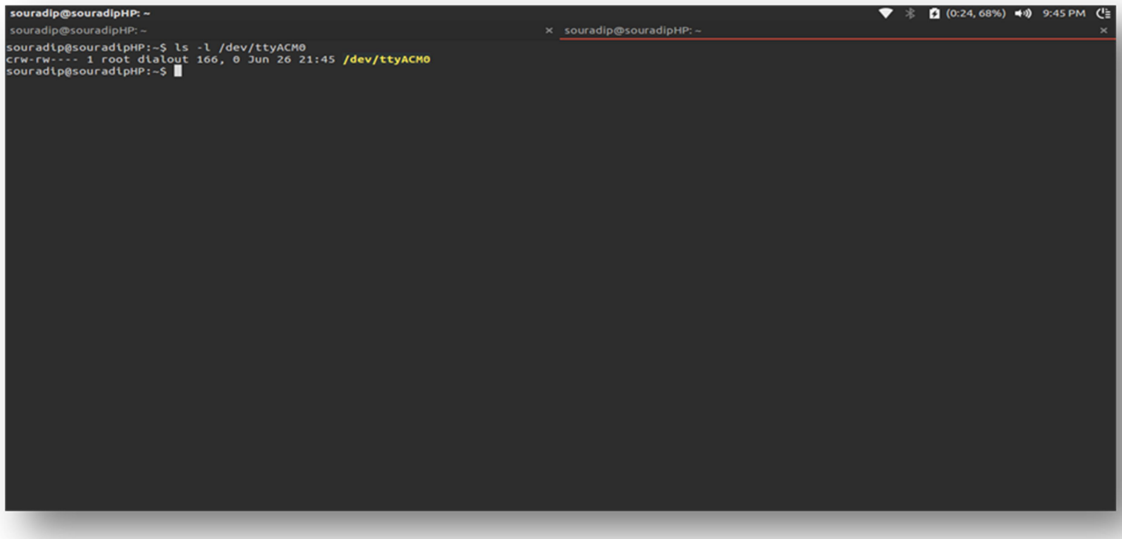


Fig 2.Checking port in Linux

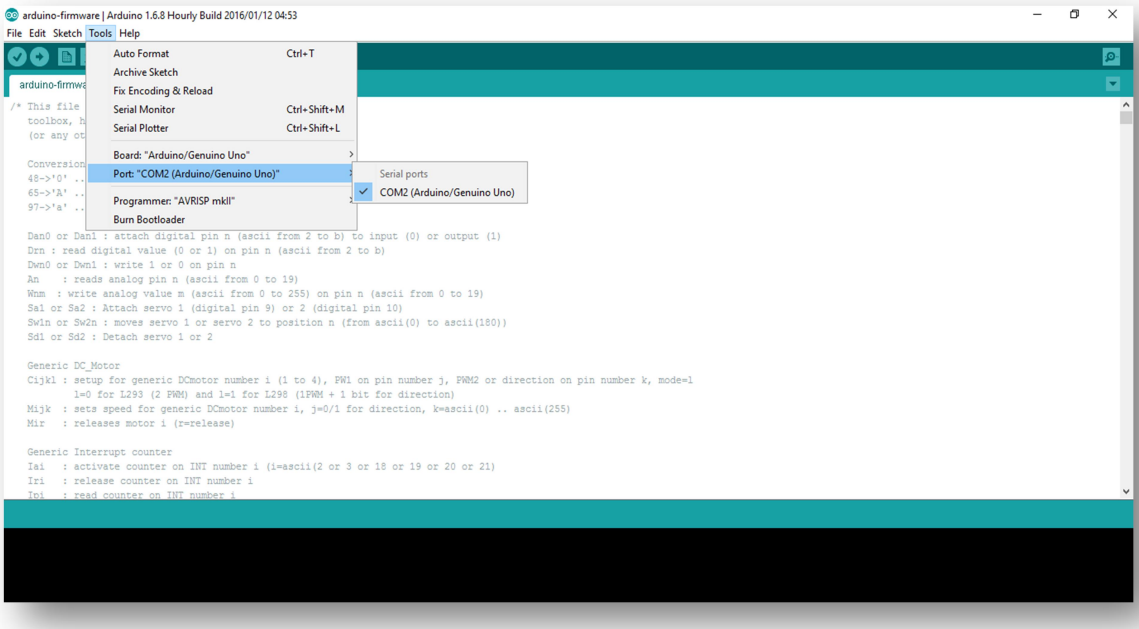


Fig3. Arduino IDE port settings

Click on File->Open and browse to the 'path_to_OpenModelica-Arduino' -> tools -> arduino-firmware and select Arduino-firmware.ino and open it. Go to tools and select the board as Arduino UNO and Port as the port no. to which the Arduino is attached. Then click on upload and upload the code.

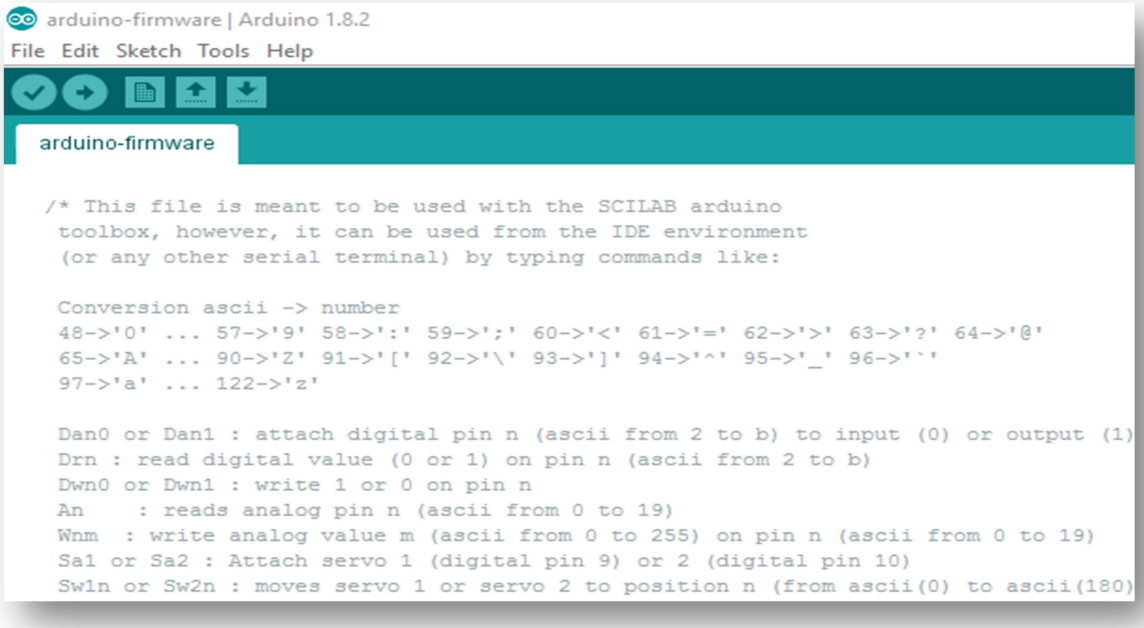


Fig 4.Arduino-Firmware

6. Interfacing Arduino with OpenModelica

OpenModelica supports the calling of external C functions and that is extensively used for this interfacing process. OpenModelica-Arduino library can be downloaded from GitHub from the following links:

Windows:
https://github.com/VanessaSingh/OpenModelica-Arduino-Windows.git

Linux:
https://github.com/Souradip-sopho/OpenModelica-Arduino.git

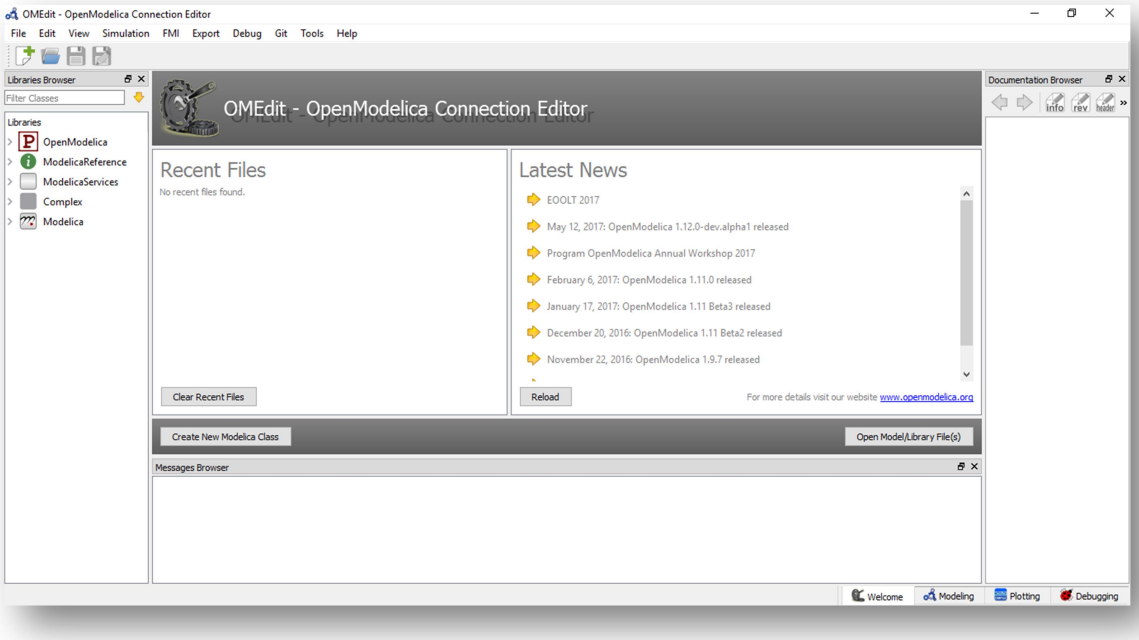


Fig 5.OMEdit

6.1 Interfacing in Windows

1. Open OpenModelica Connection Editor or OMEdit (You can either create a shortcut or find it in the search bar). Go to
   File ->Open Model and browse to **Arduino.mo** in the OpenModelica-Arduino-Windows library and load it.
   Open the Arduino IDE and upload **arduino-firmware.ino** program into the arduino board.
2. Now load **testfirmware.mo** file for testing the arduino firmware. (Refer Step 1).
3. Change the port in the open_serial function of the code in the models according to port id to which the arduino is connected.
   For example:
   If arduino is connected to port 'COM2'
   Change
   ok := sComm.open_serial(1, 0, 115200);
   to
   ok := sComm.open_serial(1, 2, 115200);
4. Click on the green arrow to simulate the example.

5. The simulation settings can be done by clicking on the 'S' symbol (Fig 5). If serial communication is successful, status shown will be zero.
   If it doesn't give any error then the firmware is loaded correctly. Similarly, any example model provided in Arduino.mo can be run.

The source code can be found in the '**src**' directory, the header file can be found in the Include directory and the shared object file can be found in the Library directory. (Fig 6)
Commands to make a shared object file:

Open command prompt and go to path of OpenModelica-Arduino-Windows->Resources->src.

After that type the following commands:

gcc –c SerialComm.c (Makes a SerialComm.o file in the same folder as SerialComm.c)

gcc –shared –o ../../SerialComm.dll SerialComm.o (Makes a dynamic link library from the SerialComm.o file in OpenModelica(Windows))

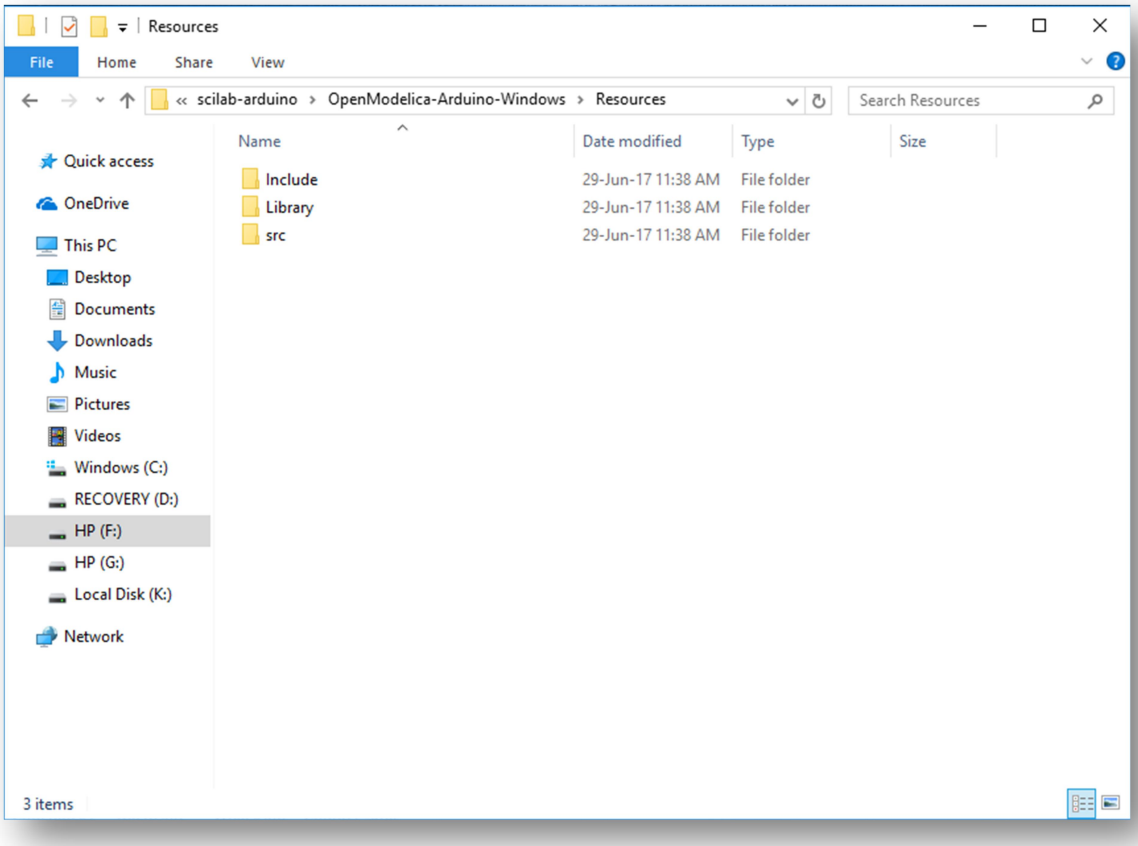


Fig 6. Directory Structure of OpenModelica

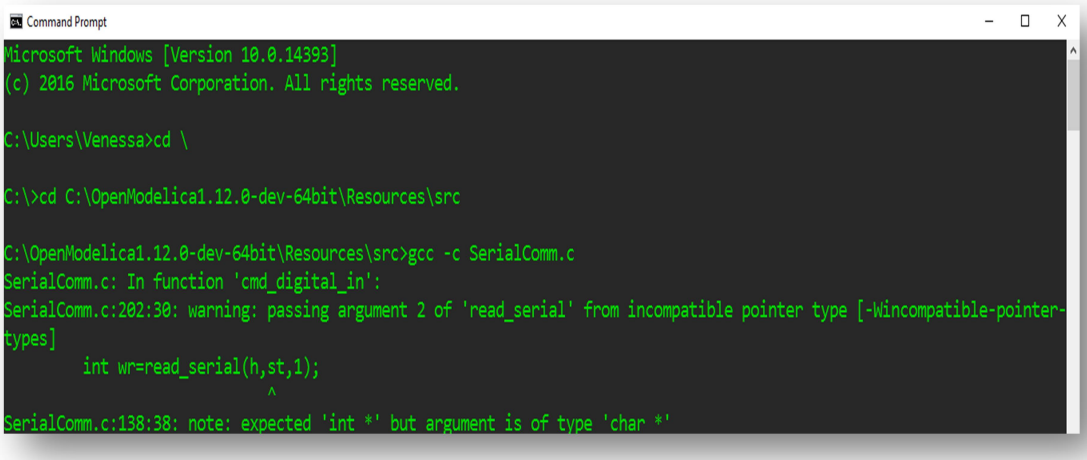


Fig 7. Compiling SerialComm.c on Windows

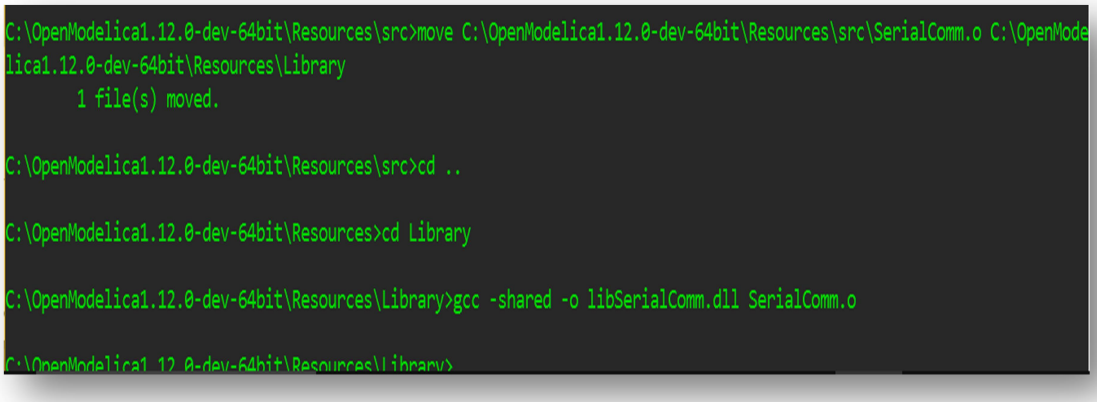


Fig 8. Making a shared object file on Windows

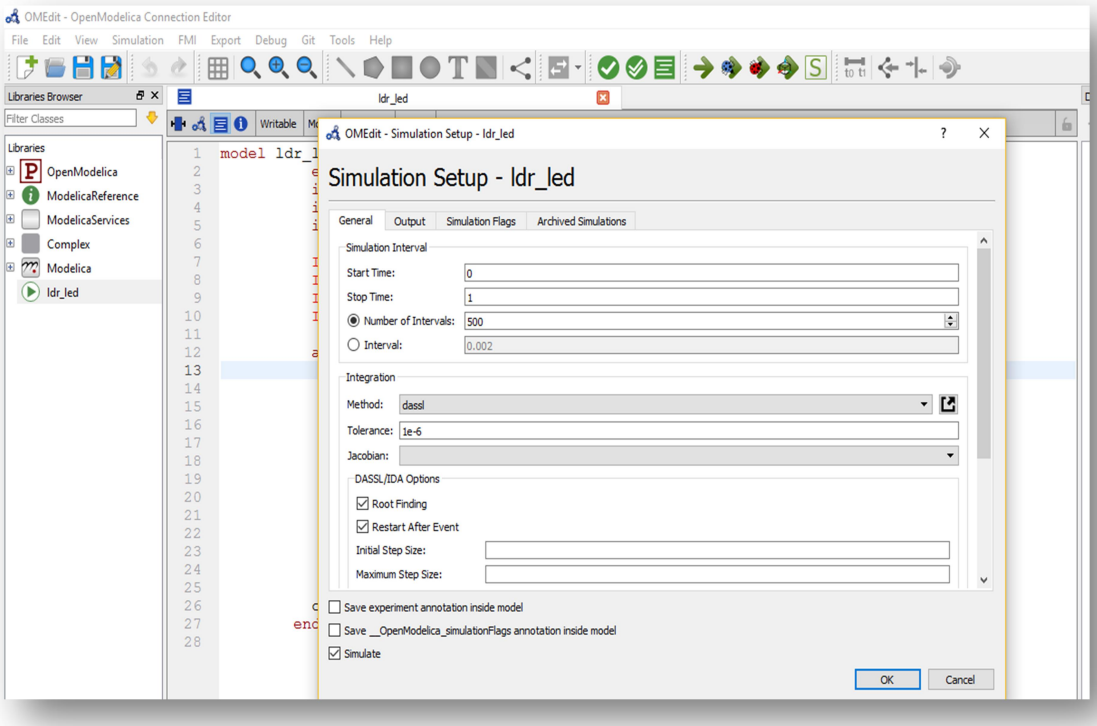


Fig 9. Simulation Setup

6.2 Interfacing in Linux

1. Open OpenModelica Connection Editor or OMEdit by executing OMEdit in terminal. Go to File -> Open Library and browse to **Arduino.mo** from the OpenModelica-Arduino library and load it.
Open the Arduino IDE and upload **arduino-firmware.ino**.

2. Now load **testfirmware.mo** file for testing the arduino firmware. (Refer Step 1)
3. Change the port as necessary in the example.
   For example:
   If the port is '/dev/ttyACM2' make
   ok := sComm.open_serial(1, 0, 115200);
   to
   ok := sComm.open_serial(1, 2, 115200);
4. Click on the green arrow to simulate the example.
5. The simulation settings can be by clicking on the 'S' symbol (Fig 5). If serial communication is successful, status shown will be zero.If it doesn't give any error then the firmware is loaded correctly. Similarly, any example model from Arduino.mo can be run.

The source code can be found in the src directory, the header files can be found in the Include directory and the shared object files can be found in the Library directory.(Fig 6)

Commands to make a shared object file:

Open command line and go the path of OpenModelica-Arduino -> Resources ->Library.
After that type the following commands:

gcc –c –Wall –fPIC ../src/serialComm.c (Makes a serialComm.o file in the same folder as SerialComm.c)

gcc –shared –fPIC SerialComm.o –o libSerialComm.so (Make a shared object from the SerialComm.o file)

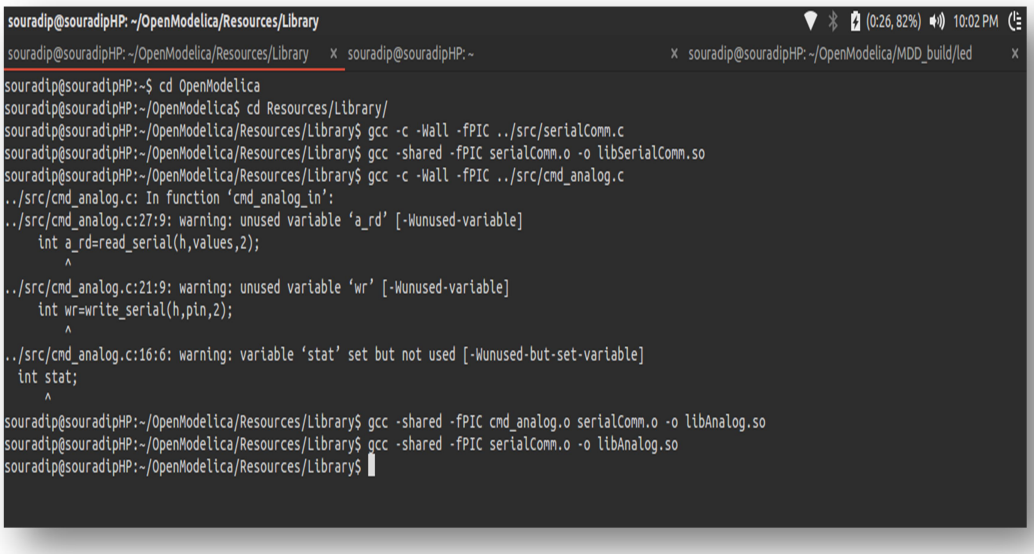


Fig 10.Compiling serialComm.c on Linux

NOTE:

The port to which Arduino was attached was COM2 therefore 2 is passed in the open_serial function of the user codes. (In Linux it was attached to port 0 therefore by default 0 is passed in the open_serial function.)

7. How is it created?

The OpenModelica-Arduino package is based on serial communication with Arduino using UART protocol. Basic idea behind serial communication with Arduino is to configure the port where the Arduino board is connected to PC using USB cable and identifying the port. The information is therefore used in establishing serial communication route with Arduino and OpenModelica software in the system. All the configurations of the serial port are done using external C functions which can be called by OpenModelica.

7.1 Help from Scilab

The source code for OpenModelica-Arduino interfacing is mostly based on the idea of establishing serial communication with Arduino as done in **Scilab-Arduino Toolbox**. The same function call structure has been implemented here. The five basic functionalities required in this case are: open_serial, close_serial, read_serial, write_serial & status_serial. These functions allow serial communication with the Arduino platform and used in other interfacing functions for establishing communication.

Before using these functions, the Arduino platform must be loaded with a firmware program present in '**Arduino-firware.ino**' file. This program must be uploaded in Arduino board before the start of interfacing. This program contains specific set of identifiers to recognize instructions sent through the serial port.

7.2. Functions:-

Basic Functions:

- **open_serial**- The function 'open_serial' takes in parameters an integer handle and port number on which arduino is attached and baud rate at which it has to communicate with arduino. The function opens the serial port (a file descriptor) and returns 0 if serial port is successfully opened and in case of a bad file descriptor/failure to open serial port returns the integer 2. It also calls function 'set_interface_attribs' to set the baud rate and other attributes of the serial port interface and the function 'set_blocking' to disable blocking(no blocking/0).

- **close_serial**- The function 'close_serial' takes in handle to the serial port as argument. The function closes the serial port (file descriptor) and returns 0. If the port closes successfully then a success message is printed else not.
- **read_serial**- The function 'read_serial' takes in parameters handle, a character array that will return the characters read from the file identified by handle and the number of characters/bytes to be read from the serial port. The function reads 'n' number of characters from the serial port where n is the size specified by the function caller. If read is successfully performed than the characters are copied to the input argument buffer and a 0 is returned else a integer 2 is returned by the function to denote error.
- **write_serial**- The function 'write_serial' takes in parameters handle ,character array to be written to serial port and the size of the character array. The function sends/writes the given char array to the serial port and on successful write, a message is printed else nothing is printed. The function returns a 0.
- **status_serial**- The function 'status_serial' takes the parameter handle and contains the information of the bytes of data read and written through the serial port. It returns integer 0 on success.

Interfacing Functions:

Digital:

- **cmd_digital_in**: The function 'cmd_digital_out' takes in the handle, the pin number and the value to be written as parameters. The code sends 'Da' (representing digital attach for configuring digital pins) along with ASCII value of pin number and '1'(char 1) to setup the pinMode of the corresponding pin to output. Then it converts the value to 1 if it is greater than 0.5 else converts it to 0. The code sends a character array containing 'Dw'(representing digital write),

ASCII value of the pin and the value to be written to the serial port. The firmware on receiving the values performs digitalWrite(). The function returns a 0 that is returned if the write_serial function is successful.

- **cmd_digital_out**: The function 'cmd_digital_in' takes in the handle to the serial port and pin number as input. It converts the pin number to its ASCII value and sends it to serial channel and along with char 'Da' (representing digital attach for configuring digital pins) and a '0'. This is to set the mode of the corresponding pin to input. Then it sends character array containing 'Dr'(representing digital read) and ASCII value of pin for digitalRead() to occur. After this, it reads the value received after checking the status of the serial port.

Analog:

- **cmd_analog_in**: The function 'cmd_analog_in' takes in the handle to the serial port and pin number as input. It converts the pin number to its ASCII value and sends it to serial channel and along with char 'A' denoting analog read. Then it reads the value received after checking the status of the serial port. The arduino_firmware upon receiving 'A' and the pin number performs the analogRead() and then serial write writes the value read. The value received serially is converted to decimal form and returned as the result.
- **cmd_analog_out**: The function 'cmd_analog_out' takes in the handle, the pin number and the value to be written as parameters. It converts the value to 255 if it is greater than it and converts it to 0 if the value is negative. The code sends a character array containing 'W' denoting analog write, ASCII value of the pin and the value to be written after converting it to char form. The firmware on re-

ceiving the values performs analogWrite(). The function returns a 0 that is returned if write_serial function is successful.

- **cmd_analog_in_volt** - Reads the analog value just like the cmd_analog_in function just returns the analog value converted to voltage.
- **cmd_analog_out_volt** - Writes the analog value just like the cmd_analog_out function except that the input parameter specifies the voltage which is converted to an analog value between 0 to 255 and then written.

DC Motor:

- **cmd_dcmotor_setup** - Used to initialise the motor driver and motor. It takes in the handle, driver type, motor number and pins on which the motor is attached as parameters. It sends a character array containing 'C', motor number in ASCII form, ASCII values of pins and a character '1' or '0' indicating the driver type to the serial port. The firmware code performs the initialisation and return "OK" which is read through read_serial and if the read is successful the C code prints a success message on screen else a failure message. The function doesn't return anything.
- **cmd_dcmotor_run** - Used to rotate the motor in the desired direction and speed. The handle, motor number and speed along with sign is passed in to the function. It decides a direction clockwise or anticlockwise depending on the sign of the value input and if the absolute value if the input is greater than 255 than it sets it to 255 else to the absolute of the ceil of the input. The function sends a character array containing 'M', motor number in ASCII form, direction and value in ASCII form over the serial port and the function returns nothing. The firmware code writes the value to one of the pins of the specified motor depending on the direction.

- **cmd_dcmotor_release** - This function stops the motor and releases it. It takes in the handle and motor number as arguments and it sends a character array comprising 'M', ASCII value of motor number and 'r' over the serial port. The function does not return anything. The firmware code writes 0 to both the pins of the motor.

Servo Motor:

- **cmd_servo_attach** - This function initialises the specified servo motor. It takes in handle to the port and servo number as input arguments. It sends a character array "Sa1" for servo number 1 or "Sa2" for servo number 2 else prints error. The firmware code initialises the servo to 0
- **cmd_servo_move** - This function is to move the servo to the desired angle. It takes in handle, servo number and value to be written to servo motor as the input parameters. If the value is greater than 180, then it is made as 180 and if it is lesser than 0, then it is made as 0. The function writes a character array consisting of 'Sw', ASCII value of servo number and ASCII value of input angle to the serial port. The firmware writes the angle to the servo motor. The function does not return anything.
- **cmd_servo_detach** - This function is to detach the specified servo motor. It takes in handle to the port and servo number as input arguments. It sends a character array "Sd1" for servo number 1 or "Sd2" for servo number 2 else prints error. The firmware code detaches the servo. The function does not return anything.

Auxilliary Functions:

- **math_floor** - This function takes a real number as argument and returns the greatest integer lesser than or equal to the argument provided.

- **ieeesingle2num**-This is ieee745 floating point converter which takes an integer in hexadecimal format and returns a number in decimal format.
- **delay**-This function provides delay in milliseconds for a certain time as mentioned in its argument.

7.3. Creating OpenModelica Package:

All the functions are called from within OpenModelica from a functions package created within the Arduino Package.

Creating a New Package:

To create a new package, click on New Modelica Class from the File menu bar and select the specialization 'Package' from the drop down menu. Add a name to the package and click on Ok to create a new package. Additionally, the package can be made to extend any other packages or can be inserted into any other package by selecting the optional parameters.

Here, Arduino package contains Serial Communication package under which all the functions and examples are present.

7.4. Compiling Modified Source and Creating a Shared Library

The src directory present in the Resources directory of OpenModelica library contains the source code written in C for the various OpenModelica functions.

Compilation of the source:-

Windows:

Commands to make a shared object file:

Open command line and go the path of OpenModelica->Resources->src. After that type the following commands:

gcc –c SerialComm.c (Makes a SerialComm.o file in the same folder as SerialComm.c)

gcc –shared –o ../../libSerialComm.dll SerialComm.o (Makes a dynamic link library from the SerialComm.o file in OpenModelica(Windows))

In general, for other sources, compilation can be done as follows:

gcc –c *path_to_source_file*

gcc –shared serialComm.o *file1.o file2.o …* -o *filename*.dll

Linux:

Navigate to the Library directory within Resources in terminal.Use the following commands to compile and create a shared library.

gcc –c -Wall –fPIC ../src/serialComm.c

gcc –shared –fPIC serialComm.o –o libSerialComm.so

Thus a shared library libSerialComm is created for the source file serialComm.c.

In general, for other sources, compilation can be done as follows:

gcc –c –Wall –fPIC *path_to_source_file*

gcc –shared –fPIC serialComm.o *file1.o file2.o …* -o lib*filename*.so

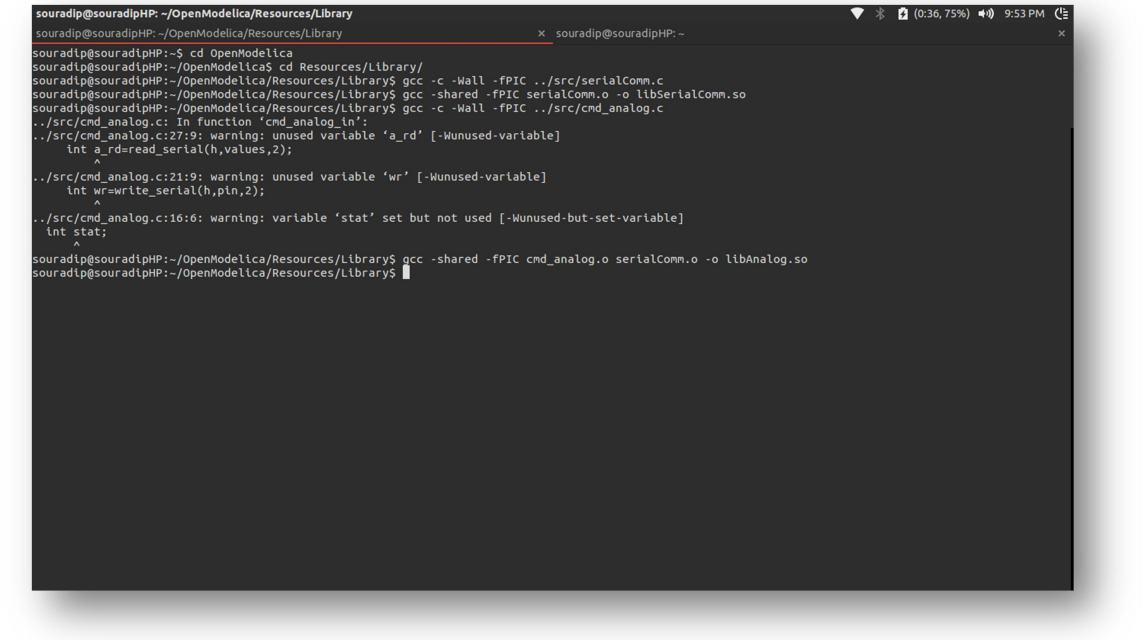


Fig 11. Making shared library file on Linux

7.5. Testing and Debugging

The procedures elaborated in section Interfacing Arduino with OpenModelica(section 6) must be followed for testing the modified source code. For debugging C source code,well known debuggers like 'gdb' can be used.

8. Interfacing Arduino with OpenModelica Using Modelica_DeviceDrivers Package

Modelica_DeviceDrivers is free library for interfacing hardware drivers to Modelica models which has support for joysticks, keyboards, UDP, TCP/IP, LCM, shared memory, AD/DA converters, serial port and other devices.

8.1 Downloading Modelica_DeviceDrivers Library

1. Download Modelica Device Drivers from https://github.com/modelica/Modelica_DeviceDrivers . Go to the Current Release section, then clone the repository using git or download the zip file from there and extract the files.
2. Open OMEdit and go to File->Load Library->Browse the path where you have extracted Modelica_DeviceDrivers and load it.

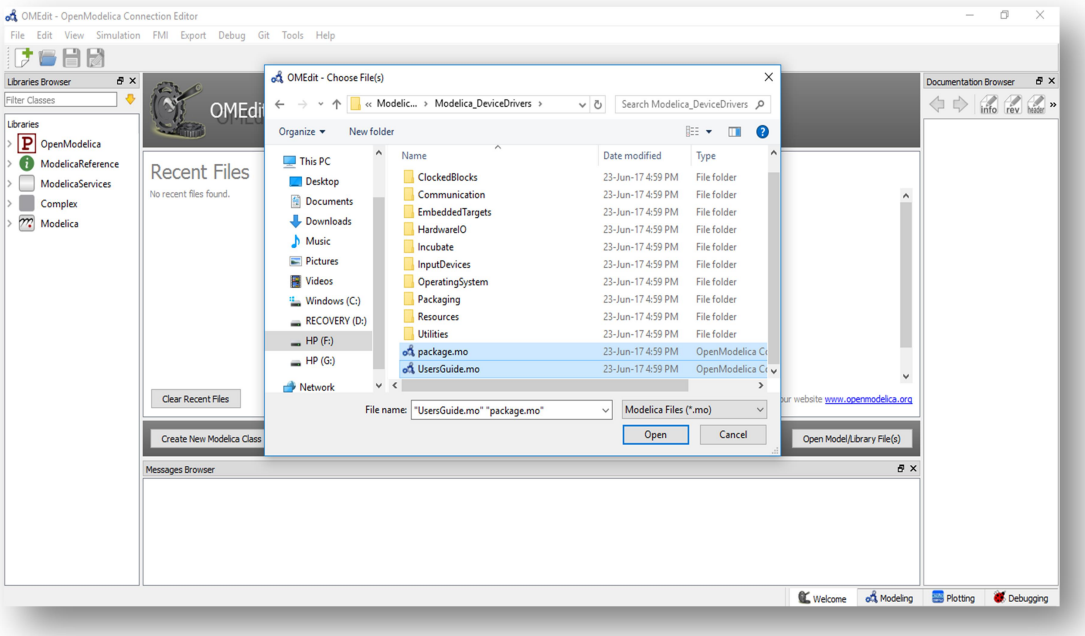


Fig 12. Loading Modelica Device Drivers

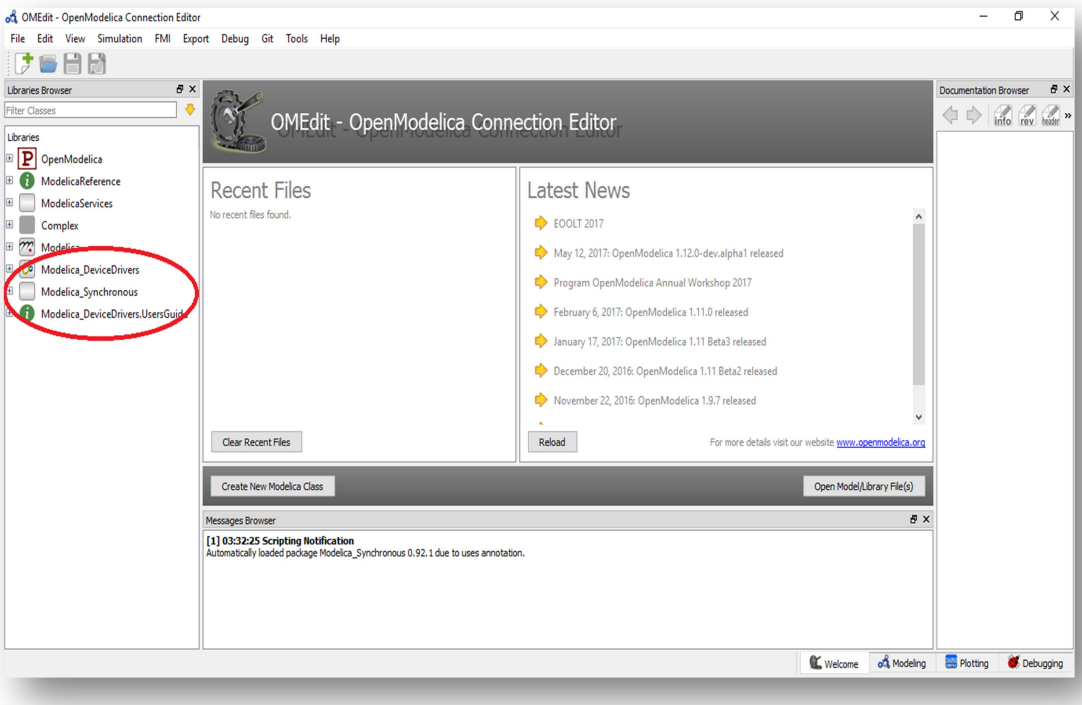


Fig 13. Loaded Modelica_DeviceDrivers package

8.2. Downloading and Installing AVR packages

We need to install **avr-gcc**, **avr-libc** and **avrdude** packages for working with Modelica_DeviceDrivers.

8.2.1 Windows

1. Go to http://andybrown.me.uk/2015/03/08/avr-gcc-492/ and install avr-gcc-4.9.2 and avr-libc-1.8.1. (Fig 5)
2. Follow the instructions till the "How to install it and use it" section (no need to go for integration with eclipse and Arduino IDE).

For easier navigation, the paths to these builds can be added to Path environment variable of the system in the following way:-
Go to System Settings ->Advanced System Settings -> Environment variables ->System Variables->Path->Edit and add **C:\avr-gcc\bin,C:\ WinAVR-20100110\bin** etc. and click OK.

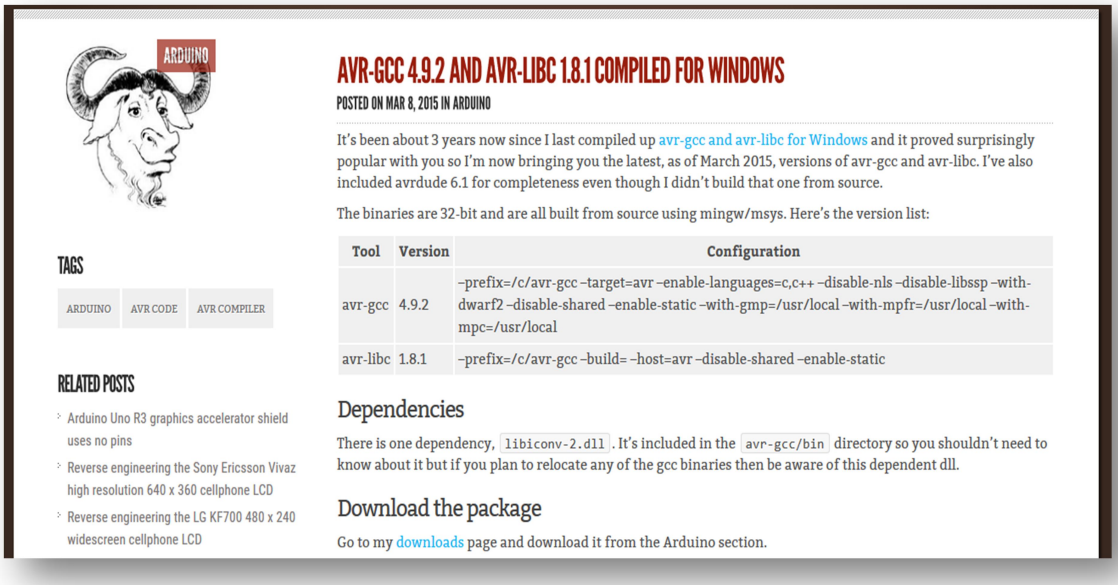


Fig 14. Downloading AVR packages for Windows

3) To install avrdude go to https://sourceforge.net/projects/winavr/ and install WinAVR- 201001101.

4) To check if the installations are done correctly open the command prompt(if it is already open, close it and open it again). Then go to C:\ and type **avr-gcc –version** if you get something like command recognized then redo the installation. If it is correct it will show you the version (Fig 15).

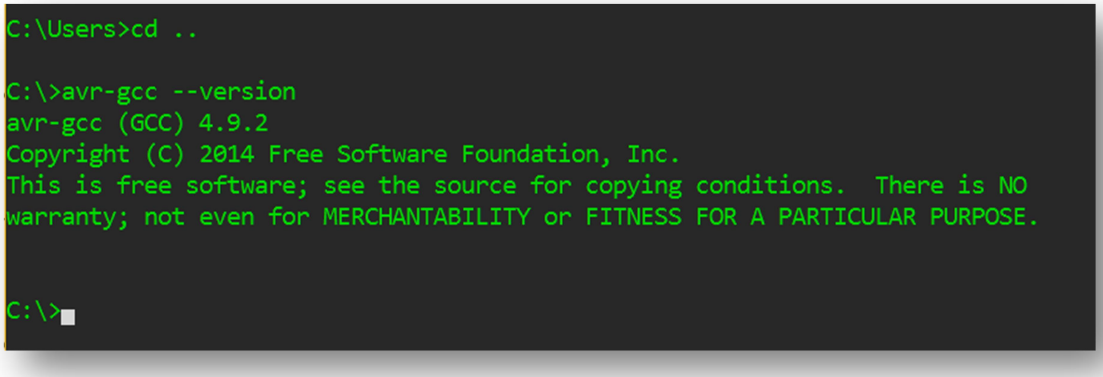


Fig 15. Checking for version of avr gcc

NOTE:

Remember to download the correct versions of avr-gcc and avr-libc (In this case avr-gcc-4.9.2 and avr-libc-1.8.1) otherwise it will give errors like "unrecognized flag std=c11".

8.2.2 Linux

Type the following commands in the terminal for the installation.

sudo apt-get install gcc-avr
sudo apt-get install avr-libc
sudo apt-get install avrdude

For manual installation go to the site :
http://maxembedded.com/2015/06/setting-up-avr-gcc-toolchain-on-linux-and-mac-os-x/
and follow the instructions given under Linux heading for installation of the above mentioned packages.

8.3 Instructions for simulation

Simulation of models created using Modelica_DeviceDrivers package are to be done in Command Prompt(Windows)/Terminal(Linux).The example given in this section shows how to light up the blue LED connected to Arduino digital pin 9.

Remember to change the path in the .mos files i.e
change
loadFile("D:/Modelica_DeviceDrivers/package.mo");
to
loadFile("*Path_to_ModelicaDeviceDrivers*/package.mo");
and
loadFile("D:/Arduino.mo");
to
loadFile("*Path_to_Arduino.mo/*Arduino.mo");

8.3.1 Windows:

1) Go to OMEdit and load Arduino.mo. Expand it and go to MDD_Examples->MDD_led->MDD_led_blue.

2) Open Command Prompt and browse to the path where you have stored the mos files. (Fig 6)

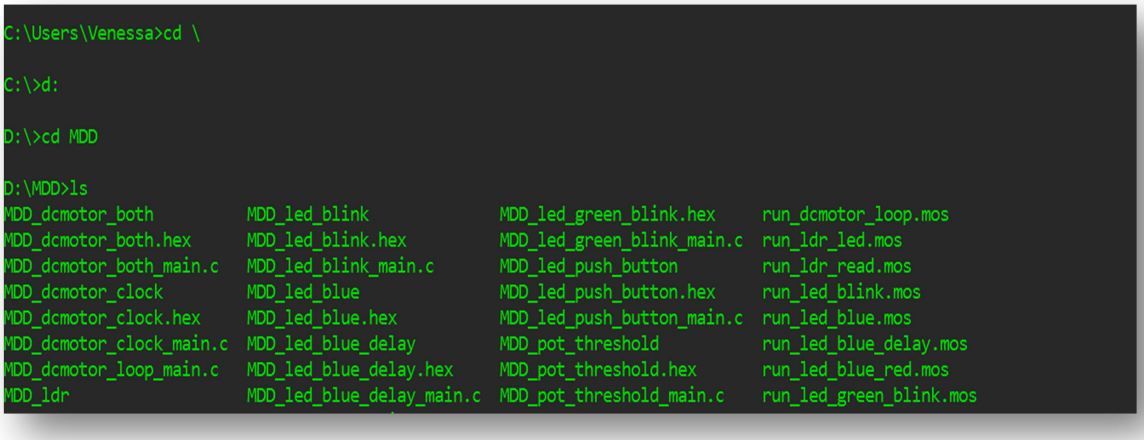


Fig 16. Browsing to path which contains the mos files

To run a simulation execute the following commands

- %OPENMODELICAHOME%\bin\omc --simCodeTarget=ExperimentalEmbeddedC run_led_blue.mos

Wait for it to return true, true, true, true.
Here “run_led_blue.mos” is the name of the mos file.(Fig 7)

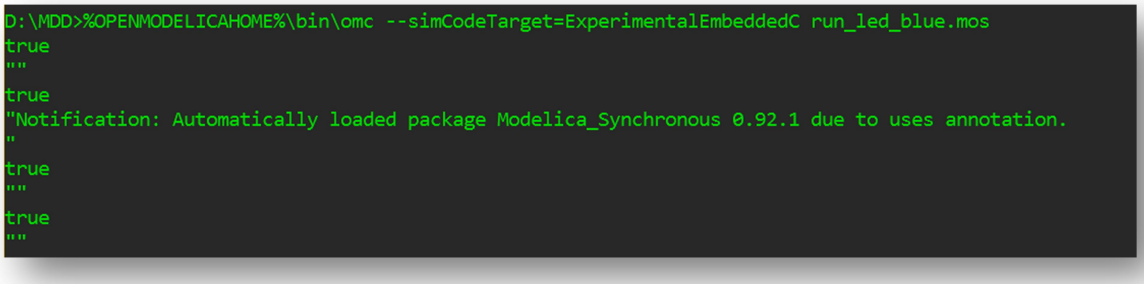


Fig 17. Compiling the mos file

- Then execute the following command for compilation:

avr-gcc -Os -std=c11 -ffunction-sections -fdata-sections -mmcu=atmega328p -DF_CPU=16000000UL -Wl,--gc-sections MDD_led_blue_main.c -o MDD_led_blue -I *path_to_Modelica_DeviceDrivers*/Modelica_deviceDrivers/Resources/Include -I %OPENMODELICAHOME%/include/omc/c

Here “MDD_led_blue” is the name of the simulation.

- Next, the following command is to be executed which generates a .hex file:

avr-objcopy -O ihex -R .eeprom MDD_led_blue MDD_led_blue.hex

- For loading the program into the Arduino board execute the following command:

C:/WinAVR-20100110/bin/avrdude -F -V -c arduino -p ATMEGA328P -P COM2 -b 115200 -U flash:w:MDD_led_blue.hex

(Note: “C:/WinAVR-20100110/bin/avrdude” is the path to avrdude.exe executable file. If these library path is added to Path environment variable, avrdude command can be directly used.)(Refer to Fig 8)

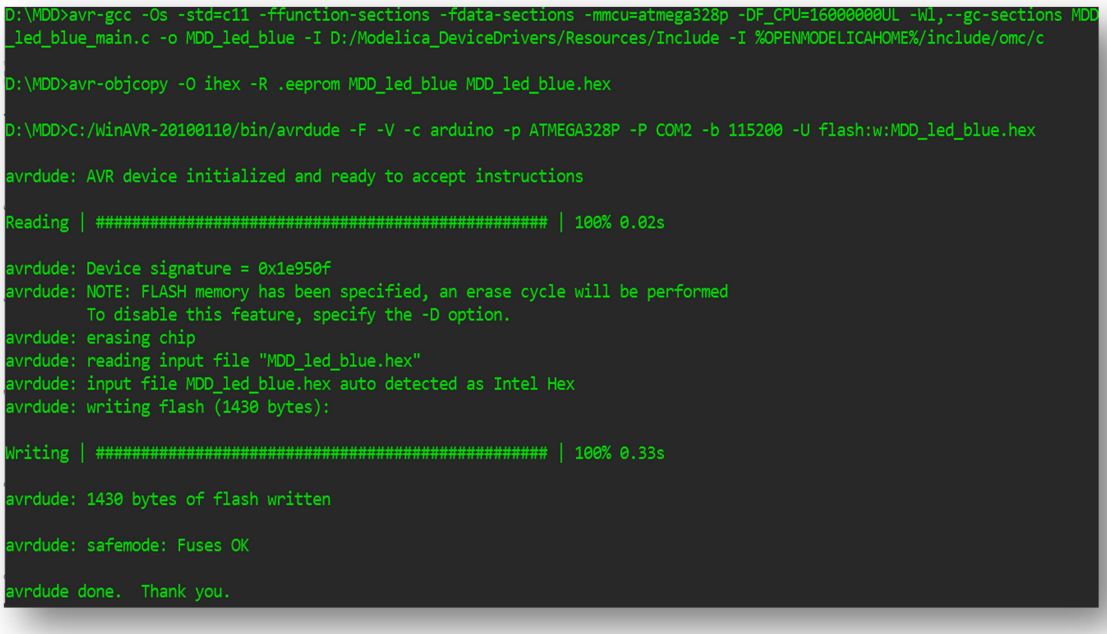


Fig 18. Flashing hex file on Arduino

8.3.2 Linux

1) Go to OMEdit and load Arduino.mo package file. Expand it and go to MDD_Examples->MDD_led->MDD_led_blue.

2) Open Terminal and browse to the path where the Modelica_DeviceDrivers Script files are stored (preferably MDD_build).

3) To run a simulation execute the following commands

- **omc --simCodeTarget=ExperimentalEmbeddedC runMDD_led_blue.mos**

Wait for it to return true, true, true, true.
Here run_led_blue.mos is the name of the mos file.(Fig 7)

- Then execute the following command for compilation:

avr-gcc -Os -std=c11 -ffunction-sections -fdata-sections -mmcu=atmega328p -DF_CPU=16000000UL -Wl,--gc-sections led_blue_main.c -o led_blue –I /path_to_Modelica_DeviceDrivers/Modelica_DeviceDrivers/Resources/Include -I /usr/include/omc/c

Here "MDD_led_blue" is the name of the simulation model.

- Next, the following command is to be executed which generates a .hex file:

avr-objcopy -O ihex -R .eeprom MDD_led_blue MDD_led_blue.hex

- For loading the program into the Arduino board execute the following command:

avrdude -F -V -c arduino -p ATMEGA328P -P COM2 -b 115200 -U flash:w:MDD_led_blue.hex

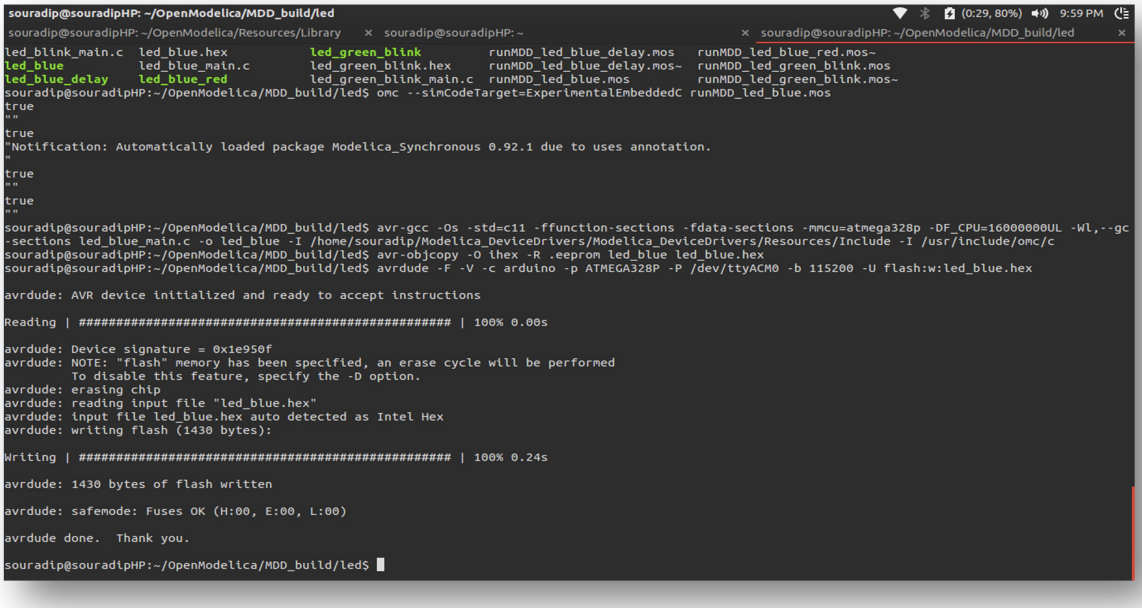


Fig 19. Flashing the hex file on Arduino in Linux

8.4 Simulation Settings for MDD Models

AVR Block Settings:

Microcontroller Block Settings:

The microcontroller block designated '**mcu**' has in-built parameters regarding the type of AVR platforms used in interfacing, using Modelica_DeviceDrivers. In this project, we have used Arduino UNO board containing Atmel Atmega328p chip. So the platform is set to ATmega328P.

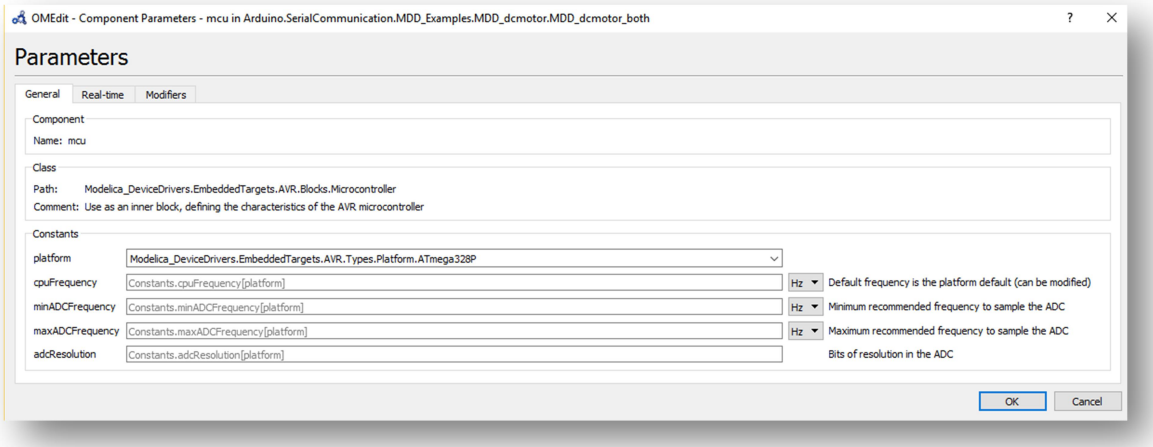


Fig 20. Microcontroller Block Settings

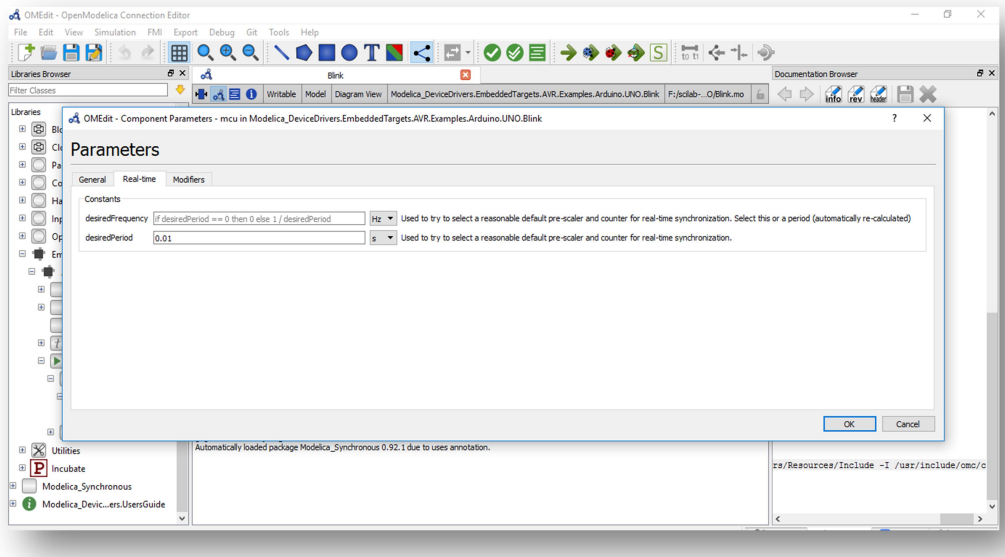
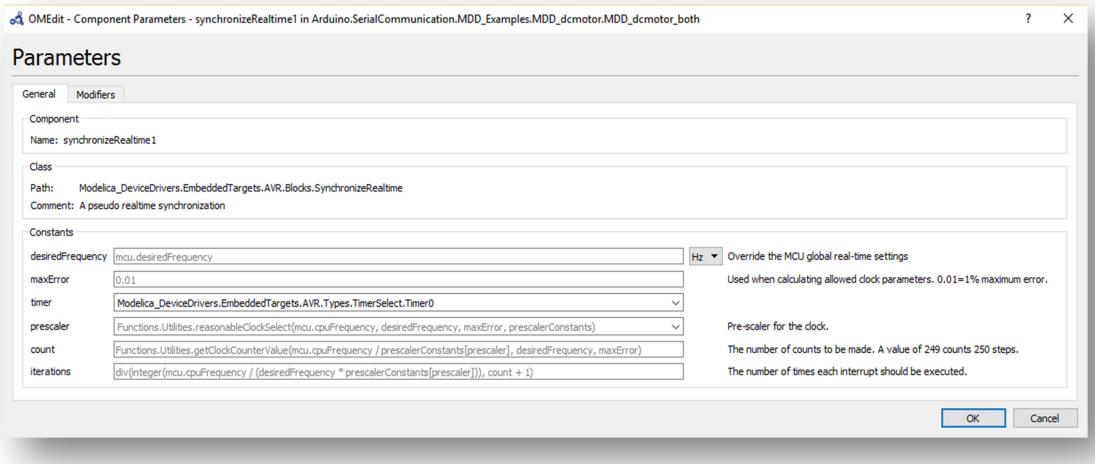


Fig 21. Synchronize Real Time Settings

Synchronize Real Time Settings:

SynchronizeRealTime block is a pseudo realtime synchronization functional block.It synchronizes the simulation time of the simulation process with the operating system real-time clock.(If desired period =0.01,1 time unit= 5 seconds. In this case, 'time' variable in OpenModelica runs faster 5 times the actual clock time. 'time' can be exactly synchronized with actual clock time by setting the desired period = 0.002)



Timer0 is an 8-bit timer which is used here to synchronize the platform clock with CPU clock of the system. Further details about platform timer can be found here http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf.

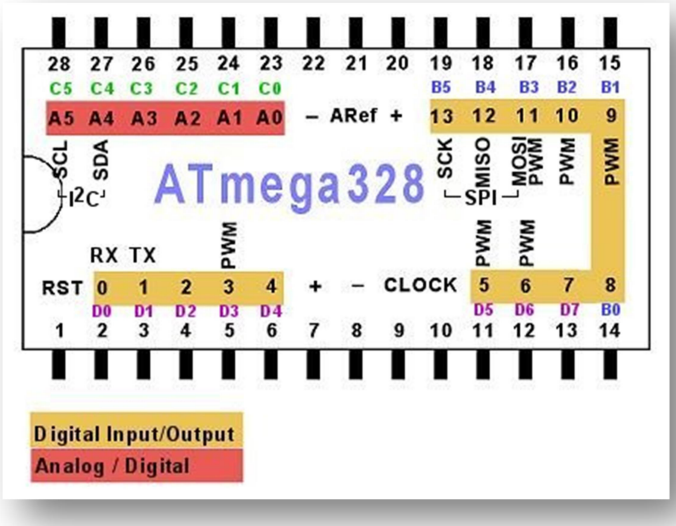


Fig 22. Arduino to ATMega328P Mapping

The above mentioned settings are common in general for all the examples given here.

8.4.1 Interfacing LED using Modelica Device Drivers

Example 1: MDD_led_blue_delay

The following is an example to turn on the blue led for 5 seconds.

Double clicking on each block opens the parameter windows for it.

Change the parameters according to Fig 23.(1 time unit =5 seconds)

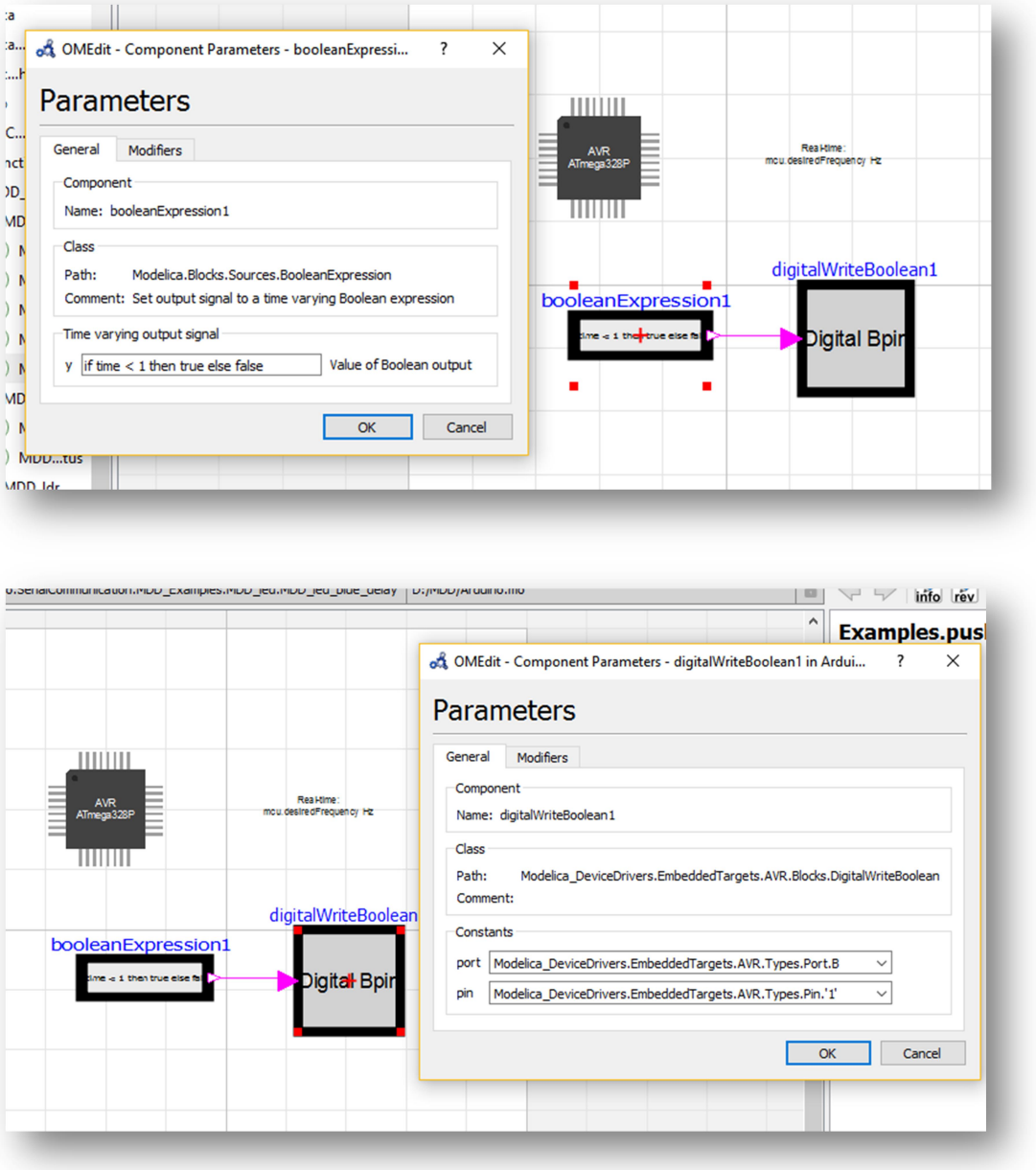


Fig 23. MDD_led_blue_delay

Example 2: MDD_led_blue

The following is an example to turn on the blue led forever.

Double clicking on each block opens the parameter windows for it.

Change the parameters according to Fig 24.

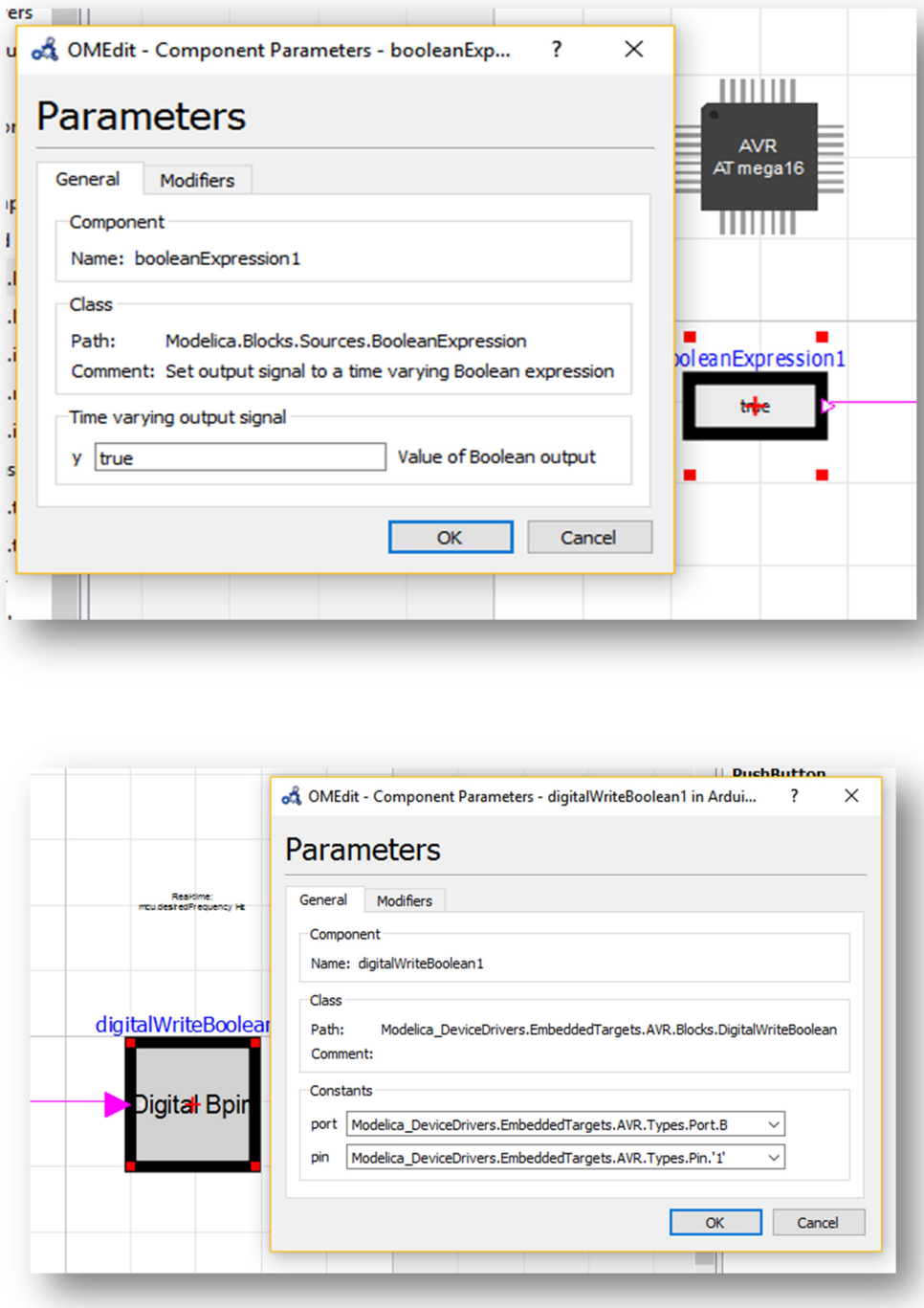
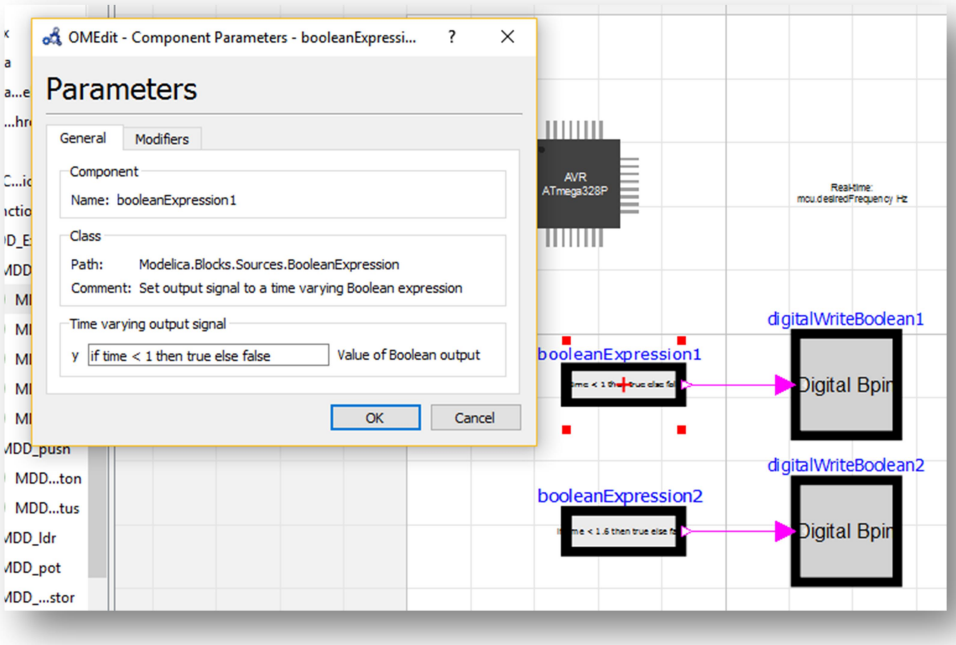


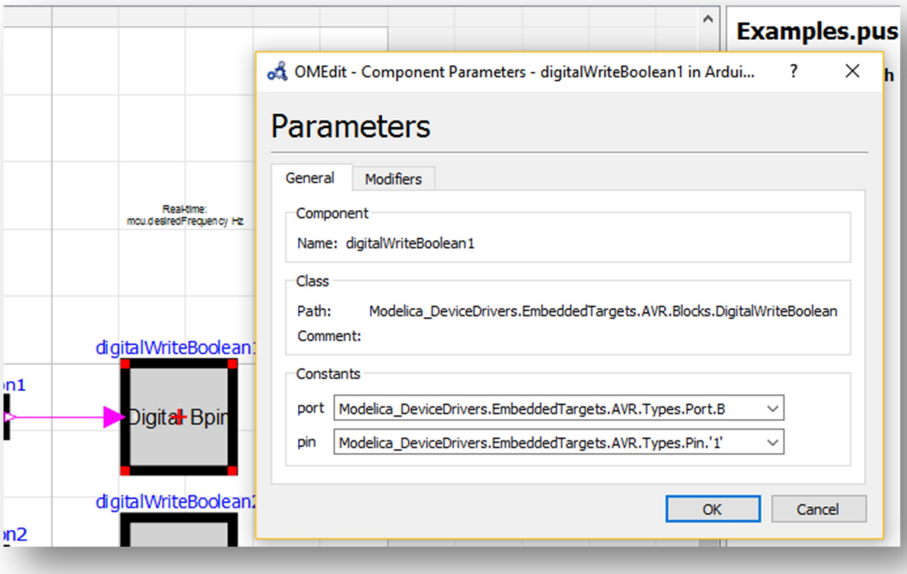
Fig 24. MDD_led_blue

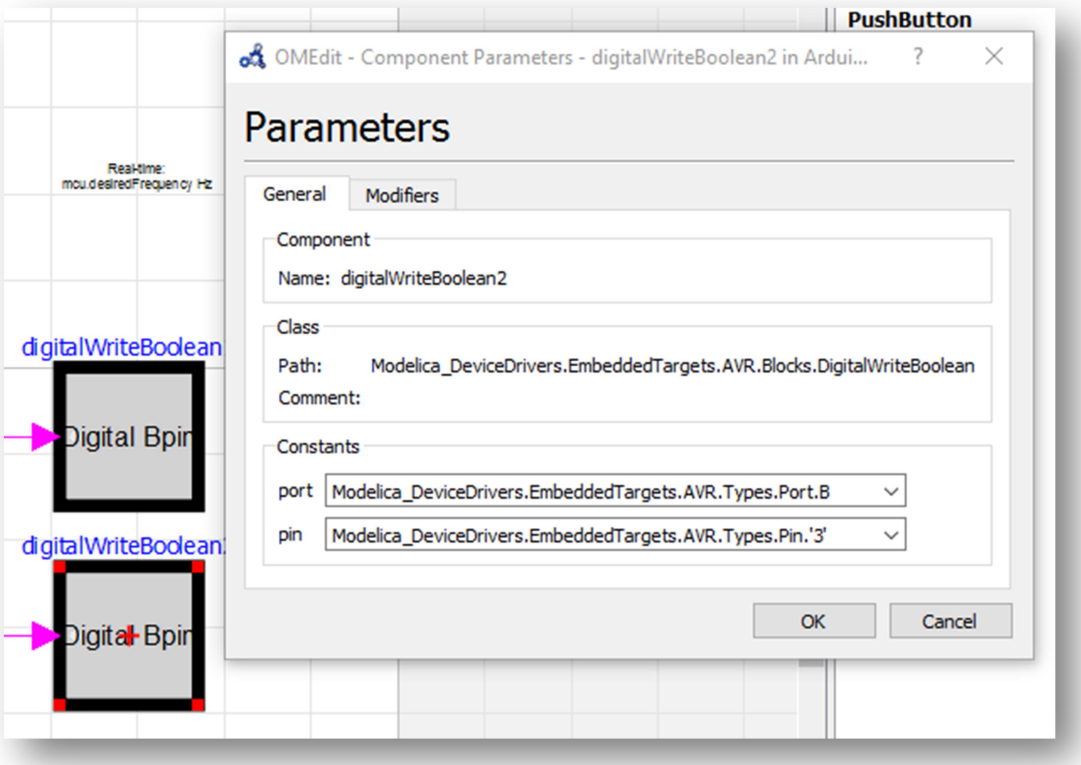
Example 3: MDD_led_blue_red

The following is an example to turn on the both blue and red led for 5 seconds. Then turn off the blue led and after 3 seconds also turn off the red led.

Double Clicking on each block opens the parameter windows for it. Change the parameters according to Fig 25.







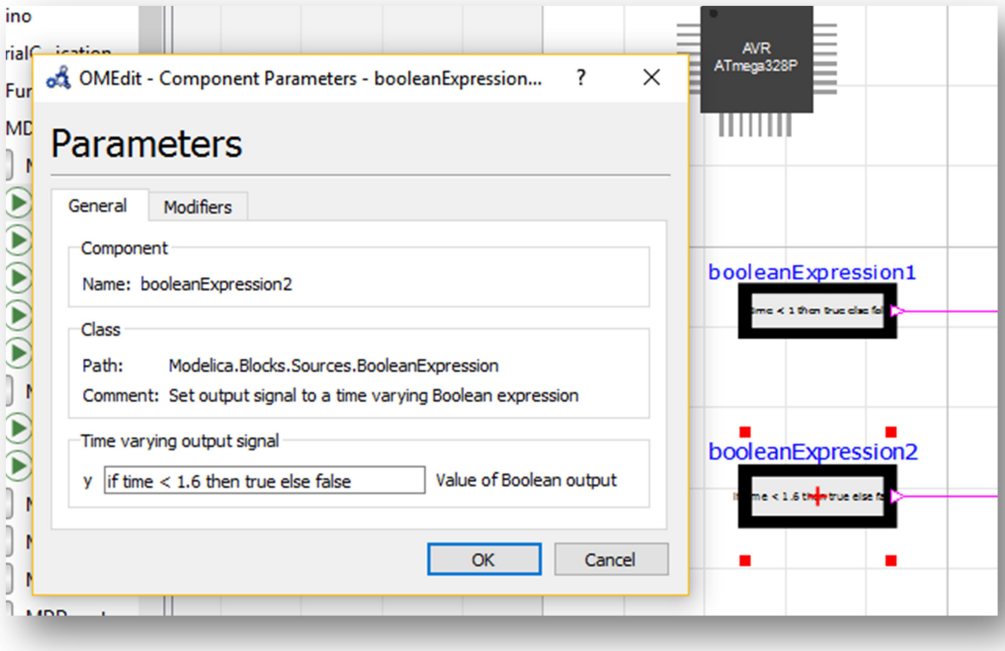
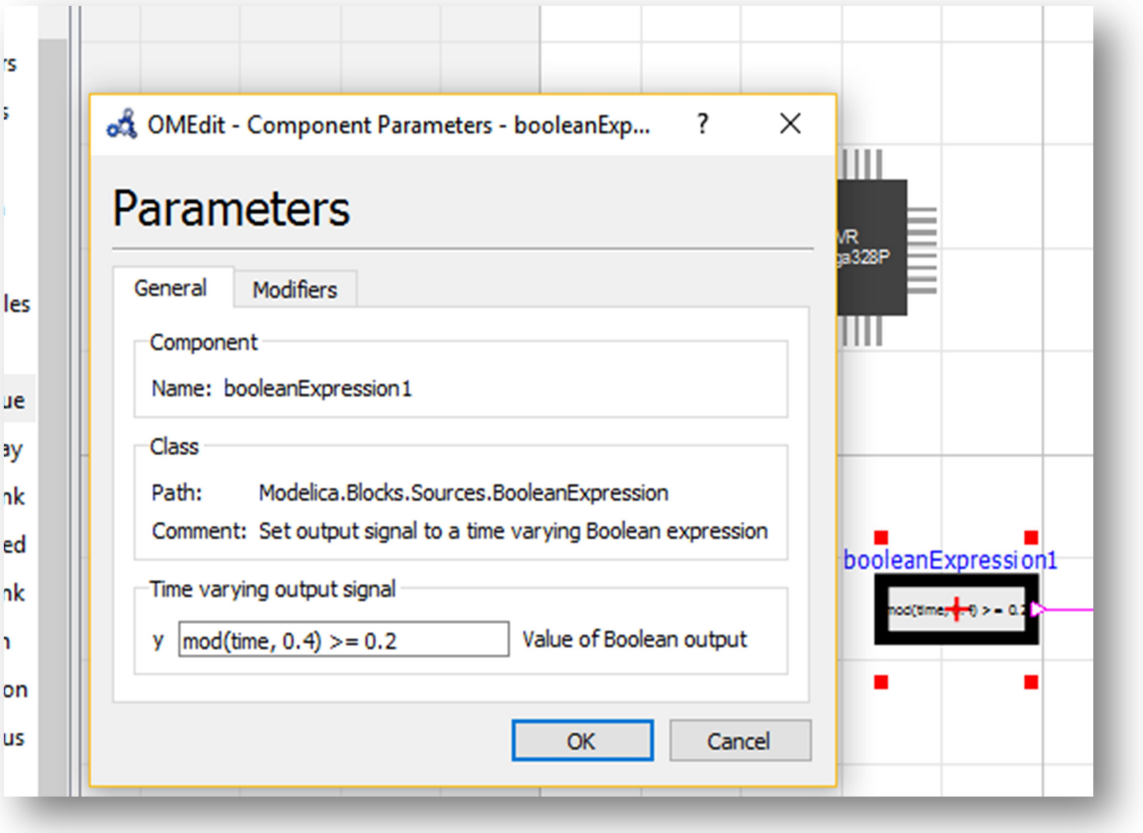


Fig 25. MDD_led_blue_red

Example 4: MDD_led_green_blink
The following is an example to turn on and off the led inbuilt on arduino board having a period of 5 seconds.
Double Clicking on each block opens the parameter windows for it.
Change the parameters according to Fig 26.



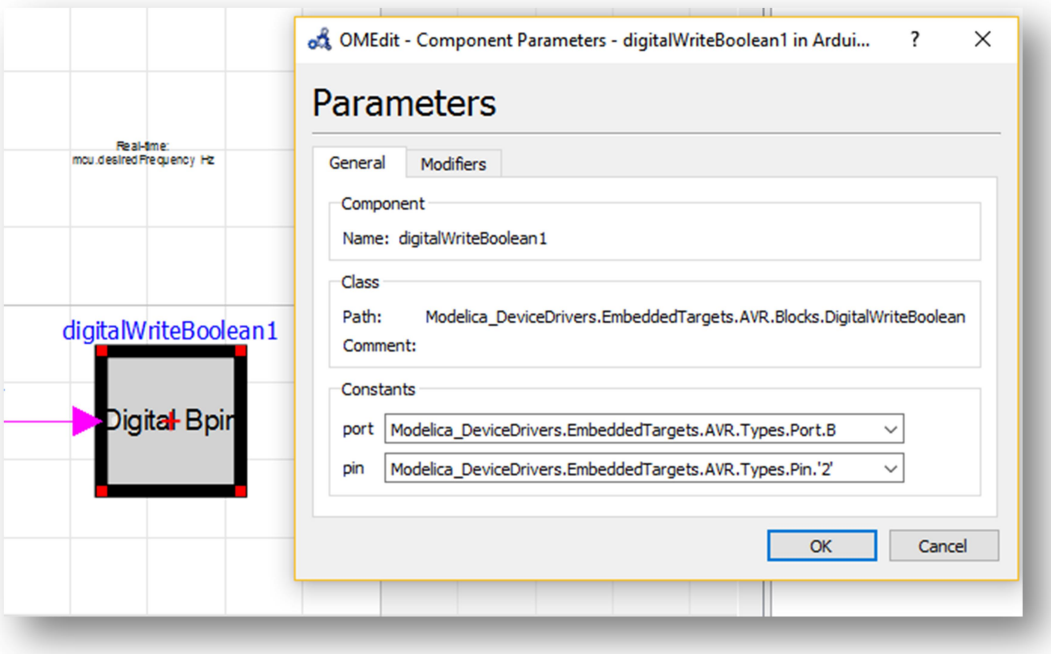


Fig 26. MDD_led_green_blink

Example 5: MDD_led_blink

The following is an example to turn on and off the led inbuilt on arduino board having a period of 5 seconds.

Double Clicking on each block opens the parameter windows for it.

Change the parameters according to Fig 27.

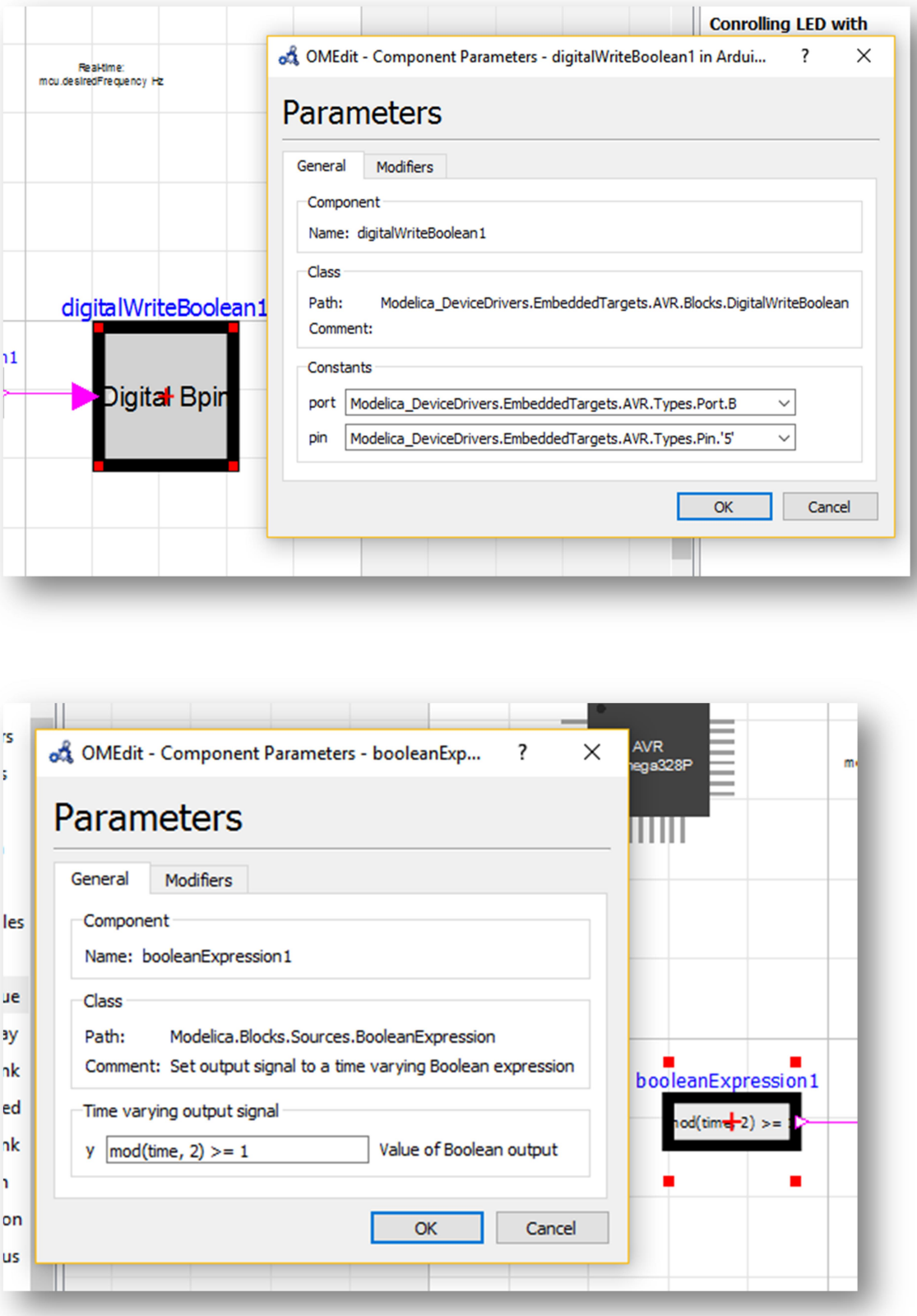


Fig 27.MDD_led_blink

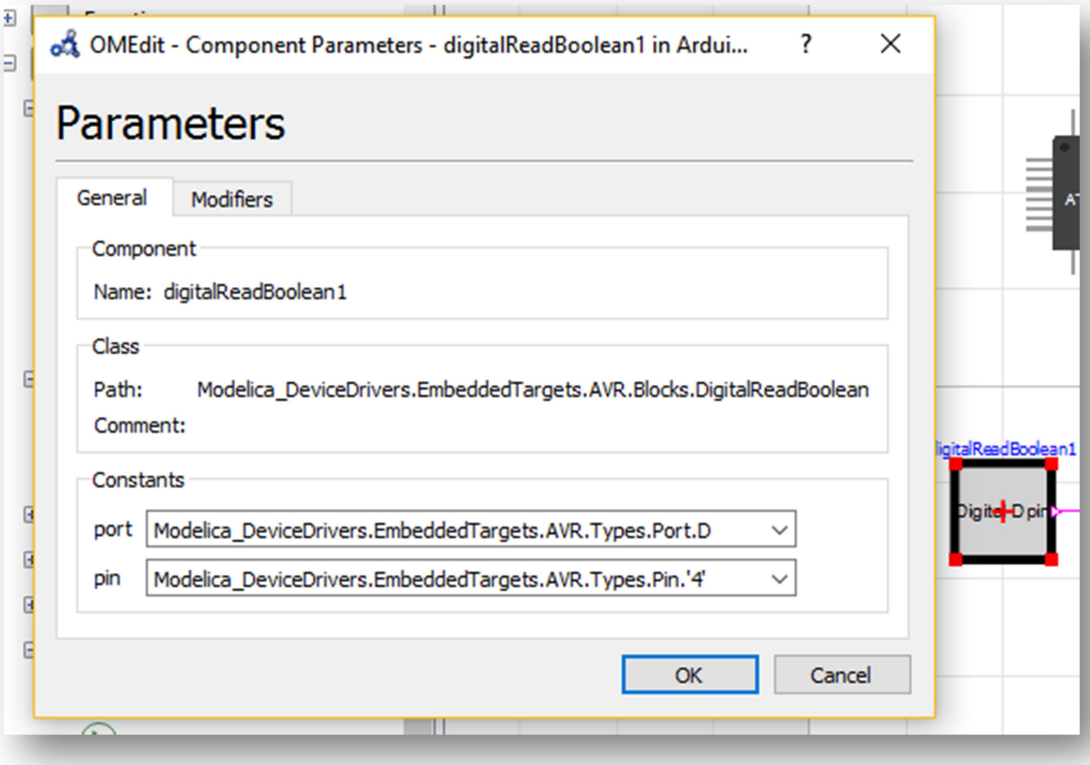
8.4.2 Interfacing Push Button with Modelica_DeviceDrivers

Example 1: MDD_led_push_button

The following is an example to turn on the blue led when the pushbutton is pushed.

Double Clicking on each block opens the parameter windows for it.

Change the parameters according to Fig 28.



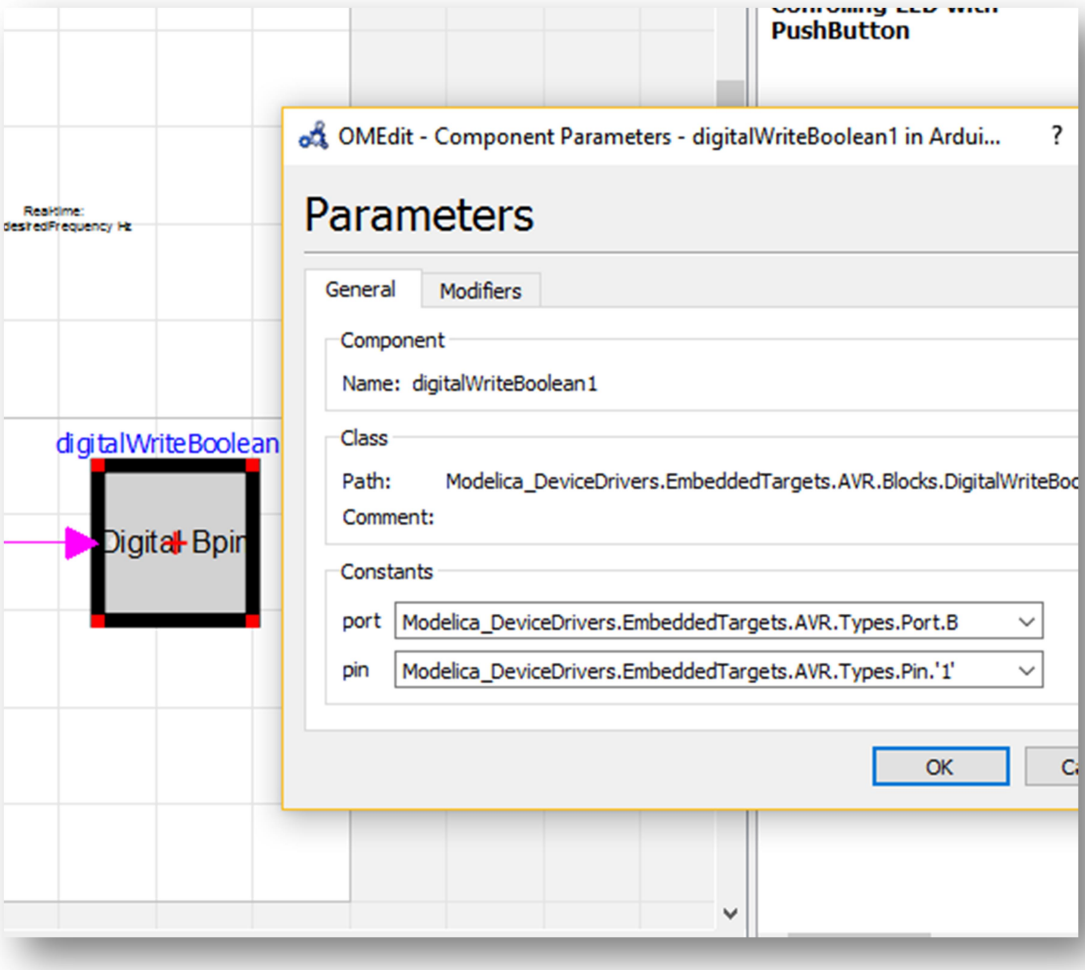


Fig 28. MDD_led_push_button

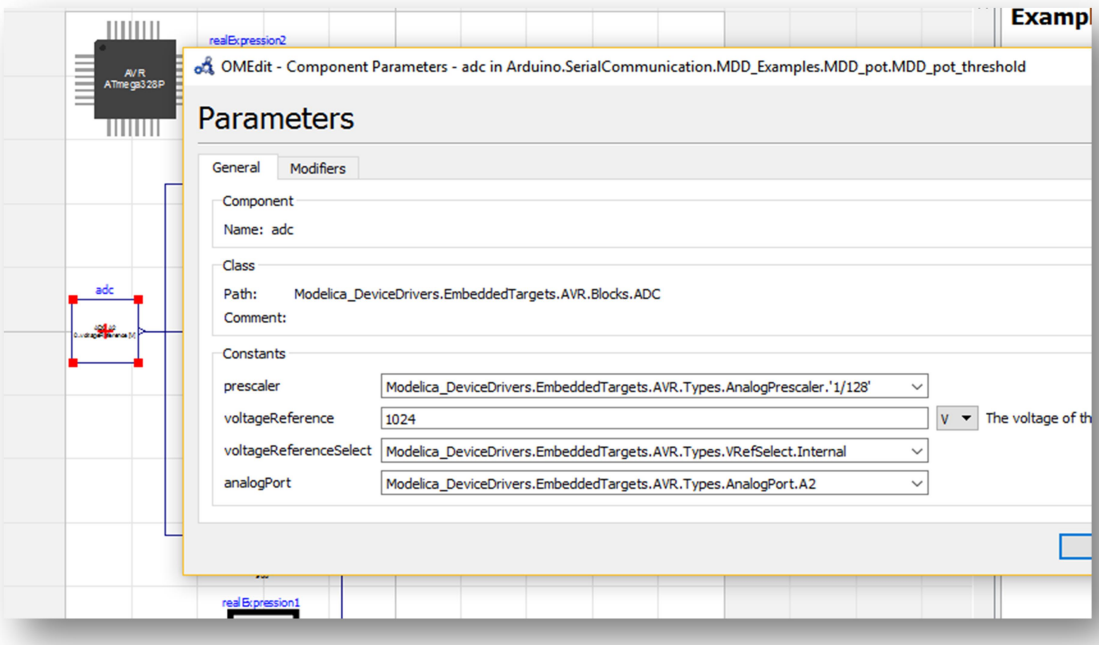
8.4.3 Interfacing Potentiometer with Modelica_DeviceDrivers

Example 1: MDD_pot_threshold

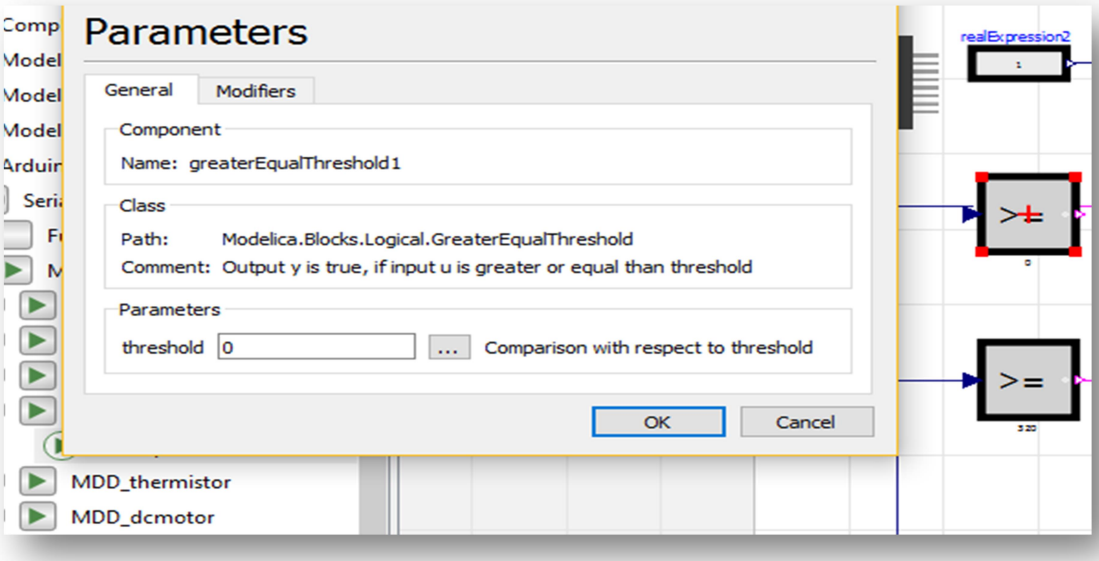
The following is an example to control the RGB led using potentiometer.

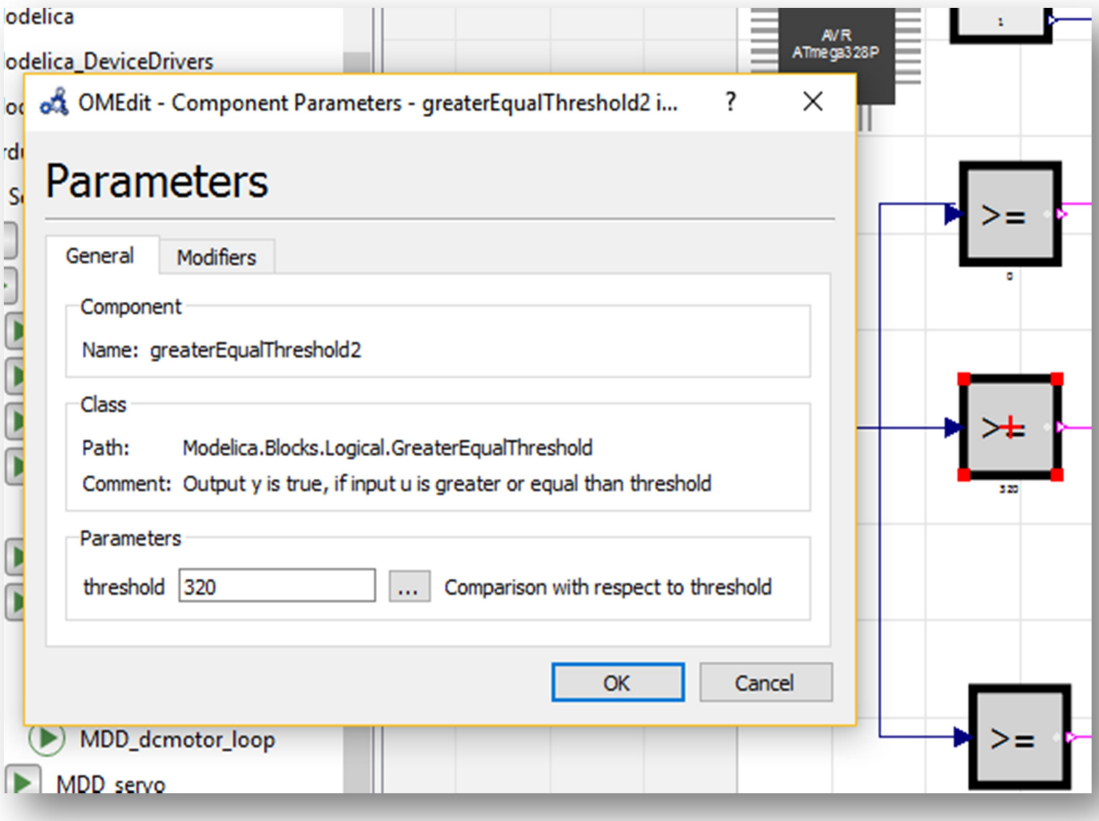
Double Clicking on each block opens the parameter windows for it.

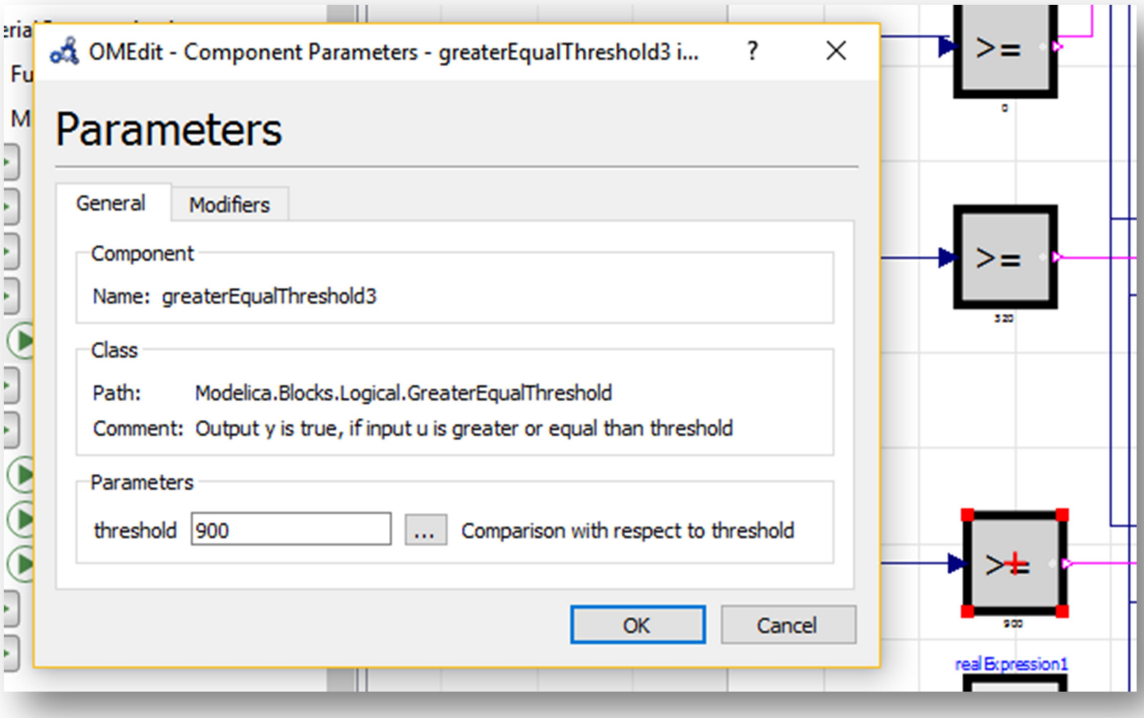
Change the parameters according to Fig 29.

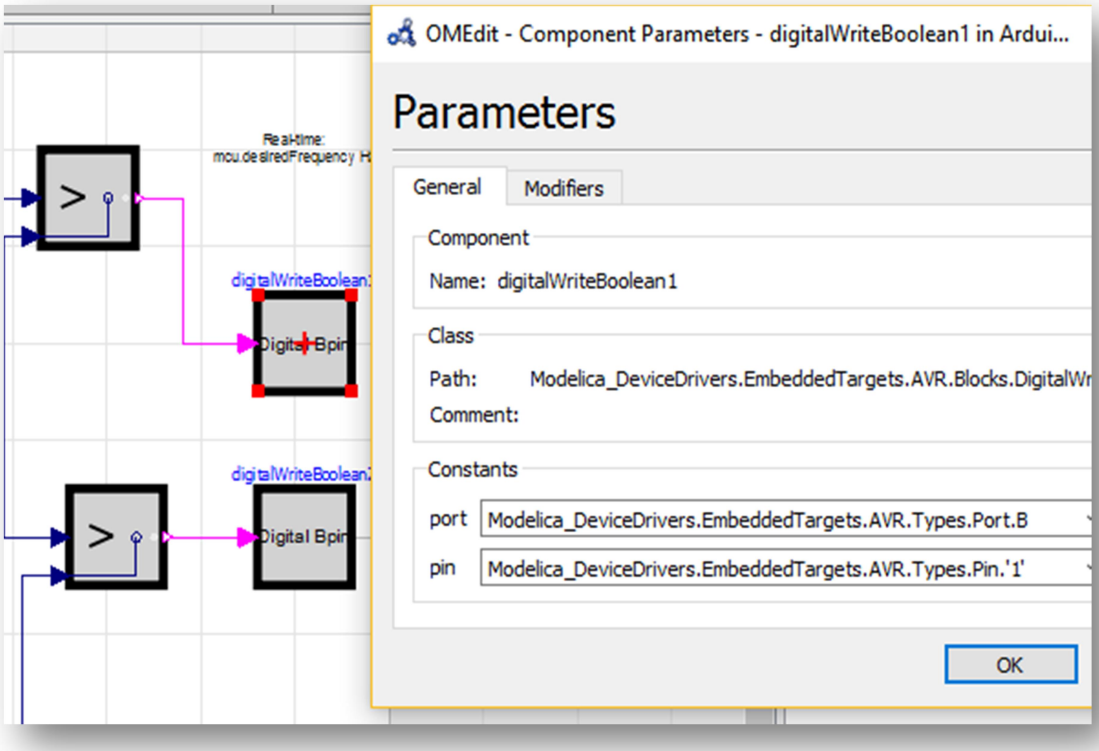


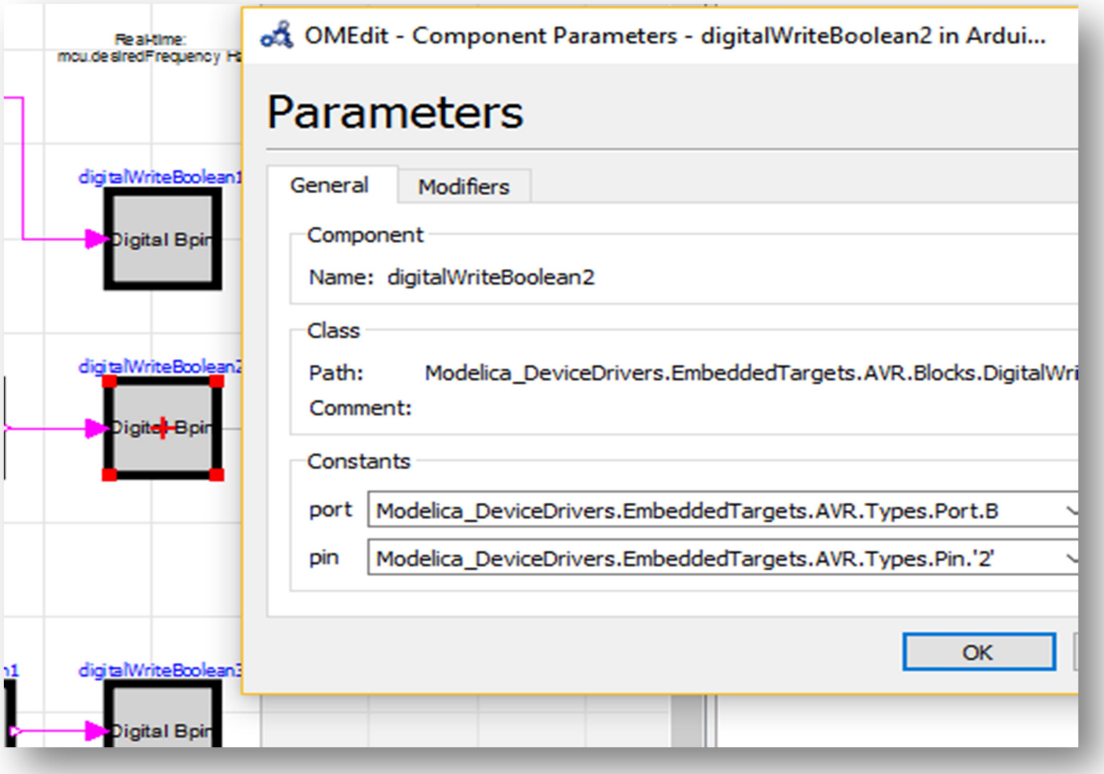
Here,the parameter 'voltageReference' is set to '1024' so digital values are in the range 0-1024.




odelica
odelica_DeviceDrivers
AVR
ATmega328P
OMEdit - Component Parameters - greaterEqualThreshold2 i...
Parameters
General
Modifiers
Component
Name: greaterEqualThreshold2
Class
Path: Modelica.Blocks.Logical.GreaterEqualThreshold
Comment: Output y is true, if input u is greater or equal than threshold
Parameters
threshold 320
...
Comparison with respect to threshold
OK
Cancel
>=
MDD_dcmotor_loop
MDD_servo


OMEdit - Component Parameters - greaterEqualThreshold3 i...
Parameters
General
Modifiers
Component
Name: greaterEqualThreshold3
Class
Path: Modelica.Blocks.Logical.GreaterEqualThreshold
Comment: Output y is true, if input u is greater or equal than threshold
Parameters
threshold 900
...
Comparison with respect to threshold
OK
Cancel
>=
realExpression1


OMEdit - Component Parameters - digitalWriteBoolean1 in Ardui...
Parameters
General
Modifiers
Component
Name: digitalWriteBoolean1
Class
Path: Modelica_DeviceDrivers.EmbeddedTargets.AVR.Blocks.DigitalWr
Comment:
Constants
port Modelica_DeviceDrivers.EmbeddedTargets.AVR.Types.Port.B
pin Modelica_DeviceDrivers.EmbeddedTargets.AVR.Types.Pin.'1'
OK


OMEdit - Component Parameters - digitalWriteBoolean2 in Ardui...
Parameters
General
Modifiers
Component
Name: digitalWriteBoolean2
Class
Path: Modelica_DeviceDrivers.EmbeddedTargets.AVR.Blocks.DigitalWri
Comment:
Constants
port Modelica_DeviceDrivers.EmbeddedTargets.AVR.Types.Port.B
pin Modelica_DeviceDrivers.EmbeddedTargets.AVR.Types.Pin.'2'
OK

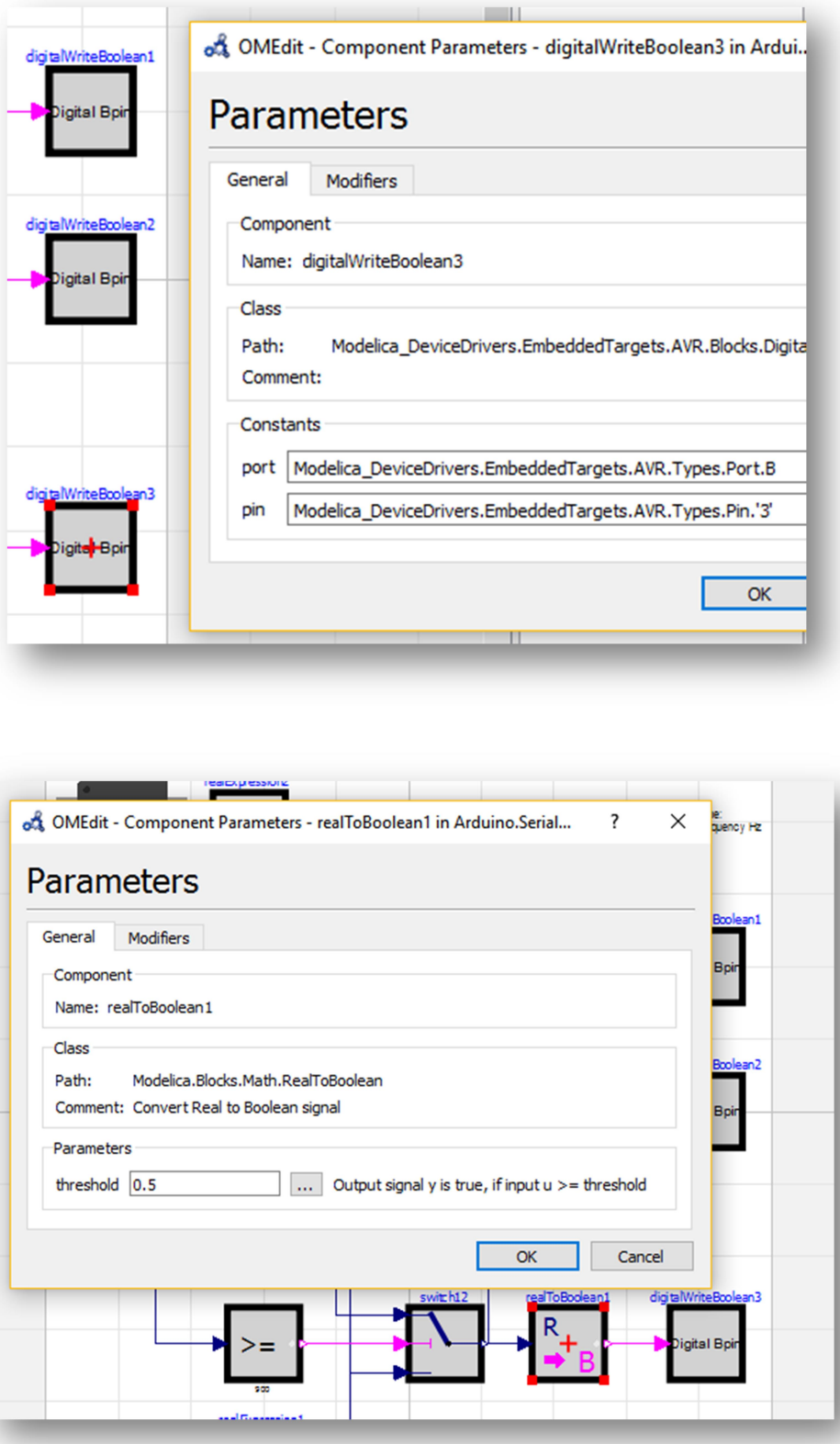


Fig 29. MDD_pot_threshold

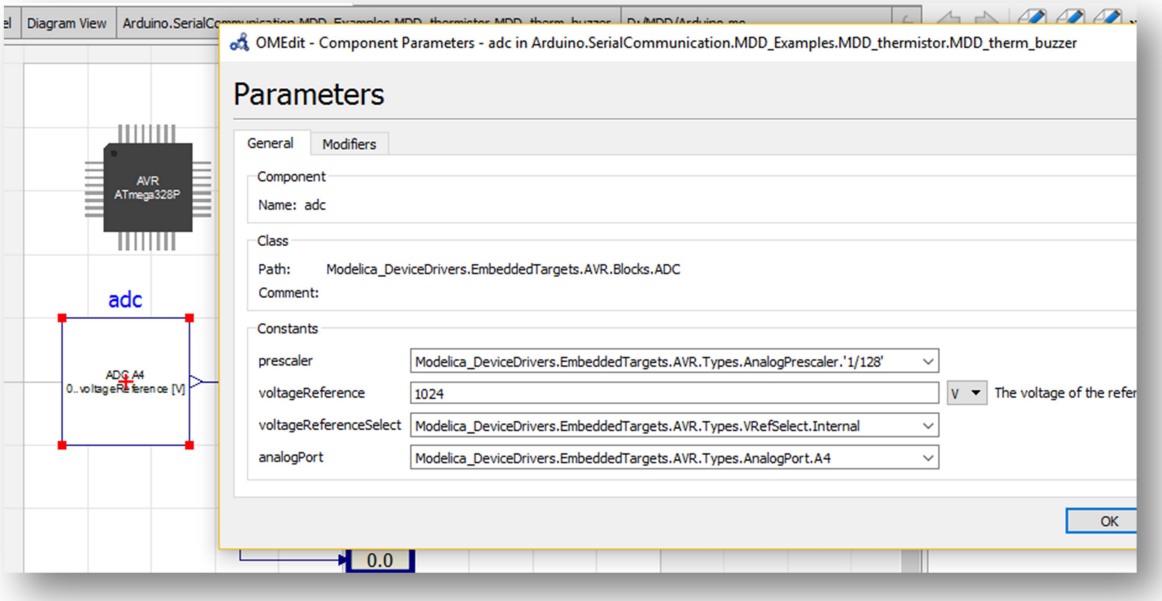
8.4.5 Interfacing Thermistor with Modelica_DeviceDrivers

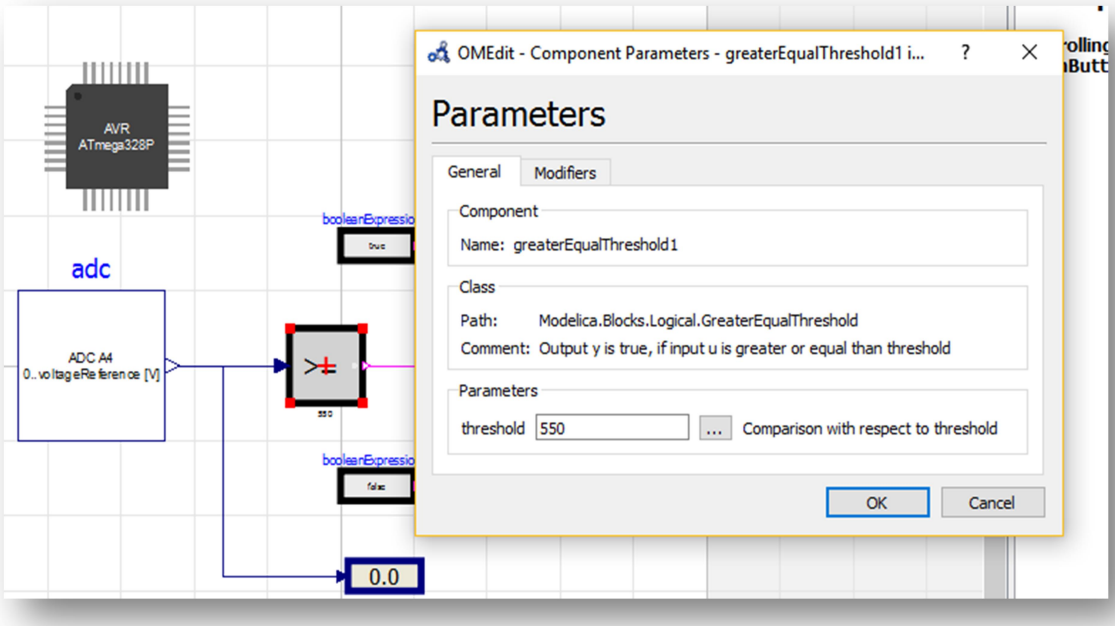
Example 1: MDD_therm_buzzer

The following is an example to control the buzzer based on thermistor readings.

Double Clicking on each block opens the parameter windows for it.

Change the parameters according to Fig 30.





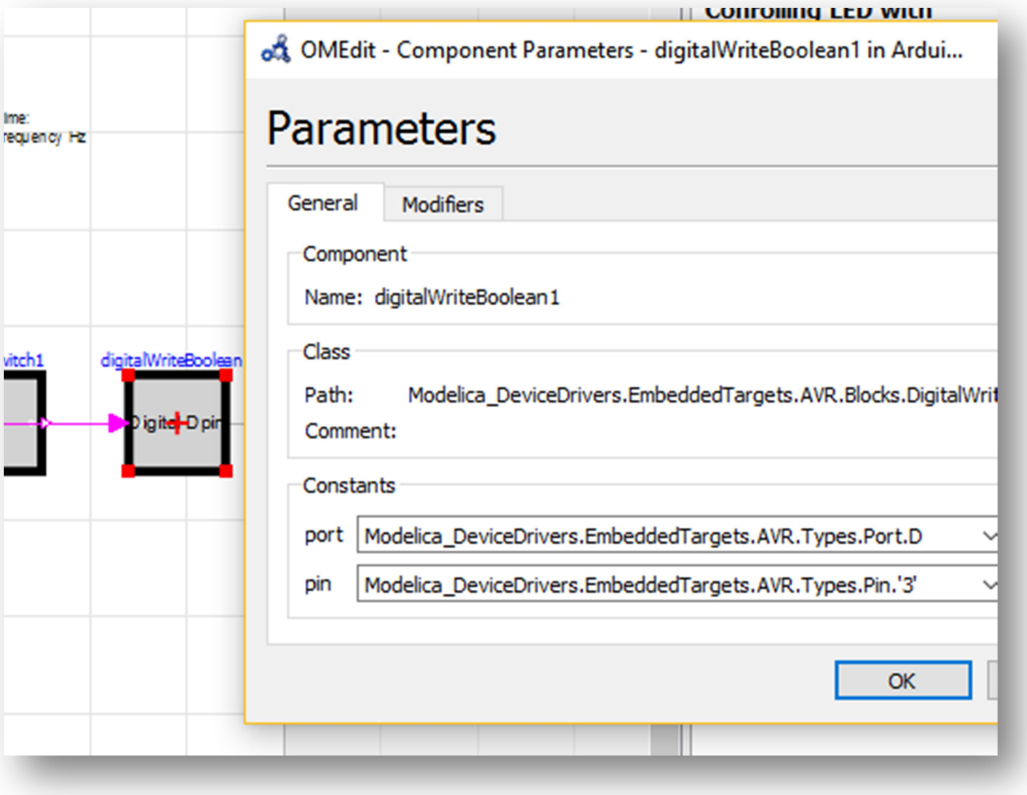


Fig 30. MDD_therm_buzzer

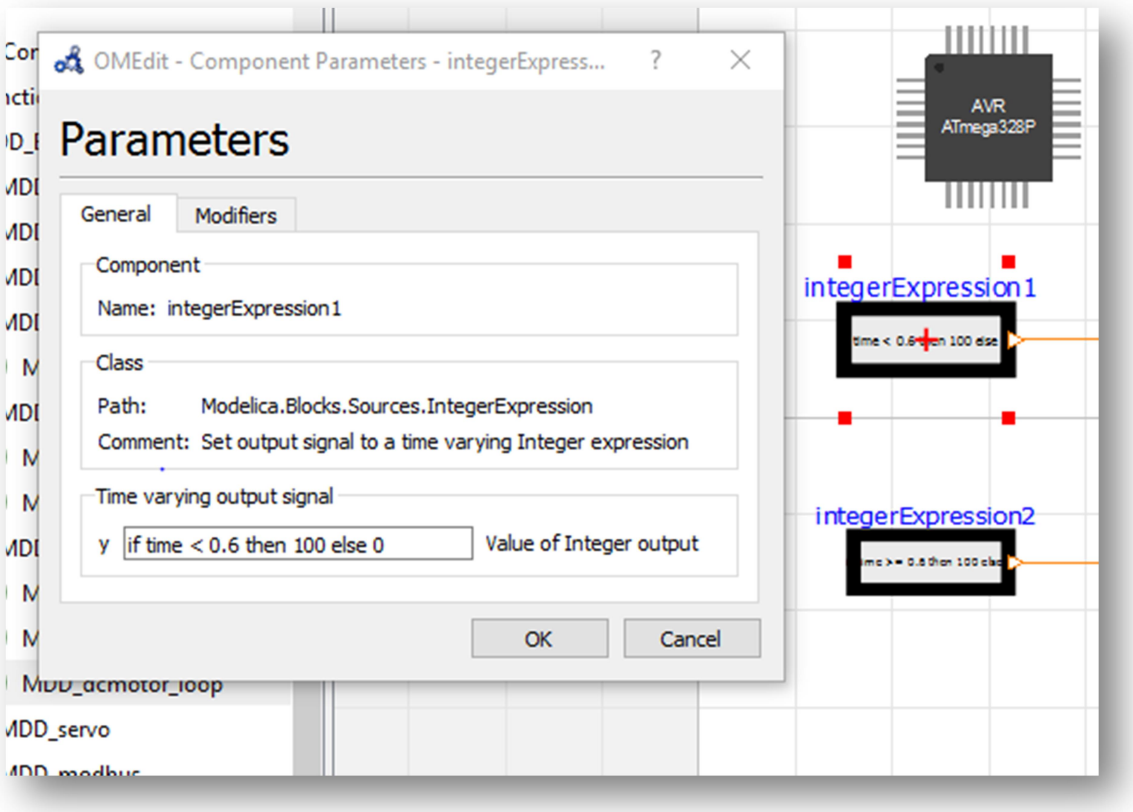
8.4.5 Interfacing DC Motor with Modelica_DeviceDrivers

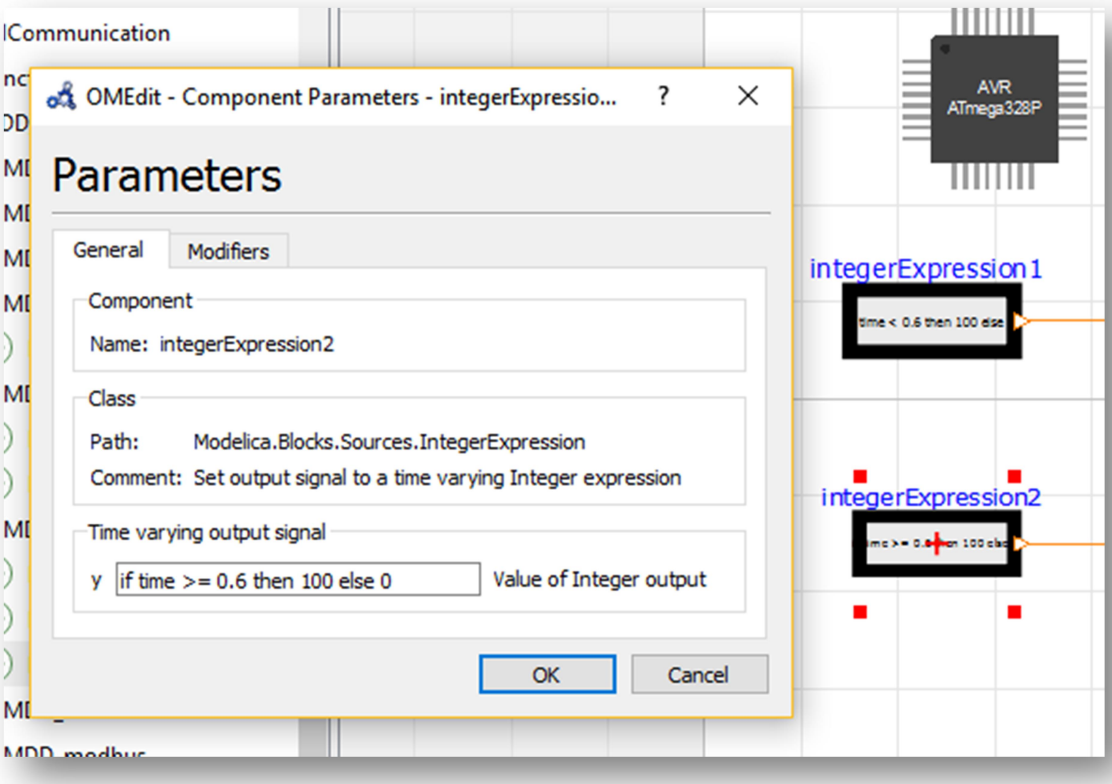
Example 1: MDD_dcmotor_loop

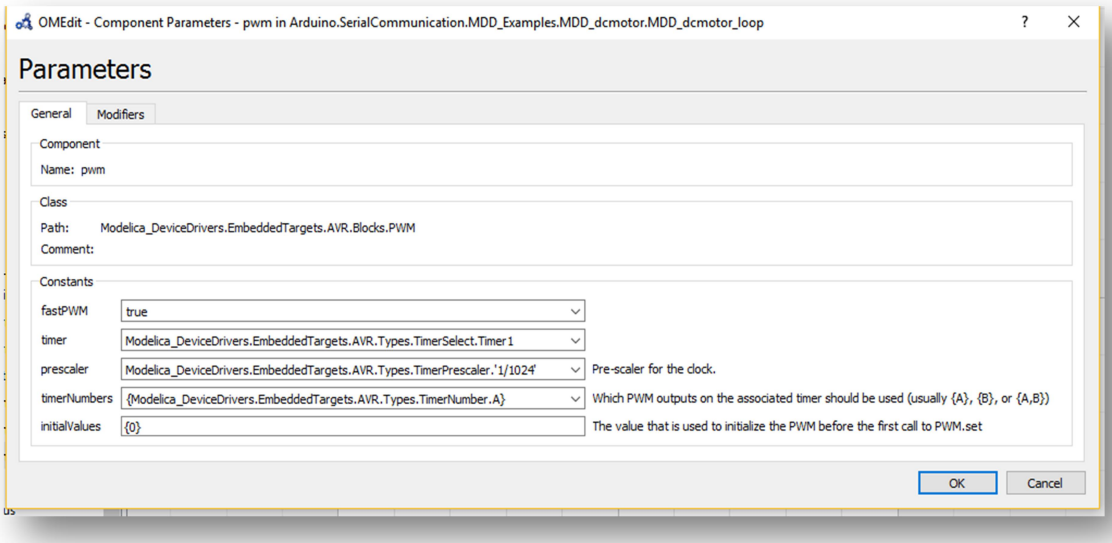
The following is an example to rotate the dcmotor.

Double Clicking on each block opens the parameter windows for it.

Change the parameters according to Fig 31.


OMEdit - Component Parameters - integerExpress...
Parameters
General
Modifiers
Component
Name: integerExpression1
Class
Path: Modelica.Blocks.Sources.IntegerExpression
Comment: Set output signal to a time varying Integer expression
Time varying output signal
y
if time < 0.6 then 100 else 0
Value of Integer output
OK
Cancel
AVR
ATmega328P
integerExpression1
integerExpression2
MDD_servo


Communication
OMEdit - Component Parameters - integerExpressio...
Parameters
General
Modifiers
Component
Name: integerExpression2
Class
Path: Modelica.Blocks.Sources.IntegerExpression
Comment: Set output signal to a time varying Integer expression
Time varying output signal
y
if time >= 0.6 then 100 else 0
Value of Integer output
OK
Cancel
AVR
ATmega328P
integerExpression1
integerExpression2


OMEdit - Component Parameters - pwm in Arduino.SerialCommunication.MDD_Examples.MDD_dcmotor.MDD_dcmotor_loop
Parameters
General
Modifiers
Component
Name: pwm
Class
Path: Modelica_DeviceDrivers.EmbeddedTargets.AVR.Blocks.PWM
Comment:
Constants
fastPWM
true
timer
Modelica_DeviceDrivers.EmbeddedTargets.AVR.Types.TimerSelect.Timer1
prescaler
Modelica_DeviceDrivers.EmbeddedTargets.AVR.Types.TimerPrescaler.'1/1024'
Pre-scaler for the clock.
timerNumbers
{Modelica_DeviceDrivers.EmbeddedTargets.AVR.Types.TimerNumber.A}
Which PWM outputs on the associated timer should be used (usually {A}, {B}, or {A,B})
initialValues
{0}
The value that is used to initialize the PWM before the first call to PWM.set
OK
Cancel

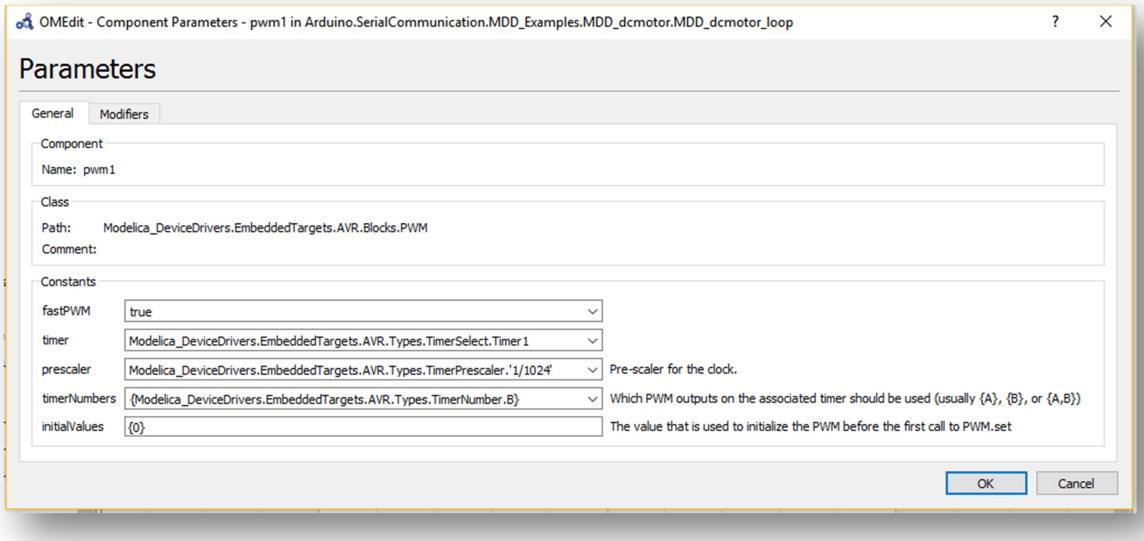
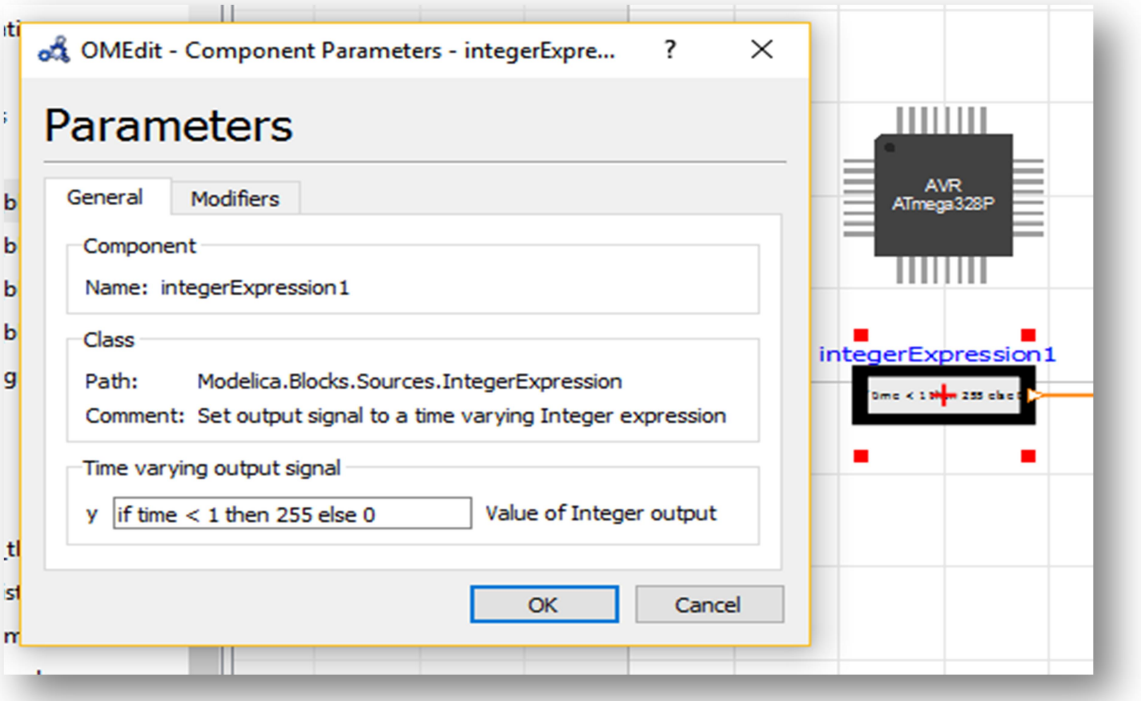


Fig 31.MDD_dcmotor_loop

Example 2: MDD_dcmotor_clock

The following is an example to rotate the dcmotor only clockwise. Double Clicking on each block opens the parameter windows for it. Change the parameters according to Fig 32.



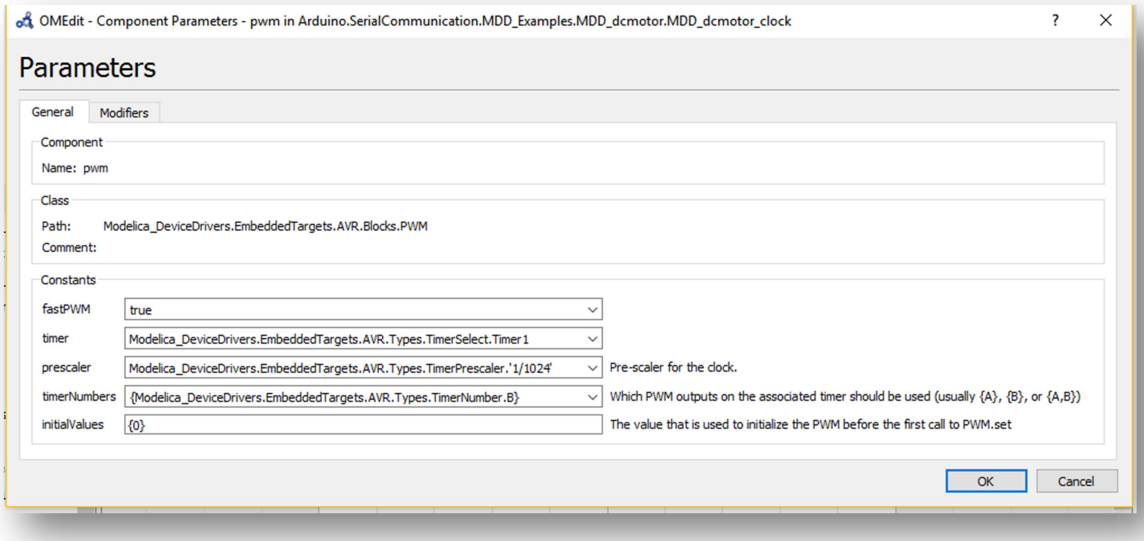


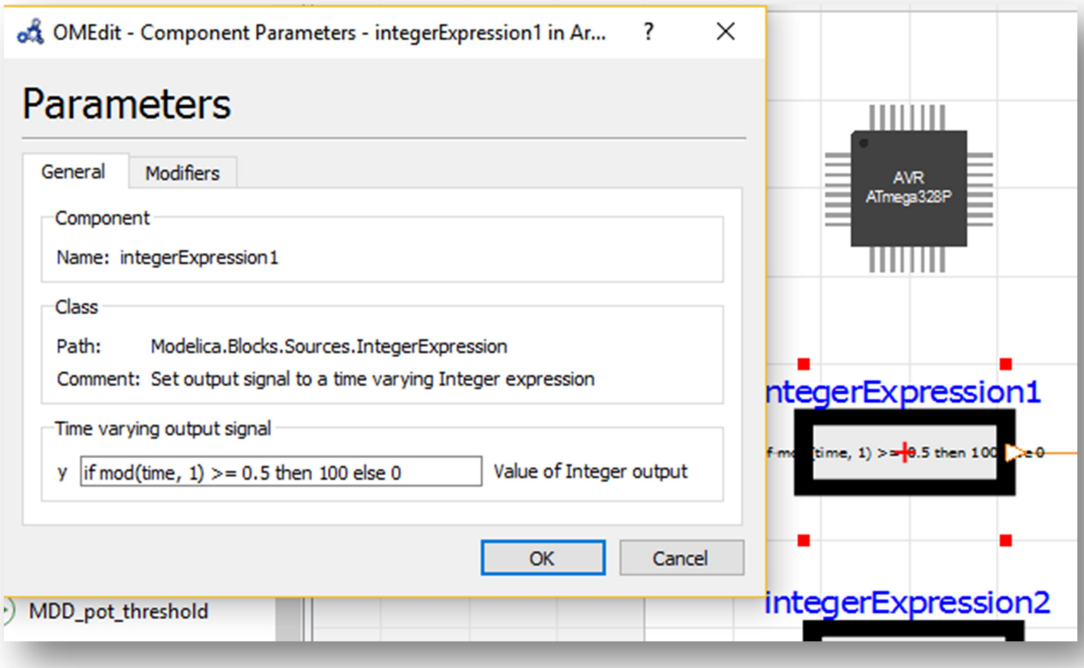
Fig 32.MDD_dcmotor_clock

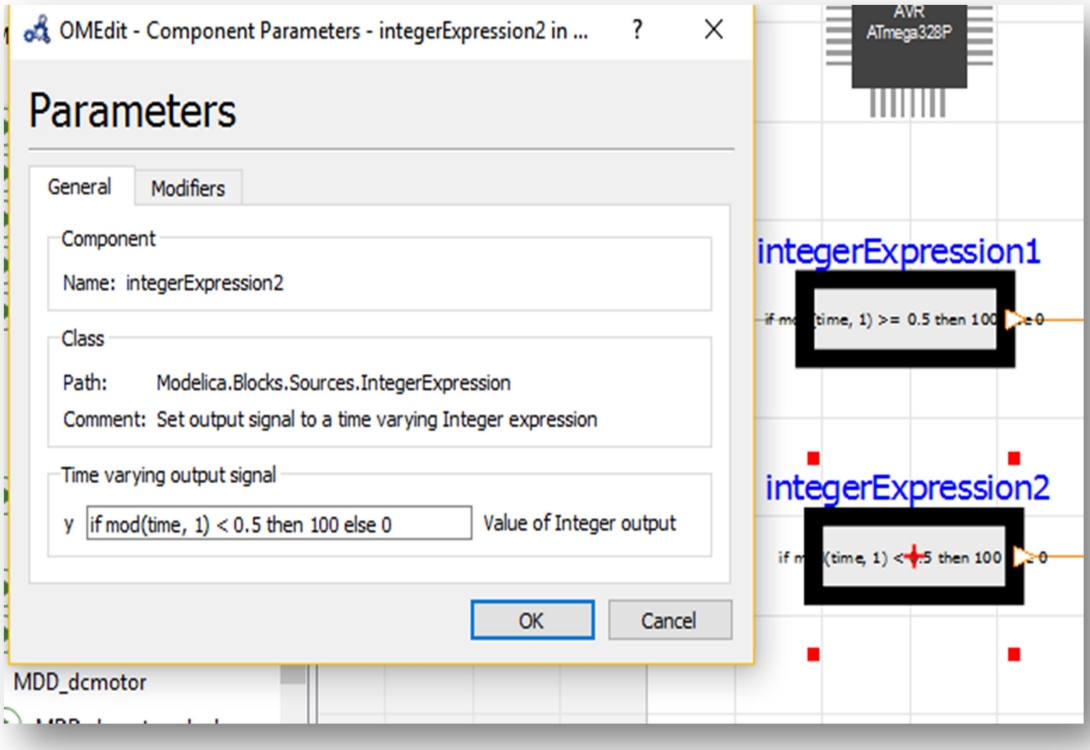
Here, timerNumbers are the pins to which PWM values are output. 'A' & 'B' correspond to PWM pins 9 & 10 respectively in this case.

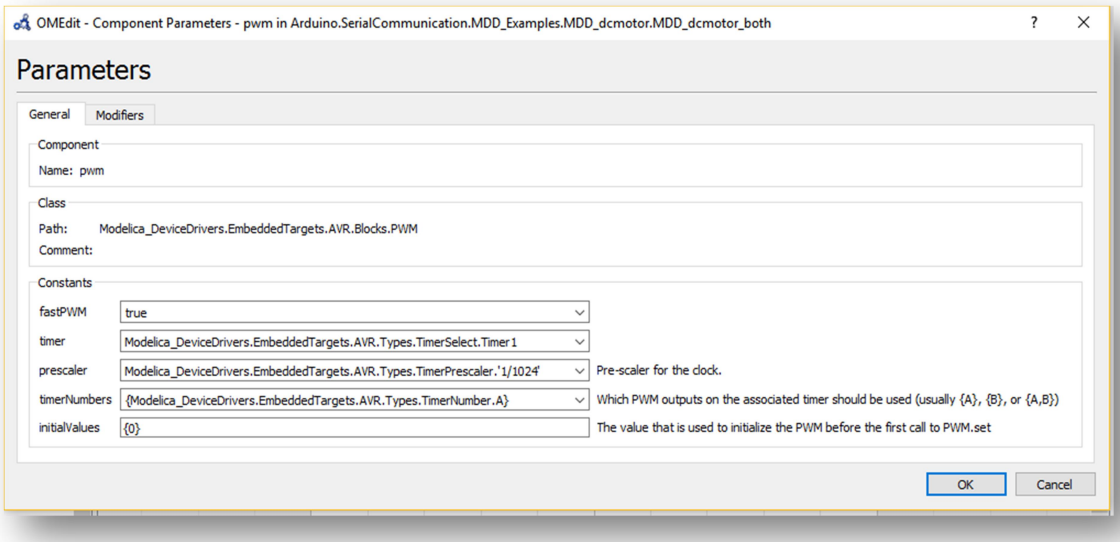
Example 3: MDD_dcmotor_both

The following is an example to rotate the dcmotor both clockwise as well as anticlockwise.

Double Clicking on each block opens the parameter windows for it. Change the parameters according to Fig 33.




OMEdit - Component Parameters - integerExpression2 in ...
Parameters
General
Modifiers
Component
Name: integerExpression2
Class
Path: Modelica.Blocks.Sources.IntegerExpression
Comment: Set output signal to a time varying Integer expression
Time varying output signal
y
if mod(time, 1) < 0.5 then 100 else 0
Value of Integer output
OK
Cancel
AVR
ATmega328P
integerExpression1
integerExpression2
MDD_dcmotor


OMEdit - Component Parameters - pwm in Arduino.SerialCommunication.MDD_Examples.MDD_dcmotor.MDD_dcmotor_both
Parameters
General
Modifiers
Component
Name: pwm
Class
Path: Modelica_DeviceDrivers.EmbeddedTargets.AVR.Blocks.PWM
Comment:
Constants
fastPWM
true
timer
Modelica_DeviceDrivers.EmbeddedTargets.AVR.Types.TimerSelect.Timer1
prescaler
Modelica_DeviceDrivers.EmbeddedTargets.AVR.Types.TimerPrescaler.'1/1024'
Pre-scaler for the clock.
timerNumbers
{Modelica_DeviceDrivers.EmbeddedTargets.AVR.Types.TimerNumber.A}
Which PWM outputs on the associated timer should be used (usually {A}, {B}, or {A,B})
initialValues
{0}
The value that is used to initialize the PWM before the first call to PWM.set
OK
Cancel

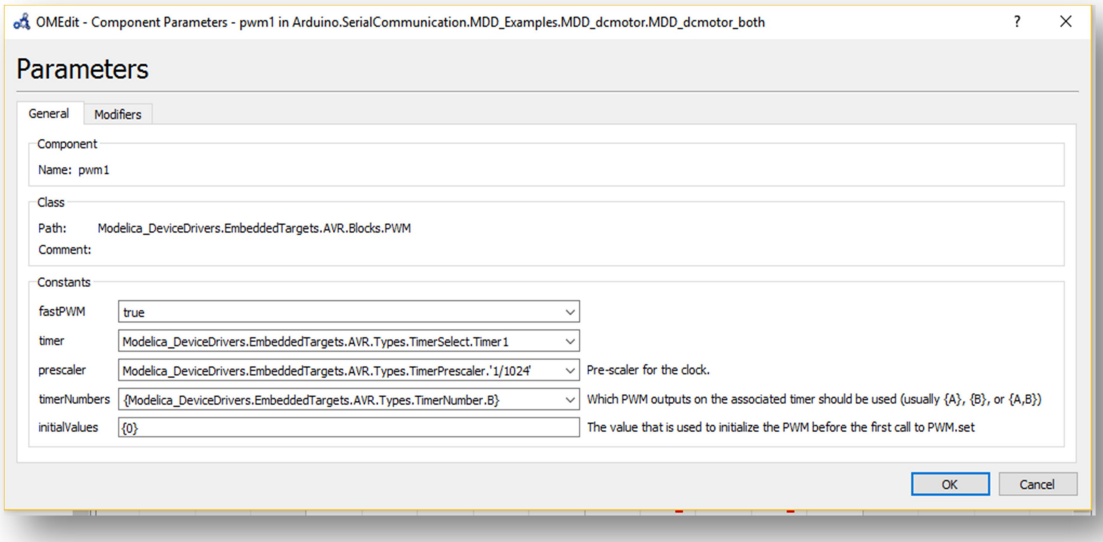


Fig 33.MDD_dcmotor_both

9. Experiments and Evaluation

The package has been tested with various examples included in the Examples package of the library. Experiments are done on LED, LDR, Potentiometer, Thermistor, DC Motor, Servo Motor etc. and are tested under varying parametric conditions. Testing is done on both Windows and Linux. Modelica_DeviceDrivers examples were also tested and expected outcomes were obtained.

10. Issues

On the course of our experimentation we were faced with a number of challenging issues some of which are still unresolved and are open for further development. The primary unresolved issues are illustrated below:

Unresolved Issues:

1. This package has support only for UART protocol and has not yet been extended to support Modbus protocol which has already been implemented in Scilab Arduino Toolbox.
2. Perfect synchronism has not yet been established in some cases due to difference in software structure and simulation properties of OpenModelica. It executes an algorithm multiple times within the stipulated time interval than the times specified by the user.
3. 'DigitalReadBoolean' block within AVR package in Modelica_DeviceDrivers library still shows anomaly. It returns default value 'false' at all times.
4. There is no graphical block which has the functionality of initialising a Servo Motor connected to arduino. The development of this block is still in progress.
5. Support for graphical visualization of models which uses Modelica_DeviceDrivers package has not been achieved due to limited functionality in its execution through command-line.

11. Conclusion

The project "Interfacing OpenModelica with Arduino", is based on calling C code from OpenModelica and its interaction with the firmware uploaded on Arduino. 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.

12. Bibliography

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://stackoverflow.com/
- https://www.openmodelica.org/
- http://book.xogeny.com/
- https://github.com/modelica/Modelica_DeviceDrivers
- http://andybrown.me.uk/2015/03/08/avr-gcc-492/
- https://build.openmodelica.org/Documentation/Modelica.html
- https://build.openmodelica.org/Documentation/Modelica_DeviceDrivers.html