Thanh Huynh – 2688093

Bilal Butt – 2688700

Michel Clerger – 2694646

Zirui Song - 2652528

Design document

GROUP MEMBERS:

Contents

[INTRODUCTION: 2](#_Toc472704239)

[IMPORTANT DECISIONS 3](#_Toc472704240)

[CLASS DIAGRAM 4](#_Toc472704241)

[Class diagram 4](#_Toc472704242)

[Description of the classes and their members 5](#_Toc472704243)

[HARDWARE RELATED CLASSES 5](#_Toc472704244)

[immediate CLASSES 10](#_Toc472704245)

[intelligence CLASSES 19](#_Toc472704246)

[STATE DIAGRAMs 22](#_Toc472704247)

[SEQUENCE DIAGRAMS 22](#_Toc472704248)

[UNIT TEST 22](#_Toc472704249)

[Problems and solution for environment setup using CodeBlock: 22](#_Toc472704250)

[Problem of installing and configuring the compiler: 22](#_Toc472704251)

[Problem with the data types for cross-complier the source code 25](#_Toc472704252)

[STUBHARDWARE CLASS 25](#_Toc472704253)

[TESTCOINWALLET CLASS 26](#_Toc472704254)

[TEstpgrogramSELECT CLASS 27](#_Toc472704255)

[PIN CONFIGURATIONS 28](#_Toc472704256)

# INTRODUCTION

This document is the design document for building an application that programs the simulation of a washing machine.

In the first section of this document, the class diagram of the application is presented in the form of diagrams along with their descriptions. The descriptions are divided into 3 sub-parts: hardware related class, intermediate classes which implementation the working logics of each hardware components the main classes for executions.

The second section presents some states diagrams for some of the important controlling processes.

The third section presents selected sequence diagrams for some of the more complex and important methods related to various classes.

The four section presents some configurations description of the pins.

# IMPORTANT DECISIONS

For our Laundry Machine program we began with the class template provided to us in the lab manual, however over the course of the past few weeks we made some significant changes to the overall design of the classes and by extension how they were implemented. For the most part we kept the hardware and various parent interface classes for hardware the same as they were in the template.

One thing we realized early in the project was that with the class hierarchy described in the template, we may end up with an overly bulky program executor class which would make implementation more complicated. Therefore one major decision our group made was to introduce an intermediate intelligent layer of classes which would bridge the gap between the hardware and the program executor class. There were several benefits to this decision, one was mainly that our program executor class was much smaller and easier to comprehend. Another equally important benefit of this was that we isolated the hardware capabilities in objects related to specific aspects of the hardware. For instance, the Motor object was an object belonging to the intelligent classes and would only handle hardware implementations related to the motor functionalities. This introduces a layer of isolation for various parts of the hardware even though we dealt with the same hardware.

The other form of digression from the template came in the form of how we executed the actual programs. The template had prescribed a callback mechanism with the function installStartHandler to implement the different program recipes. However, we chose another path and implemented the recipes in the start function of the program executor itself. The start function would receive a ProgramSetting pointer in its argument and based on that would determine which program washing recipe to execute.

There were also several functional decisions we had to make throughout the course of this project. One such decision was related to returning the money left over from a program when it was executed immediately after it began executing the program. We went for this decision mainly due to hardware limitations. There was a lack of LED’s on the coin interface of the hardware to adequately represent all possible amounts that could be left-over from when a program is executed.

# CLASS DIAGRAM

## Class diagram



Figure 1 - Class diagram

## Description of the classes and their members

By default, all the member variables of classes are private and their methods are public. If it is not the case, the visibilities of these properties and methods will be indicated. The class diagram is still subject to refinement and will be amended as deemed necessary.

The classes as depicted in the class diagram are based on the notion of how the Arduino itself operates. Since the Arduino does not have an operating system instead it operates by consistently running a main loop that continues indefinitely. Therefore, in the class hierarchy the LaundryMachine class plays the role of the class that contains the main loop.

The loop of the LaundryMachine class is constantly polling the ProgramSelect and CoinWallet classes to see if they have received any input from the user indicated by changes in their respective member variables. Once it has been determined if the program selected and the relevant payment has been completed and the user has indicated that he wants to begin the washing program, the ProgramExecutor will be called.

The ProgramExecutor executes the complete washing program for the washing program setting initially identified by the user. The interface classes are implemented in the HardwareControl class and are responsible for implementing functions that are related to the manipulation of the actual hardware.

## HARDWARE RELATED CLASSES

#### ICOIN

ICoin is the interface class containing the virtual functions which is handling the hardware inputs and outputs related to the detecting and showing the LEDs which are represented the amount of money given to the machine.

|  |  |
| --- | --- |
| ICOIN | |
| Functions | Description |
| *virtual boolean GetCoin10Button() = 0;* | This is used for getting the status of “Coin 10” button which indicates that a coin of value 10 is put into the machine. Return true if button is pressed, if not false is returned |
| *virtual boolean GetCoin50Button() = 0;* | This is used for getting the status of “Coin 50” button which indicates that a coin of value 50 is put into the machine. Return true if button is pressed, if not false is returned |
| *virtual boolean GetCoin200Button() = 0;* | This is used for getting the status of “Coin 200” button which indicates that a coin of value 200 is put into the machine. Return true if button is pressed, if not false is returned |
| *virtual void SetCoin10(unsigned char led) = 0;* | This is used for setting the LED (#coin10) corresponding to the given led parameter is put into the machine.  For example, if we call SetCoin10(B00000010), the second LED will turn on.  It also can use to set more than one LEDs and clear the LEDs.  For example, if we call SetCoin10(B00000111), all three LEDs will be turn on. And if we call SetCoin10(0x00) all the LEDs corresponding to #coin10 are off. |
| *virtual void SetCoin50(unsigned char led) = 0;* | This is used for setting the LED (#coin50) corresponding to the given led parameter is put into the machine.  Working principles are similar to SetCoin10. |
| *virtual void SetCoin200(unsigned char led) = 0;* | This is used for setting the LED (#coin200) corresponding to the given led parameter is put into the machine.  Working principles are similar to SetCoin10. |
| *virtual boolean GetClearButton() = 0;* | This is used for detecting that the Clear button is pressed or not. Return true if button is pressed, if not, false is returned. |
| Remarks:  All functions are implemented and working well.  In order to make the google test for the CoinWallet class. SetCoinX(byte led) (x is 10,50 or 200) was edited to become SetCoinX(unsigned char led). The unsigned char data type which is the C++ standard type.  The notation for the binary as “B000000001” is not accepted by others complier. It should be written as 0x01. | |

#### IPROGRAM

IProgram is the interface class containing the virtual which is handling the hardware inputs and outputs related to the detecting and showing which program is selected and it also contains the function to detect the pressing of start button.

|  |  |
| --- | --- |
| IPROGRAM | |
| Functions | Description |
| *virtual boolean GetStartButton() = 0;* | This is used for getting the status of start button. Return true if button is pressed, if not false is returned |
| *virtual boolean GetProgramButton() = 0;* | This is used for getting the status of program button. Return true if button is pressed, if not false is returned |
| *virtual void SetProgramIndicator(int programIndicator) = 0;* | This is used for setting on of 3 LEDS which are indicating the program A, B or C. We can SetProgramIndicator(B00000001) for selecting program A.  SetProgramIndicator(B00000010) for selecting program B.  SetProgramIndicator(B00000100) for selecting program C. |
| Remarks:  All function are implemented and working well | |

#### ILOCK

ILock is the interface class containing virtual functions which are implemented in the hardware class. It describes functions related to the hardware lock elements on board, which are mainly just one switch and one corresponding LED.

|  |  |
| --- | --- |
| ILOCK | |
| Functions | Description |
| *virtual boolean GetLockStatus() = 0;* | This is used for getting the status of the switch belonging to the Door Lock on the simulator board. It returns a true if switch is turned on else it returns a false. It will only yield meaningful return values when keyselect is low. |
| *void SetLockStatus(boolean lock)* | This function is used for setting the lock related LED on or off depending on the Boolean lock passed to the function. |
| Remarks: | |

#### ISOAP

ISoap is the interface class containing virtual functions which are implemented in the hardware class. This interface describes functions that are related to hardware that has to do with the soap elements on the board, namely the soap switches and its corresponding LEDs.

|  |  |
| --- | --- |
| ISOAP | |
| Functions | Description |
| *virtual boolean GetSoap1() = 0;* | This is used for getting the status of the switch belonging to the Soap 1 on the simulator board. It returns a true if switch is turned on else it returns a false. It will only yield meaningful return values when keyselect is low. |
| *virtual boolean GetSoap2() = 0;* | This is used for getting the status of the switch belonging to the Soap 2 on the simulator board. It returns a true if switch is turned on else it returns a false. It will only yield meaningful return values when keyselect is low. |
| *virtual void SetSoap1(boolean On) = 0;* | Will turn on the LED labelled Soap 1 on the simulation board if given argument true. Otherwise it will turn off LED for Soap 1 if given argument false. |
| *virtual void SetSoap2(boolean On) = 0;* | Will turn on the LED labelled Soap 2 on the simulation board if given argument true. Otherwise it will turn off LED for Soap 2 if given argument false. |
| Remarks: Function SetSoap2 still has some glitches because it is intended to be turned on by setting particular bits for Data and Group. However, it seems to not be working despite using the prescribed method of deactivating and activating strobe in order to turn on or off the Soap 2 LED in question. | |

#### IWATER

IWATER is the interface class containing the basic functions that will help to perform the basic actions related to the water part of the hardware class. It will allow to simulate the pins of the sink and of the drain of the water flowing in the machine.

|  |  |
| --- | --- |
| IWATER | |
| Functions | Description |
| *virtual boolean GetWater1() = 0;* | The purpose of this function will be to check if the pin of the Water1 is set or not by reading its value. |
| *virtual boolean GetWater2() = 0;* | The purpose of this function will be to check if the pin of the Water2 is set or not by reading its value. |
| *virtual void OpenSink() = 0;* | The purpose of this function will be to open the Sink. |
| *virtual void CloseSink() = 0;* | The purpose of this function will be to close the Sink. |
| *virtual void OpenDrain() = 0;* | The purpose of this function will be to open the Drain. |
| *virtual void CloseDrain() = 0;* | The purpose of this function will be to close the Drain. |
| Remarks: This interface is complete. All the functions are implemented. | |

#### IMOTOR

IMOTOR the interface class containing the basic functions of the motor part of the hardware class. They will allow the motor to turn in a certain direction and to turn on or off the different pins that allows us to set the speed that we want for the motor of the machine.

|  |  |
| --- | --- |
| IMOTOR | |
| Functions | Description |
| *virtual void TurnLeft() = 0;* | The purpose of this function will be to indicate the direction of the washing machine. In that function, it will turn to the left. |
| *virtual void TurnRight() = 0;* | The purpose of this function will be to indicate the direction of the washing machine. In that function, it will turn to the right. |
| *virtual void StartSpeed1() = 0;* | The purpose of this function will be to start the Speed1 of the motor. |
| *virtual void StartSpeed2() = 0;* | The purpose of this function will be to start the Speed2 of the motor. |
| *virtual void StopSpeed1() = 0;* | The purpose of this function will be to stop the Speed1 of the motor. |
| *virtual void StopSpeed2() = 0;* | The purpose of this function will be to stop the Speed2 of the motor. |
| Remarks: This interface is complete. All the functions are implemented. | |

#### Itemperature

ITemperature is the interface class containing the virtual functions relating which is implemented in the hardware class.

|  |  |
| --- | --- |
| ITEMPERATURE | |
| Functions | Description |
| *virtual void SetHeater(bool sw1tch) = 0;* | To set the heater on or off. |
| *virtual int GetTemperature() = 0;* | To get the current temperature. |
| Remarks:  “sw1tch” is to avoid conflict with C “switch” statement | |

#### IBuzzer

IBuzzer is the interface class containing the virtual functions relating which is implemented in the hardware class.

|  |  |
| --- | --- |
| IBUZZER | |
| Functions | Description |
| *virtual void SetBuzzer(int ms) = 0;* | To let the buzzer make noise for a period of milliseconds. |
| Remarks:  None | |

#### HARDWARE CLASS

Hardware class implements all functions from interface classes. However, it also has its own functions which are describe as following functions.

|  |  |
| --- | --- |
| HARDWARE | |
| Functions | Description |
| *void SetKeySelect(int value);* | Setting Keyselect to 0 or 1 |
| *void SetGroup(int group);* | Setting value for all combinations value from group pins. This function is used in implications of setting output for coin indicators or program indicators. |
| *void SetData(int data);* | Setting value for all combinations value from data pins. This function is used in implications of setting output for coin indicators or program indicators. |
| *void Strobe();* | Inactive stroke for 100ms and then active for 15ms. |
| Remarks: | |

## intermediate CLASSES

#### COINWALLET CLASS

CoinWallet is the class contains the functions and variables for handling coins and the amount of money that users put into the machine

|  |  |  |
| --- | --- | --- |
| Properties | | |
| nrOfCoin10 | Integer | This is an instance variable to indicate how many coins of value 10 which is put by users |
| nrOfCoin50 | Integer | This is an instance variable to indicate how many coins of value 50 which is put by users |
| nrOfCoin200 | Integer | This is an instance variable to indicate how many coins of value 200 which is put by users |
| Balance | Integer | The current amount of money in the wallet |
| mCoin | ICoin \* | A pointer with point to an ICoin object. It is used to reference to methods related to coin functionalities in hardware class |
| Operations | | |
| CoinWallet(ICoin \*) : *constructor of the class* | | |
| mappingCoin(char nrofCoin) : char  *This is used to map the nrOfCoinX variable to become a level of bits so that the hardware class is able to handle it.* | | |
| Withdraw(int amount): Boolean  *This is used to for withdrawing the amount of money indicated by given parameters. If the amount is less than or equal the balance, the amount of money is withdrawn and this functions return true. Otherwise, false is return.* | | |
| AddCoin10(): void  *Is called in the poll function to always detect weather the button to put coin 10 is pressed or not. If it is, increase the nrOfCoin10 by 1 (only allow it up to 3) and turn on the corresponding LED to indicate the coin is successfully put.* | | |
| AddCoin50(): void  *Is called in the poll function to always detect weather the button to put coin 50 is pressed or not. If it is, increase the nrOfCoin50 by 1 (only allow it up to 3) and turn on the corresponding LED to indicate the coin is successfully put.* | | |
| AddCoin200(): void  *Is called in the poll function to always detect weather the button to put coin 200 is pressed or not. If it is, increase the nrOfCoin200 by 1 (only allow it up to 2) and turn on the corresponding LED to indicate the coin is successfully put.* | | |
| WithdrawAll() : void  *Withdraw all the current money in the wallet. Then the balance is set into 0.* | | |
| ReturnChange(): void  *Return the remain money after washing program cost is subtracted to user. All the LEDs with is indicated the amount of money are turned off.* | | |
| SetInterface(ICoin \*): void  *Set the new ICOIN inferface object of the CoinWallet class.* | | |
| GetAmount() : int  *Return the current balance in the coin wallet.* | | |
| Poll(): void  *This function is called in the main loop for polling all functions related to coin mentioned above.* | | |
| Remarks:  All functions are implemented.  This class is tested with google test.  ReturnChange function is added comparing with the previous version. | | |

#### programselect CLASS

ProgramSelect is the class contains the functions and variables for handling program select.

|  |  |  |
| --- | --- | --- |
| Properties | | |
| currentProgram | Integer | This is an instance variable to indicate the current program |
| mProgram | IProgram \* | An pointer with point to an IProgram object. It is used to reference to methods related to coin functionalities in hardware class |
| (\* mStartHandler)() | Void | Function pointer which points to the start function in program executor class. |
| Operations | | |
| ProgramSelect(): *constructor of the class. In the constructor, currentProgam is initialized as 1 for washing program A.* | | |
| ProgramSelect (IProgram \*): *constructor of the class with IProgram as a input parameter.* | | |
| GetProgramType(): char  *This is used to get the type of the program selected by detecting the program select button.* | | |
| InstallStartHandler(void (\* handler)()) : void  *This is used to pass the address of function pointer “handler” to the “mStartHandler” pointer.* | | |
| setProgramInterface(int value): void  *This is used to control the hardware to change the program* | | |
| Poll(): void  *This function is called in the main loop for polling all the time which program is currently selected and call the function that “mStartHandler” point to and execute it.* | | |
| Remarks:  All functions are implemented.  This class is tested with google test.  In the final implementation, the InstallStartHandler is not called from ProgramExecutor, but it is still kept there for the possibility of another scenario. | | |

#### LOCK CLASS

The Lock class contains the functions and variables for handling the lock interface and its associated hardware controls. This class is part of the intelligent layer of classes and allows interaction with the lock related hardware and also retain various important information content related to the lock.

|  |  |  |
| --- | --- | --- |
| Properties | | |
| iLock | ILock \* | iLock is a pointer to a ILock type interface which is implemented in the HardwareControl class. The iLock will essentially point to a HardwareControl object which it can use to interact with the hardware IO. |
| Lock | Boolean | The lock is a Boolean property that describes the status of the lock switch on the board and needs to be otherwise true in order to run a washing program. |
| Operations | | |
| Lock(): *default constructor of the class* | | |
| Lock(ILock \*) : *constructor of the class which takes an ILock pointer and assigns it to iLock* | | |
| lockMachine() : void  *This is the polling function to check what the lock switch status is and will turn on or turn off the lock.* | | |
| checkLock(): Boolean  *Returns status of lock switch on the board.* | | |
| setLock(Boolean ): void  *Is the setter function for lock property.* | | |
| setInterface(ILock \*) : void  *Assigns the pointer to ILock object (actually HardwareControl object) in its argument to the iLock property.* | | |
| Remarks:  All functions are implemented. The classes are yet to be tested. | | |

#### SOAP CLASS

The Lock class contains the functions and variables for handling the lock interface and its associated hardware controls. The soap class is also part of the intelligent layer of classes and is used as an intermediary class between the soap related hardware and higher class layers, which is primarily the ProgramExecutor.

|  |  |  |
| --- | --- | --- |
| Properties | | |
| iSoap | ISoap \* | iSoap is a pointer to an ISoap type interface which is implemented in the HardwareControl class. The iSoap will essentially point to a HardwareControl object which it can use to interact with the hardware IO. |
| soapCpt1 | Boolean | The soapCpt1 is a Boolean property that describes the status of the soap compartment 1 switch on the board and needs to be otherwise true in order to run a washing program. |
| soapCpt2 |  | The soapCpt2 is a Boolean property that describes the status of the soap compartment 2 switch on the board and needs to be otherwise true in order to run a washing program. |
| Operations | | |
| Soap(): *default constructor of the class* | | |
| Soap(ISoap \*) : *constructor of the class which takes an ISoap pointer and assigns it to iSoap* | | |
| checkCpt1(): Boolean  *This is the polling function that will check if soap switch 1 is turned on and will turn on soap 1 LED accordingly.* | | |
| checkCpt2(): Boolean  *This is the polling function that will check if soap switch 2 is turned on and will turn on soap 2 LED accordingly.* | | |
| lockCpt1(Boolean): void  *Is the setter function for soapCpt1 property, and will turn the soap 1 LED on or off according to the Boolean argument provided.* | | |
| lockCpt2(Boolean): void  *Is the setter function for soapCpt2 property, and will turn the soap 2 LED on or off according to the Boolean argument provided.* | | |
| setInterface(ISoap \*) : void  *Assigns the pointer to ISoap object (actually HardwareControl object) in its argument to the iSoap property.* | | |
| Remarks:  All functions are implemented. The classes are yet to be tested. | | |

#### Water CLASS

Water is the intermediate class that contains the functions and variables that are necessaire for the water system of the laundry machine. It allows us to sink water, to drain water and to also set the level of water that we need for our different washing processes.

|  |  |  |
| --- | --- | --- |
| Properties | | |
| Full | Constant Integer | Constant integer which will be used to indicate the full level of the water contained in the washing machine. |
| Medium\_water | Constant Integer | Constant integer which will be used to indicate the medium level of the water contained in the washing machine. |
| Low\_water | Constant Integer | Constant integer which will be used to indicate the low level of the water contained in the washing machine. |
| Empty | Constant Integer | Constant integer which will be used to indicate the empty level of the water contained in the washing machine. |
| oWater | IWater \* | A pointer with point to an IWater object. It is used to reference to methods related to water functionalities in hardware class |
| Operations | | |
| Water (IWater \*): *constructor of the class* | | |
| CheckLevel(): int  *This function is used to check the current level of the water in the washing machine* | | |
| SetLevel(int level): void  *This function is used to set the water level at a certain level. We can pass the desired level as the value of the parameter of that function, to be set as the water level.* | | |
| SetSink(boolean state):void  *This function is used to set the state of the Sink whether ON of OFF. We can pass the desired state as the value of the parameter of that function, to be set as the state of the Sink.* | | |
| SetDrain(boolean state): void  *This function is used to set the state of the Drain whether ON of OFF. We can pass the desired state as the value of the parameter of that function, to be set as the state of the Drain.* | | |
| ~Water()  *This is the destructor of the object.* | | |
| setInterface(IWater \*) : void  *Assigns the pointer to IWater object in its argument to the iWater property.* | | |
| Remarks:  All functions are implemented. | | |

#### Motor CLASS

Motor is the intermediate class that contains the functions and variables that are useful for the motor of the laundry machine. It allows to control the motor’s actions in a more ordered manner. This class allow us to either start or stop the motor, to set its rotating direction, and to also set its speed at will.

|  |  |  |
| --- | --- | --- |
| Properties | | |
| High | Constant Integer | Constant integer which will be used to indicate the high speed of the motor of the washing machine. |
| Medium | Constant Integer | Constant integer which will be used to indicate the medium speed of the motor of the washing machine. |
| Low | Constant Integer | Constant integer which will be used to indicate the low speed of the motor of the washing machine. |
| OFF | Constant Integer | Constant integer which will be used to indicate when we want to stop the motor. Therefore, the speed is 0 at that time. |
| Speed | Integer | Current speed of the motor. |
| Direction | Boolean | Boolean which we be used to set the direction of the washing machine. Left will be for 0 and Right will be for 1. |
| oMotor | IMotor \* | A pointer with point to an IMotor object. It is used to reference to methods related to motor functionalities in hardware class |
| Operations | | |
| Motor (IMotor \*): *constructor of the class* | | |
| Start(int s): void  *This function is used to set the speed of the Motor. We can pass the desired speed as the value of the parameter of that function, to be set as the current speed of the motor.* | | |
| Stop():void  *This function is used to stop the motor.* | | |
| SetDirection(boolean dir): void  *This function is used to set the direction of the washing machine. We can pass the desired direction of the wash as the parameter of that function, to be set as the direction of the washing machine.* | | |
| rotateLM(boolean dir, int Speed): void  *This function is used to rotate the motor at a certain speed and a chosen direction. We can pass the desired speed and the desired direction as the values of the parameters of that function, to be set as the current speed and direction of the motor.* | | |
| setInterface(IMotor \*) : void  *Assigns the pointer to IMotor object in its argument to the iMotor property.* | | |
| ~Motor()  *This is the destructor of the object.* | | |
| Remarks:  All functions are implemented. | | |

#### Buzzer CLASS

Buzzer is the class contains the functions and variables for handling the buzzer.

|  |  |  |
| --- | --- | --- |
| Properties | | |
| Buzzer | IBuzzer \* | An pointer with point to an IBuzzer object. It is used to reference to methods related to buzzer functionalities in hardware class |
| Operations | | |
| Buzzer(IBuzzer \*) : *constructor of the class* | | |
| SingleBuzzer(int duration) : void  *This is used to let buzzer beep once for a specific duration (in ms).* | | |
| MultipleBuzzer(int duration) : void  *This is used to let buzzer beep for specific times, each beep in specific duration (in ms) and pauses in same duration between every two beeps.* | | |
| Remarks:  All functions are implemented. The unit test environment and implementation of the unit test are setting up | | |

#### temperature CLASS

Temperature is the class contains the functions and variables for handling heater and the temperature of the water in the machine

|  |  |  |
| --- | --- | --- |
| Properties | | |
| currentTemperature | Integer | This is an instance variable to indicate the current temperature inside |
| desiredTemperature | Integer | This is an instance variable to indicate the temperature that the program desires |
| Temp | ITemperature \* | A pointer with point to an ITemperature object. It is used to reference to methods related to coin functionalities in hardware class |
| Operations | | |
| Temperature(ITemperature \*) : *constructor of the class* | | |
| GetDesired() : int  *This is used to get the desired temperature* | | |
| CheckTemperature() : int  *This is used to get the current temperature* | | |
| SetTemperature(int value): void  *This is used to control the heater to set and keep the temperature to the desired value* | | |
| Poll(): void  *This function is called in the main loop for polling all functions related to coin mentioned above.* | | |
| Remarks:  All functions are implemented. The unit test environment and implementation of the unit test are setting up | | |

# intelligence CLASSES

#### PROGRAMEXECUTOR CLASS

The Program Executor class contains the functions and variables for handling the intermediate classes that interact with various hardware control interfaces.

|  |  |  |
| --- | --- | --- |
| Properties | | |
| mProgramSettings | ProgramSettings \* | mProgramSettings is a pointer to an ProgramSettings type object which contains information on the various programs available and their associated costs. |
| mCoinWallet | CoinWallet \* | mCoinWallet is a pointer to an CoinWallet type object which contains functions and variables related to the amount of money the user puts in the laundry machine. |
| mWater | Water | mWater is an object of type Water which contains various variables and functions related to controlling the water-related hardware of the machine. |
| mTemperature | Temperature | mTemperature is an object of type Temperature which contains various variables and functions related to controlling the heater and temperature related hardware of the machine. |
| mSoap | Soap | mSoap is an object of type Soap which contains various variables and functions related to controlling the two soap compartments and their related hardware on the machine. |
| mLock | Lock | mLock is an object of type Lock which contains various variables and functions related to controlling the lock of the machine and its related hardware. |
| mMotor | Motor | mMotor is an object of type Motor which contains various variables and functions related to controlling the motor and its related hardware of the machine. |
| mBuzzer | Buzzer | mBuzzer is an object of type Buzzer which contains various variables and function related to controlling the buzzer and its related hardware of the machine. |
| Operations |  |  |
| Soap(): *default constructor of the class* | | |
| ProgramExecutor(IBuzzer\* b, IMotor\* m, ILock\* l, ISoap\* s, ITemperature\* t, IWater\* w) : *constructor of the class which takes an ISoap, ILock, IMotor, IBuzzer, Itemperature and IWater pointer and assigns it to the respective Lock, Buzzer, Motor, Soap, Temperature and Water objects* | | |
| Start(ProgramSettings \*) : boolean  *This function assigns the program function object in its arguments to the mProgramSettings variable.* | | |
| StepSwitches() : boolean  *This is the polling function that reads the soap and lock switches and turns the corresponding LED’s accordingly.* | | |
| StepCoinWallet() : Boolean  *Calls the polling function of its mCoinWallet object to read and interpret the button presses for the coins and turn on or off the corresponding LED’s.* | | |
| IsReady(char) : Boolean  *Resets the program for ProgramSettings to the one provided to it in its arguments. It then compares the amount of money in its CoinWallet object against the cost of the program of the ProgramSettings objects and returns a true if the money is sufficient or a false otherwise.* | | |
| setCoinWallet(CoinWallet\* ): void  *Assigns the pointer in its argument to the mCoinWallet property.* | | |
| stopDelay(int Speed): void  This is a delay function to be used when stopping the motor and changing direction. This is because the motor needs some time to stop before it can change direction, and the delay will vary depending on the current speed of the motor, which is given as the argument of that function. | | |
| Centrifugate(char prog): void  This function performs the tasks of the centrifugation recipe for each of the programs as provided in its argument. The centrifugation is described in the lab manual for each program and involves spinning the laundry at high speeds for some time. | | |
| Prewash(char prog):void  This function performs the tasks of the pre-wash recipes for each given program in its argument as provided by the lab manual. | | |
| Mainwash\_Phase1(char prog): void  This function performs the tasks of the main wash part 1, depending on the program given in its argument as provided in the recipe described in the lab manual. | | |
| Mainwash\_Phase2(char prog): void  This function performs the tasks of the main wash part 2, depending on the program given in its argument as provided in the recipe described in the lab manual. | | |
| DoFullRotating(int NbrOfTimes, int Speed, int DelayVal):void  This function rotates the motor first in clockwise then in counter-clockwise direction at the speed given in the argument called ‘Speed’, and during a given amount of time that we can indicate by setting the value of its argument called ‘DelayVal’. Its first arguments called ‘NbrOfTimes’ is used to indicate the number of times that we want the motor to perform that same full rotation. | | |
| tempDelay(int d):void  The function ‘tempDelay’ replaces the regular delay in that it breaks up a regular delay function into smaller intervals during which it can poll the temperature and turn the heater on or off according to what the desired temperature has been set. The total delay lasts for the amount of time ‘d’ given in milliseconds. | | |
| Remarks:  All functions are implemented. The classes are yet to be tested. | | |
|  | | |

#### PROGRAMSETTING CLASS

ProgramSetting is the class contains the functions and variables for handling the setting for washing programs

|  |  |  |
| --- | --- | --- |
| Properties | | |
| Type | Char | A char to indicate the program |
| Cost | Int | The cost of each program |
| Operations | | |
| ProgramSettings(char): *constructor of the class* | | |
| GetProgramType() : char  *This is used to get the type of the program* | | |
| GetProgramCost() : int  *This is used to get the cost of the program* | | |
| ToString() : String  *This is used to transfer the object to string* | | |
| Remarks:  All functions are implemented. | | |

# STATE DIAGRAMs

It will be updated later.

# SEQUENCE DIAGRAMS

It will be updated later.

# UNIT TEST

## Problems and solution for environment setup using CodeBlock:

### Problem of installing and configuring the compiler:

First, trying to install GCC complier with MinGW package from following link <http://www.mingw.org/category/wiki/download> for Codeblocks and do the following set up as Figure 2. MinGW includes a port of the GNU Compiler Collection (GCC), including C, C++ for Window environment.

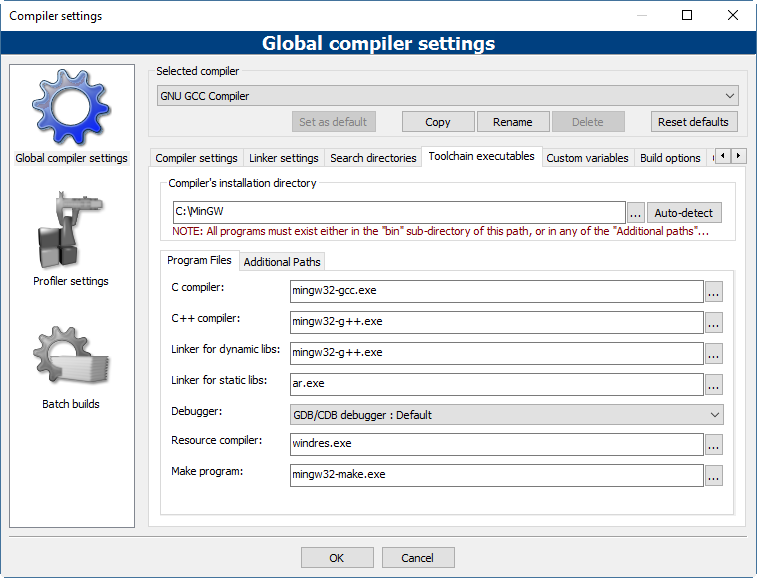


Figure 2 - Compiler setting with MinGW

This setting turned out some following errors on the gtest-port.h

* *Error: ‘fileno’ was not declared in this scope*
* *Erorr: \_stricmp is not declared in this scope*

To get rid of these errors, one of the suggested solution is changing the -std=c\*\*\* flag -std=gnu++0x should fix your problem. In Codeblocks, it is done by adding new compiler flags. It is done by going to the ***compiler Flags*** tag then right click on the area that the Fag will be added and choose “New flag”, then give the flag’s information. For more details, it can be done by reference on this link: <http://stackoverflow.com/questions/31171979/enabling-std-c14-flag-in-codeblocks>

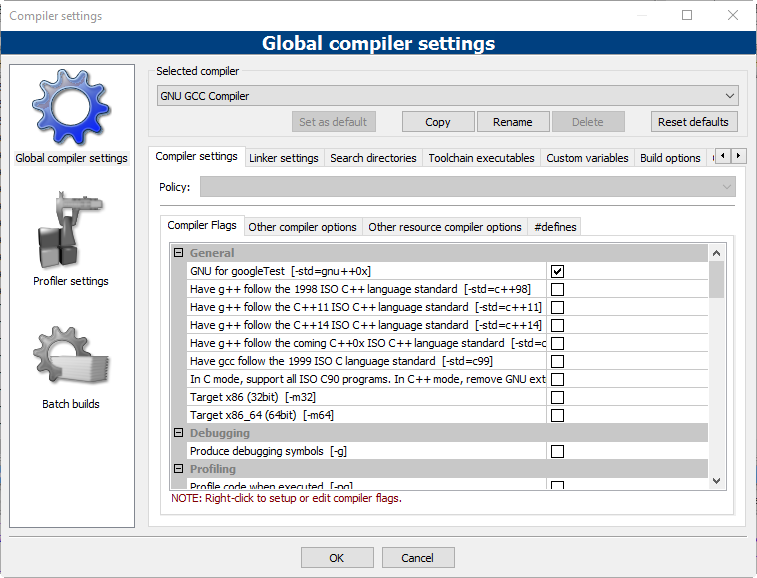


Figure 3 - New flag is added

However, this above solution still left the error with *\_stricmp which is indicated as following*

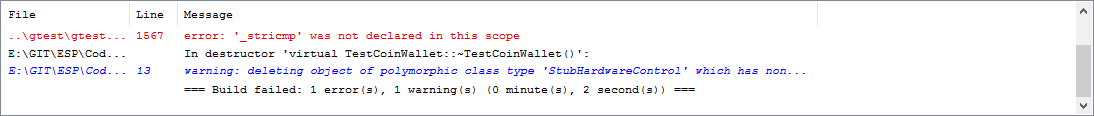


Figure 4 - \_strcmp error

It could be the configuration for minGW was not correct. However, we cannot indicated what is going wrong, since the minGW provided many options to install the compiler. Then, an alternative solution was tried. Besides minGW, one of the tutorial (<http://wiki.codeblocks.org/index.php/MinGW_installation>) of Codeblocks guides us to use TDM-GCC which is the easier way to set up a working compiler is to install a recent GCC release targeting the MinGW compiler system. The following settings (Figure 5) for the compiler on Codeblocks worked.

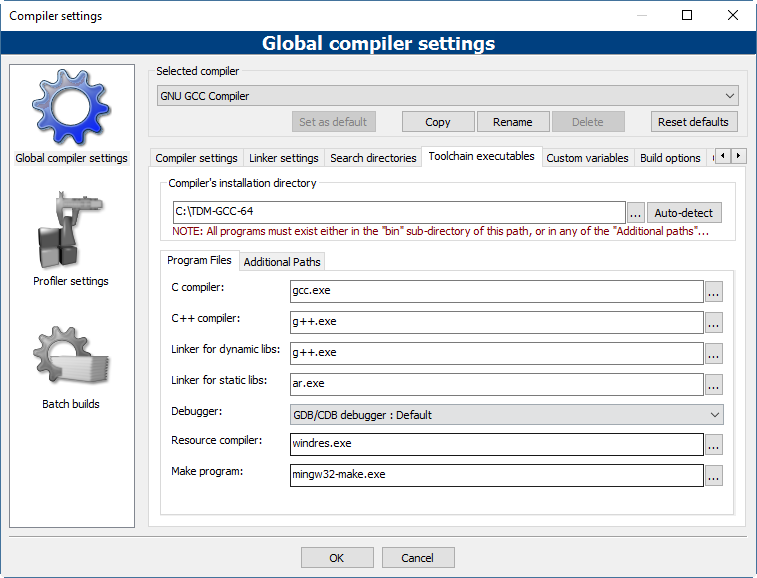


Figure 5 - Configuration of CodeBlock compiler with TDM-GCC

### Problem with the data types for cross-complier the source code

When the code was implemented on Arduino platform, we did use the Byte data type which is only work on this platform but not for the others one. This leads to a problem when we would like to implement the test for google test, it did not work properly. Since, CodeBlock or Linux environment required some extra libraries to make it complies and currently, CodeBlock does not support the cross-complier for Arduino platform. One of the possibilities is used “typedef” to define the type Byte as a trick to make it compile in CodeBlock as well as Linux environment. However, changing the Byte type to one of the standard types of C++ such as unsigned char is an implemented solution.

After changing the data type back to the C++ standard types, the classes can be run in both Codeblock and Linux environments.

### STUBHARDWARE CLASS

Stubhardware is stub of the class HardwareControl which simulates the behaviors of the real Hardware class. This class contains all the functions that the HardwareControll class has. However, the Stubharware’s functions only return value or have the empty body in term of void function. These functions are used to make sure that the test can work in the condition of all hardware components work well. It also turns out that if any errors occur, it will be the problem of the upper class (CoinWallet or PropramSelect) and it is easier to maintain and debug the errors.

### TESTCOINWALLET CLASS

Testcoinwallet class is the implementation of unit tests for class CoinWallet. This contains 17 tests which mainly check for the amount of the money when users put the number of coins into the laundry machine. It also checks for the balance when one of washing programs is executed and the corresponding amount of money is subtracted. The following table lists these tests in detail.

|  |  |
| --- | --- |
| Tests | Explanations |
| test\_getBalance | Test the getBalance method in the term of the wallet does not have any coins and the expected balance is 0 |
| test\_coin10\_one | Test the balance of the coin wallet when 1 coin of 10 is added, the expected balance is 10 |
| test\_coin10\_two | Test the balance of the coin wallet when 2 coins of 10 are added, the expected balance is 20 |
| test\_coin10\_three | Test the balance of the coin wallet when 3 coins of 10 are added, the expected balance is 30 |
| test\_coin10\_four | Test the balance of the coin wallet when 4 coins of 10 are added. Since the machine only allows 3 coins of 10 are put. Then when the fourth one is put, in real situation, it is returned to the user and the expected balance is 30 |
| test\_coin50\_one | Test the balance of the coin wallet when 1 coin of 50 is added, the expected balance is 50 |
| test\_coin50\_two | Test the balance of the coin wallet when 2 coins of 50 are added, the expected balance is 100 |
| test\_coin50\_three | Test the balance of the coin wallet when 3 coins of 50 are added, the expected balance is 150 |
| test\_coin50\_four | Test the balance of the coin wallet when 4 coins of 50 are added. Since the machine only allows 3 coins of 50. Then when the fourth one is put, in real situation, it is returned to the user and the expected balance is 150 |
| test\_coin200\_one | Test the balance of the coin wallet when 1 coin of 200 is added, the expected balance is 200 |
| test\_coin200\_two | Test the balance of the coin wallet when 2 coins of 200 are added, the expected balance is 400 |
| test\_coin200\_three | Test the balance of the coin wallet when 3 coins of 200 are added. Since the machine only allows 2 coins of 200. Then when the fourth one is put, in real situation, it is returned back to the user and the expected balance is 400 |
| test\_balance\_programA | Test the balance of 360, which is the price for program A, by adding 1 coin of 200, 3 coins of 50 and 1 coin 10. The expected balance is 360. |
| test\_balance\_programB | Test the balance of 480, which is the price for program B, by adding 2 coins of 200, 1 coin of 50 and 3 coins of 10. The expected balance is 480. |
| test\_balance\_programC | Test the balance of 510, which is the price for program C, by adding 2 coins of 200, 1 coin of 50 and 3 coins of 10. The expected balance is 510. |
| test\_withdraw\_all | Test for calling the WithDrawAll function in the CoinWallet class. The expected balance is 0. |
| test\_withdraw | Test for calling the WithDraw function in the CoinWallet class, which the given amount of withdraw money is 510 and the balance is 550. Then, the expected balance is 40. |

### TEstpgrogramSELECT CLASS

Testcoinwallet class is the implementation of unit tests for class ProgramSelect. This contains 6 tests which mainly check for which washing program is selected when the program button is pressed.

|  |  |
| --- | --- |
| Tests | Explanations |
| getProgramA | By default, the selected program is A. This test called the GetProgramType of ProgramSelectClass and expected result is washing program “A”. |
| getProgramB | This test calls the Poll function of the ProgramSelectClass. The StubHardware class is built based on the assumption that all the reading input from buttons, when they are called, will return true. It means that, when the Poll function called the program button is pressed. Calling the Poll function one time means that the button is pressed one time and the expected result is washing program “B”. |
| getProgramC | The testing concept of this test is similar to getProgramB. However, the Poll function is called twice and the expected result is washing program “C”. |
| getProgram\_backtoA | This test applied the same concept as getProgramB and getProgramC. However, after we get “C”, the new circle is started and back again to “A” if the poll function is called 3 times and expected result is washing program “A”. |
| getProgram\_backtoB | This test applied the same concept as getProgram\_backtoA. However, the poll function is called 4 times and the expected result is washing program “B” |
| getProgram\_backtoC | The same concept as getProgram\_backtoA and getProgram\_backtoB is applied. In this case, the poll function is called 5 time and the expected result is washing program “C”. |

# PIN CONFIGURATIONS

From our independent investigations, we tried to determine which buttons correspond to which exact input descriptors. In the hardware control class the constants used for each of the corresponding buttons and switches were as follows:

Keyselect must be HIGH in order to read button inputs.

Button 1: IN\_IN0 and IN\_IN3, pressing button 1 raises IN0 and IN3 to high  
Button 2: IN\_IN0, pressing button 2 raises IN0 to high  
Button 3: IN\_IN3, pressing button 3 raises IN3 to high  
Button 4: IN\_IN2, pressing button 3 raises IN2 to high  
Button 5: IN\_IN1, pressing button 3 raises IN1 to high  
Button 6: IN\_IN1, IN\_IN2 and IN\_IN3, pressing button 1 raises IN1, IN2 and IN3 to high

Keyselect must be LOW in order to read switch inputs.

Switch 2: IN\_IN0, pulling up switch 2 raises IN0 to high  
Switch 3: IN\_IN3, pulling up switch 3 raises IN3 to high  
Switch 4: IN\_IN1, pulling up switch 4 raises IN1 to high  
Switch 5: IN\_IN2, pulling up switch 5 raises IN2 to high

*Figure 1: Buttons and Switches on Laundry Machine*