Thanh Huynh – 2688093

Bilal Butt – 2688700

Michel Clerger – 2694646

Zirui Song - 2652528

Design document

GROUP MEMBERS:

Contents

[CLASS DIAGRAM 2](#_Toc468475422)

[Class diagram 2](#_Toc468475423)

[Description of the classes and their members 2](#_Toc468475424)

[HARDWARE RELATED CLASSES 2](#_Toc468475425)

[STATE DIAGRAMs 7](#_Toc468475426)

[PIN CONFIGURATIONS 9](#_Toc468475427)

# 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, immediate class 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.

# CLASS DIAGRAM

## Class diagram

We are working on. It will be updated.

## Description of the classes and their members

By default, all of the properties of classes are private and methods of them are public. If it is not the case, the visibilities of these properties and methods will be indicated.

By default, all of 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 have to do with manipulating the actual hardware.

## HARDWARE RELATED CLASSES

#### ICOIN

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

|  |  |
| --- | --- |
| 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(byte 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(0xB00000111), all three LEDs will be turn on. And if we call SetCoin10(0x00) all the LEDs corresponding to #coin10 are off. |
| *virtual void SetCoin50(byte 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(byte 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.  Detecting the clear button is done by getting in3 & in2 & in1 == HIGH. Using delay and redetect the input signal for preventing the button is keeping pressed. However, it seems did not work properly. From testing, it turns out that the clear button need to be keeping pressing until all LEDs are off. | |

#### IPROGRAM

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

|  |  |
| --- | --- |
| 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.  By testing and the information from the LAB manual, we know that program button is detected when in3 OR in0 == HIGH and start button is detected when in0 == HIGH. Should it be better if we only detect program button if in3 == HIGH? | |

#### ILOCK

ILock is the interface class containing virtual functions which are implemented in the hardware class.

|  |  |
| --- | --- |
| 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. |
| Remarks: | |

#### ISOAP

ISoap is the interface class containing virtual functions which are implemented in the hardware class.

|  |  |
| --- | --- |
| 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 virtual functions relating which is implemented in the hardware class.

|  |  |
| --- | --- |
| IWATER | |
| Functions | Description |
| *virtual void SetWaterLevel(int level) = 0;* | The purpose of this function will be to set the appropriate water lever for the washing process. It takes one parameter for the level that we want to check. |
| *virtual boolean CheckWaterLevel(int level) = 0;* | The purpose of this function will be to check if the washing machine has reached the desired water level. It takes one parameter for the level that we want to check. |
| *virtual void SinkWater() = 0;* | The purpose of this function will be to |
| Remarks: This interface is complete. All the functions are implemented.  To discuss: I just wonder if we need any more functions for this interface or are those 3 enough? | |

#### IMOTOR

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

|  |  |
| --- | --- |
| IMOTOR | |
| Functions | Description |
| *virtual void SetDirection(char dir) = 0;* | The purpose of this function will be to indicate the direction of the washing machine. It takes one parameter to allow us to indicate the direction that we want to set. |
| *virtual void SetSpeed(int level) = 0;* | The purpose of this function will be to indicate the speed of the washing machine. It takes one parameter to allow us to indicate the speed that we want to set. |
| *virtual void StopMotor() = 0;* | The purpose of this function will be to stop the motor. |
| *virtual void StartMotor() = 0* | The purpose of this function will be to start the motor at an initial state assuming that all the prerequisites are handled properly within the IProgram interface. |
| *virtual void CheckLoadingLevel(int level) = 0;* | The purpose of this function will be to check le loading level of the washing machine. It takes one parameter for the level that we want to check. |
| Remarks: This interface is complete. All the functions are implemented.  To discuss: To start the motor can we just set the initial speed, or we also should consider some other states? Should we take care of the prerequisites in this function of in the IProgram interface? | |

#### Itemperature

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

|  |  |
| --- | --- |
| ICOIN | |
| Functions | Description |
| *virtual void SetTemperature(int level) = 0;* | To set the heater to heat to the specified temperature. |
| Remarks:  Implemented, not tested | |

#### IBuzzer

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

|  |  |
| --- | --- |
| ICOIN | |
| Functions | Description |
| *virtual void SetBuzzer(int ms) = 0;* | To let the buzzer make noise for a period of milliseconds. |
| Remarks:  Not implemented to the HardwareControl.cpp | |

#### 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: | |

## immediate 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 a instance variable to indicate how many coin of value 10 which is put by users |
| nrOfCoin50 | Integer | This is a instance variable to indicate how many coin of value 50 which is put by users |
| nrOfCoin200 | Integer | This is a instance variable to indicate how many coin of value 200 which is put by users |
| balance | Integer | The current amount of money in the wallet |
| mCoin | ICoin \* | An 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 for 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 of the current money in the wallet. Then the balance is set into 0.* | | |
| 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.* | | |

# STATE DIAGRAMs

# SEQUENCE DIAGRAMS

# 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*