System Design Document

ECSE 421 - Embedded Systems

Green house system

Group 1

Sary Amatoury (260171383)
Carlos Asmat (260148251)
Yevgeniy Goldenberg (260191384)
François Grondin (260188420)
Stephen Hopkins (260044918)
David Lopez (260146414)
Ritvik Mudur (260191555)
Maram Qouqa (260142490)

March 11, 2008

Table of contents

Page

1. Introduction 1

2. Deployment 2

Figure 2.0.0.0: Deployment diagram 2

3. Traceability matrix 3

Table 3.0.0.0: Overview of the traceability matrix for the system 3

Table 3.0.0.1: Traceability matrix for the system 4

4. System hardware 8

Table 4.0.0.0: Color codes used for controllers 8

4.1 Real-time clock timer 8

Figure 4.1.0.0: Inputs and outputs of the real-time clock timer 8

Figure 4.1.0.1: Normal mode timing 9

Figure 4.1.0.2: Update mode timing 9

Figure 4.1.0.3: Finite-state machine for the real-time clock timer 10

4.2 Real-time clock divider 11

Figure 4.2.0.0: Inputs and outputs of the real-time clock divider 11

Figure 4.2.0.1: Logic circuit of the real-time clock divider 11

4.3 Motion decoder 12

Figure 4.3.0.0: Inputs and outputs of the motion decoder 12

Figure 4.3.0.1: Subcontrollers of the motion decoder 12

Figure 4.3.0.2: Structure of each subcontroller inside the motion decoder 13

4.4 Light controller 14

Figure 4.4.0.0: Inputs and outputs of the light controller 14

Figure 4.4.0.1: Subcontrollers of the light controller 14

4.4.1 Subcontroller A: Bedrooms and bathroom room lights 16

Figure 4.4.1.0: Structure of each room light subcontroller for bedrooms and bathroom 16

4.4.2 Subcontroller B: Living room and kitchen room lights 17

Figure 4.4.2.0: Structure of each room light subcontroler for kitchen and living room 17

4.4.3 Subcontroller C: Light intensity regulator for living room and kitchen 18

Figure 4.4.3.0: Structure of each room light intensity regulator for kitchen and living room 18

Figure 4.4.3.1: Block extension 1 of the light intensity regulator 19

Figure 4.4.3.2: Block extension 2 of the light intensity regulator 20
Figure 4.4.3.3: Block extension 3 of the light intensity regulator 21
Figure 4.4.3.4: Block extension 4 of the light intensity regulator 22
4.4.4 Subcontroller D: Debouncer 23
Figure 4.4.4.0: Debouncer architecture 23
Figure 4.4.4.1: Debouncer controller finite-state machine 24
4.5 Heating controller 25
Figure 4.5.0.0: Inputs and outputs of the heating controller 25
Figure 4.5.0.1: Subcontrollers of the heating controller 26
Figure 4.5.0.2: Structure of each subcontroller inside the heating controller 27
4.6 Refrigerator controller 28
Figure 4.6.0.0: Inputs and outputs of the refrigerator controller 28
Figure 4.6.0.1: Structure of the refrigerator controller 29
4.7 Devices controller 30
Figure 4.7.0.0: Inputs and outputs of the devices controller 30
4.8 Power controller 31
Figure 4.8.0.0: Inputs and outputs of the power controller 31
4.9 PC Backup 32
Figure 4.9.0.0: Inputs and outputs of PC Backup 32
Figure 4.9.0.1: Structure of the PC Backup 32
5. Simulator hardware
Table 5.0.0.0: Color codes used for controllers 33
5.1 RTC Generator 33
Figure 5.1.0.0: Inputs and outputs of real-time clock generator 33
Figure 5.1.0.1: Structure of the real-time clock generator 34
Figure 5.1.0.2: Finite-state machine of the RTC generator 34
6. System software 35
6.1 Classes diagram 35
Figure 6.1.0.0: Software classes diagram for the decision-making software 35
6.2 Classes table 36
6.2.1 Decision making software 36
Table 6.2.1.0: Classes table for the system decision-making software 36
6.2.2 GUI 42

Table 6.2.2.0: Classes table for the system GUI 42
6.3 Interface 44
Figure 6.3.0.0: Interface preview for the decision making system 44
6.4 Program flow 45
Figure 6.4.0.0: Decision-making software program flow 45
7. Simulator software 46
7.1 Classes diagram 46
Figure 7.1.0.0: Classes diagram for the environment simulation software 46
7.2 Classes table 47
7.2.1 Environment simulation software 47
Table 7.2.1.0: Classes table for the environment simulation software 47
7.2.2 GUI 50
Table 7.2.2.0: Classes table for the simulator GUI 50
7.3 Interface 52
Figure 7.3.0.0: Interface preview for the simulator 52
7.4 Program flow 53
Figure 7.4.0.0: Environment simulation software program flow 53
8. Communication 54
8.1 Communication protocol 54
8.1.1 Write new values to the board 54
Figure 8.1.1.0: Communication sequence when writing new values to the board 54
8.1.2 Read values from the board 55
Figure 8.1.2.0: Communication sequence when reading values from the board 55
8.1.3 Error during communication 55
Figure 8.1.3.0: Communication sequence when there is an error during communication 55
8.2 Message encoding for the system 56
8.2.1 Write new values to the board 56
Figure 8.2.1.0: String for writing new values to the board 56
Figure 8.2.1.1: String for a write acknowledge 57
8.2.2 Read values from the board 58
Figure 8.2.2.0: String for reading values from the board 58
Figure 8.2.2.1: String for returning values to the board 59
8.2.3 Error during communication 60

Figure 8.2.3.0: String for flagging a communication error 60
8.3 Message encoding for the simulator 61
8.3.1 Write new values to the board 61
Figure 8.3.1.0: String for writing new values to the board 61
Figure 8.3.1.1: String for a write acknowledge 62
8.3.2 Read values from the board 63
Figure 8.3.2.0: String for reading values from the board 63
Figure 8.3.2.1: String for returning values to the board 64
8.3.3 Error during communication 65
Figure 8.3.3.0: String for flagging a communication error 65
8.4 Implementation on the board 66
Figure 8.4.0.0: System UART controller inputs and outputs 67
Figure 8.4.0.1: Simulator UART controller inputs and outputs 68
Figure 8.4.0.2: Structure of the UART controller 69
8.4.1 Baud rate counter 70
Figure 8.4.1.0: Baud rate controller architecture 70
8.4.2 Checksum generator 71
Figure 8.4.2.0: Checksum generator architecture 71
8.4.3 Serial receiver 72
Figure 8.4.3.0: Inputs and outputs of the serial receiver 72
Figure 8.4.3.1: Serial sender architecture 73
8.4.4 Serial sender 74
Figure 8.4.4.0: Inputs and outputs of the serial sender 74
Figure 8.4.4.1: Serial sender architecture 75
8.4.5 Buffer 76
Figure 8.4.5.0: Buffer architecture 76
8.4.6 PC Failure detector 77
Figure 8.4.6.0: PC Failure detector architecture 77

1. Introduction

This document is a complete design of a green house manager system. This system is made of a PC and a board. The design of a simulator is also include. This simulator is used to simulate the house. All the logic in the hardware is implemented on the boards with VHDL. The software is written in Java and runs under windows. The communication between the PC and the boards is done with a USB-to-COM cable.

The deployment diagram is first introduced. After that, a traceability matrix is produced in order to relate the previous SRS with this SDD. The hardware design is then introduced for both the system and the simulator. Once this is done, the software design is described for the system and the simulator. Finally, the communication protocol and controllers used to communicate between the PC and the boards are described.

2. Deployment

The overall system is shown in figure 2.0.0.0. The simulator is made of one computer and one board. The system is also made of one computer and one board. The inputs and outputs of both boards are connected with ribbon cables. This setup is the one used for simulating a house. In an actual implementation, the simulator would be replaced by the house components. Each part is described in the details in the following sections.

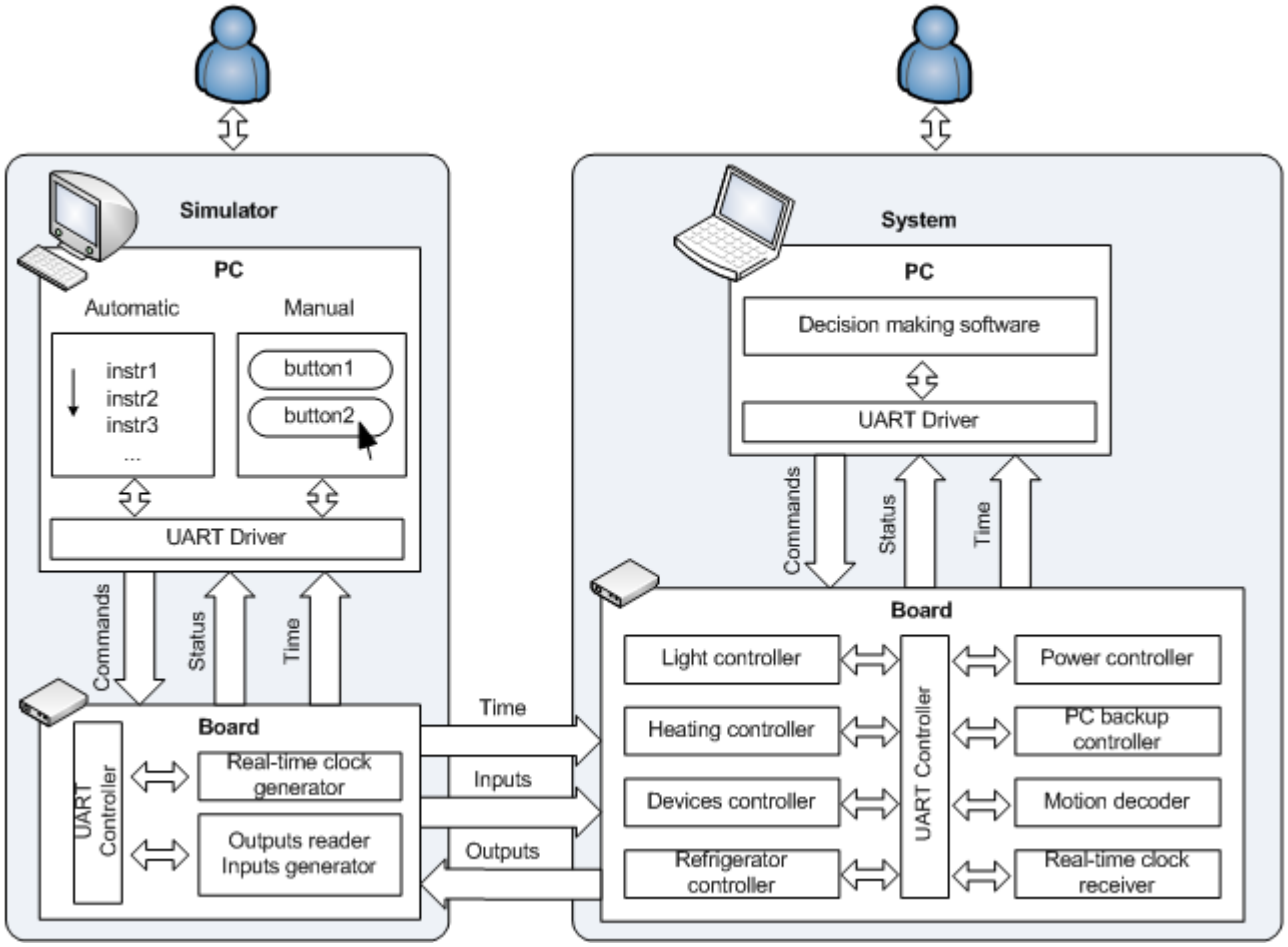


Figure 2.0.0.0: Deployment diagram

3. Traceability matrix

The general overview of the traceability matrix is shown in table 3.0.0.0. The complete traceability matrix is shown in table 3.0.0.1.

	Functional requirements										**Error cases**	
	1	2	3	4	5	6	7	8	9	10	1	2
System hardware	X	X	X	X	X	X	X	X	X	X	X	X
1 RTC timer	X	X	X	X	X	X	X	X	X	X	X	X
2 RTC divider		X	X	X	X	X	X	X	X	X	X	X
3 Motion decoder				X								
4 Light controller	X	X	X	X							X	X
5 Heating controller						X	X	X	X	X	X	X
6 Refrigerator controller					X	X	X				X	X
7 Devices controller				X							X	X
8 Power controller						X	X				X	X
9 PC Backup												X
System software				X	X	X	X	X	X	X	X	
1 DecisionControl class				X	X	X	X	X	X	X	X	
2 Scheduler class									X	X		
3 Event class									X	X		
4 PowerMonitor class						X	X				X	
5 Bathroom class				X		X	X		X	X	X	
6 Bedroom1 class				X		X	X		X	X	X	
7 Bedroom2 class				X		X	X		X	X	X	
8 Kitchen class				X	X	X	X		X	X	X	
9 LivingRoom class				X		X	X		X	X	X	
10 Power class						X	X		X	X	X	
11 Clock class									X	X		
12 UARTDriver class				X	X	X	X	X	X	X	X	

Table 3.0.0.0: Overview of the traceability matrix for the system

Functional requirements	SRS Section	Design	SDD section
Room Lights (Bedroom 1 et 2, Living-room, Kitchen, Bathroom)	Section 4.1	RTC timer	Section 4.1
		Light controller	Section 4.4
Room Lights (Living-room and Kitchen)	Section 4.2	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Light controller	Section 4.4
Bed Lamps	Section 4.3	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Light controller	Section 4.4
Motion detection	Section 4.4	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Motion decoder	Section 4.3
		Light controller	Section 4.4
		Devices controller	Section 4.7
		DecisionControl class	Section 6.1
		Bathroom class	Section 6.1
		Bedroom1 class	Section 6.1
		Bedroom2 class	Section 6.1
		Kitchen class	Section 6.1
		Living room class	Section 6.1
		UARTDriver class	Section 6.1
Refrigerator	Section 4.5	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Refrigerator controller	Section 4.6
		DecisionControl class	Section 6.1
		Kitchen class	Section 6.1
		UARTDriver class	Section 6.1
Power consumption - excess	Section 4.6	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Heating controller	Section 4.5
		Refrigerator controller	Section 4.6
		Power controller	Section 4.8
		DecisionControl class	Section 6.1
		PowerMonitor class	Section 6.1
		Bathroom class	Section 6.1
		Bedroom1 class	Section 6.1

		Bedroom2 class	Section 6.1
		Kitchen class	Section 6.1
		LivingRoom class	Section 6.1
		Power class	Section 6.1
		UARTDriver class	Section 6.1
Power consumption - shortage	Section 4.7	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Heating controller	Section 4.5
		Refrigerator controller	Section 4.6
		Power controller	Section 4.8
		DecisionControl class	Section 6.1
		PowerMonitor class	Section 6.1
		Bathroom class	Section 6.1
		Bedroom1 class	Section 6.1
		Bedroom2 class	Section 6.1
		Kitchen class	Section 6.1
		LivingRoom class	Section 6.1
		Power class	Section 6.1
		UARTDriver class	Section 6.1
Temperature	Section 4.8	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Heating controller	Section 4.5
		DecisionControl class	Section 6.1
		UART Driver class	Section 6.1
Scheduler	Section 4.9	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Heating controller	Section 4.5
		DecisionControl class	Section 6.1
		Scheduler class	Section 6.1
		Even class	Section 6.1
		Bathroom class	Section 6.1
		Bedroom1 class	Section 6.1
		Bedroom2 class	Section 6.1
		Kitchen class	Section 6.1
		LivingRoom class	Section 6.1
		Power class	Section 6.1

		Clock class	Section 6.1
		UARTDriver class	Section 6.1
Night power saving mode	Section 4.10	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Heating controller	Section 4.5
		DecisionControl class	Section 6.1
		Scheduler class	Section 6.1
		Even class	Section 6.1
		Bathroom class	Section 6.1
		Bedroom1 class	Section 6.1
		Bedroom2 class	Section 6.1
		Kitchen class	Section 6.1
		LivingRoom class	Section 6.1
		Power class	Section 6.1
		Clock class	Section 6.1
		UARTDriver class	Section 6.1
Electricity failure	Section 5.1	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Light controller	Section 4.4
		Heating controller	Section 4.5
		Refrigerator controller	Section 4.6
		Devices controller	Section 4.7
		Power controller	Section 4.8
		DecisionControl class	Section 6.1
		PowerMonitor class	Section 6.1
		Bathroom class	Section 6.1
		Bedroom1 class	Section 6.1
		Bedroom2 class	Section 6.1
		Kitchen class	Section 6.1
		LivingRoom class	Section 6.1
		Power class	Section 6.1
		UARTDriver class	Section 6.1
PC failure	Section 5.2	RTC timer	Section 4.1
		RTC divider	Section 4.2
		Light controller	Section 4.4
		Heating controller	Section 4.5

		Refrigerator controller	Section 4.6
		Devices controller	Section 4.7
		Power controller	Section 4.8
		PC Backup	Section 4.9

Table 3.0.0.1: Traceability matrix for the system

4. System hardware

A color coding scheme is used for the system hardware as shown in table 4.0.0.0.

Color	Description
Red	External input signal from the house or simulator
Blue	External output signal to the house or simulator
Orange	Signal provided to the UART controller and that will be sent to the PC
Green	Signal provided from the UART controller and previously received from the PC

Table 4.0.0.0: Color codes used for controllers

4.1. Real-time clock timer

There are 2 different kinds of clocks in this system: the real-time clock (RTC) and the system clock (clock)

Real-time clock

The real time clock is provided as an input and is produced by the simulator board. This clock is a low frequency clock and is used for synchronization in real-time. It is used to calculate long period of times in different controllers. Each clock pulse represents 1/10 of a second in real time. The clock can have a period shorter than 1/10 of a second in order to speed up simulation.

System clock

The system clock is provided on the board and is a high frequency clock (25.175 MHz). This clock is used for logic operation on the board. It is completely independent of the real-time clock.

Inputs and outputs

The Time module is responsible for keeping track of time on the system. It has the same internal counter register set up as the RTC Generator (will be discussed later). The Generator and Receiver should always be synchronized to the same time. The inputs and outputs of this timer are shown in figure 4.1.0.0.

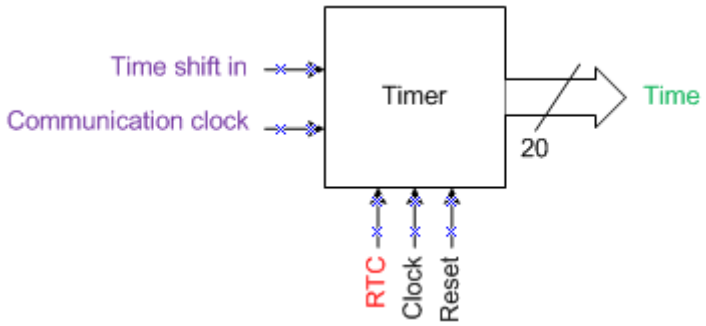


Figure 4.1.0.0: Inputs and outputs of the real-time clock timer

Internal design

The timer is used in order to keep the absolute value of time in memory. There are two different ways the time can be updated:

Normal mode

The timer is incremented on each pulse of the real-time clock. This is shown in figure 4.1.0.1.

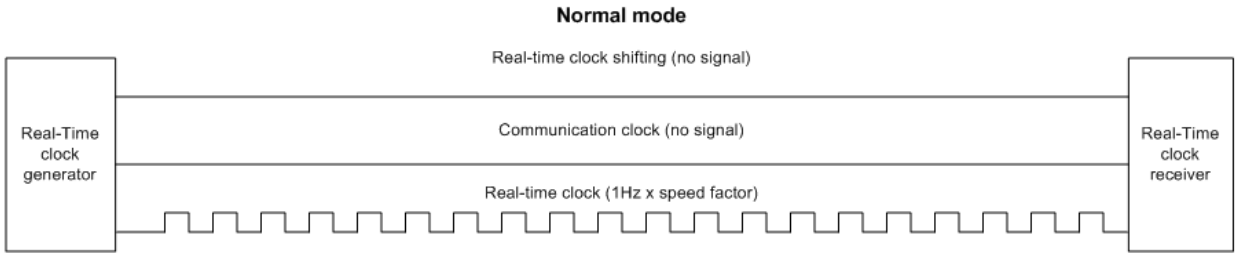


Figure 4.1.0.1: Normal mode timing

Update mode

When there is a need to jump to a specific time, the update mode is used. However, when jumping to a given time, all operations that would occur between the initial and final times are not simulated. This is only a jump in time, not a simulation at higher speed from the initial time to the final time. This mode is only a tool for simulation purposes. This is shown in figure 4.1.0.2.

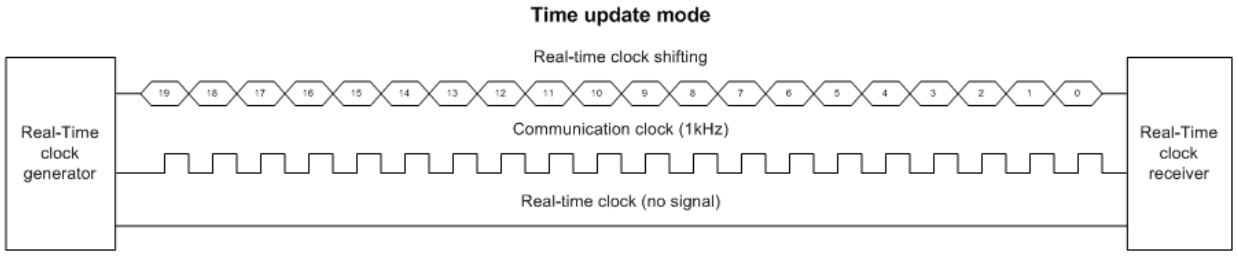


Figure 4.1.0.2: Update mode timing

The timer is implemented with a finite state machine. Figure 4.1.0.3 shows the finite state machine of the timer.

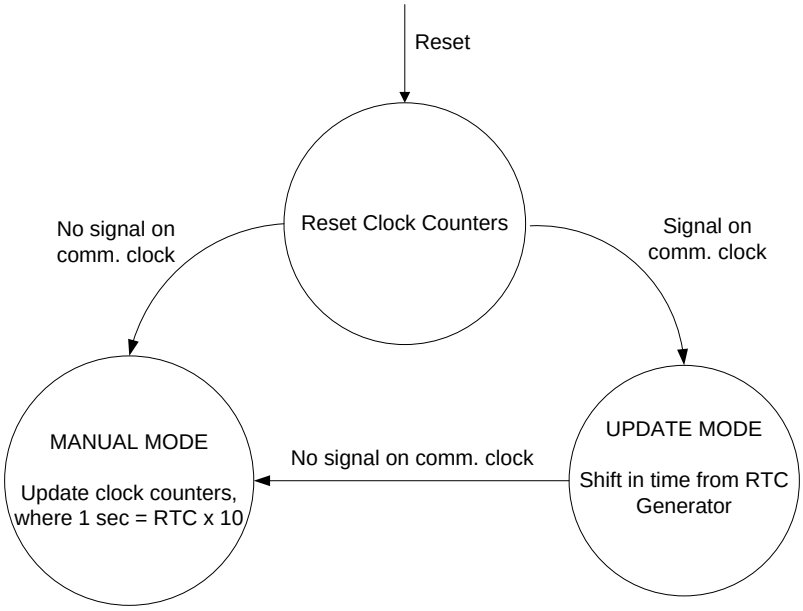


Figure 4.1.0.3: Finite-state machine for the real-time clock timer

4.2. Real-time clock divider

Inputs and Outputs

The real-time clock divider is used in order to provide a signal for controllers that need updates on a minute scale. This avoids using large counters for each controller. The real-time clock divider provides a factor of 100 and thus the signal is named RTC100. Its inputs and output are shown in figure 4.2.0.0.

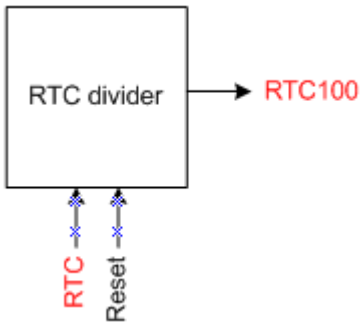


Figure 4.2.0.0: Inputs and outputs of the real-time clock divider

Internal design

This divider is implemented with a counter and a T flip-flop as shown in figure 4.2.0.1.

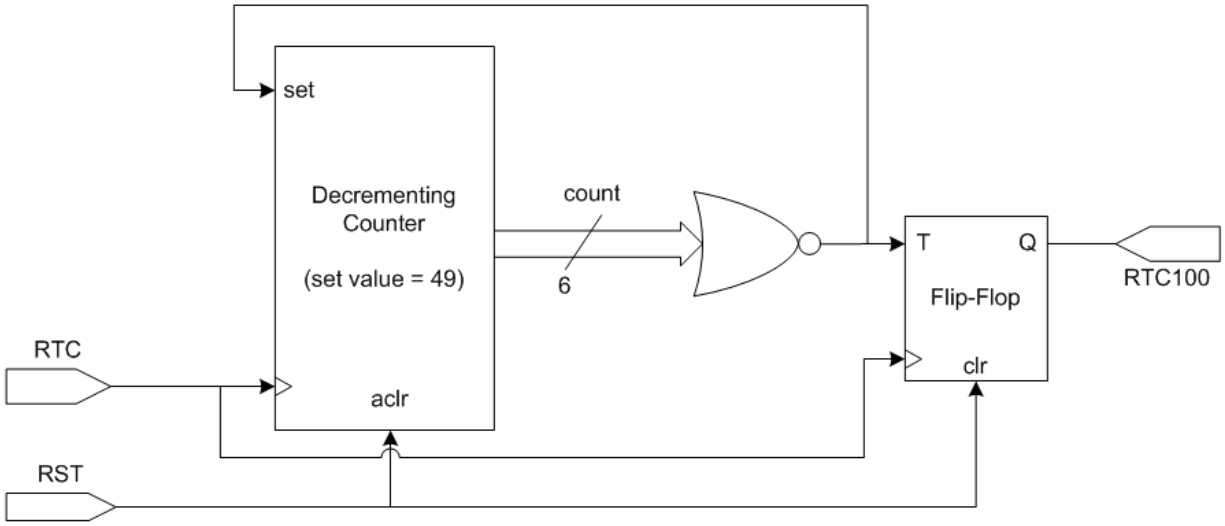


Figure 4.2.0.1: Logic circuit of the real-time clock divider

4.3. Motion decoder

Inputs and outputs

The motion decoder receives external inputs and produces outputs to the PC as shown in figure 4.3.0.0:

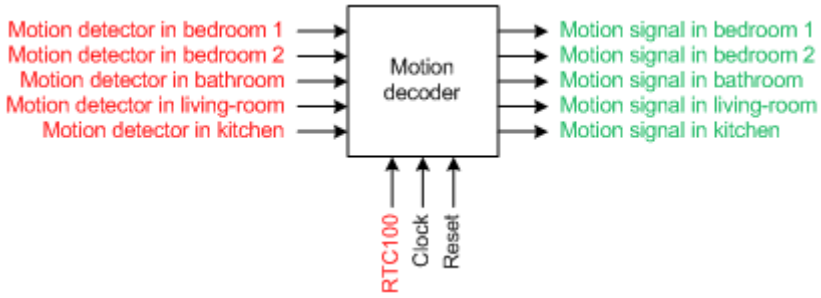


Figure 4.3.0.0: Inputs and outputs of the motion decoder

Internal design

The motion decoder is made of 5 subcontrollers of the same type as shown in figure 4.3.0.1:

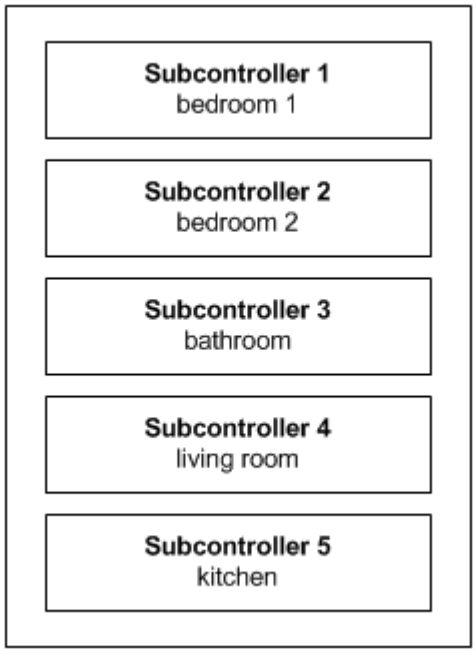


Figure 4.3.0.1: Subcontrollers of the motion decoder

Subcontrollers

Each subcontroller is made of a controller and a counter as shown in figure 4.3.0.2.

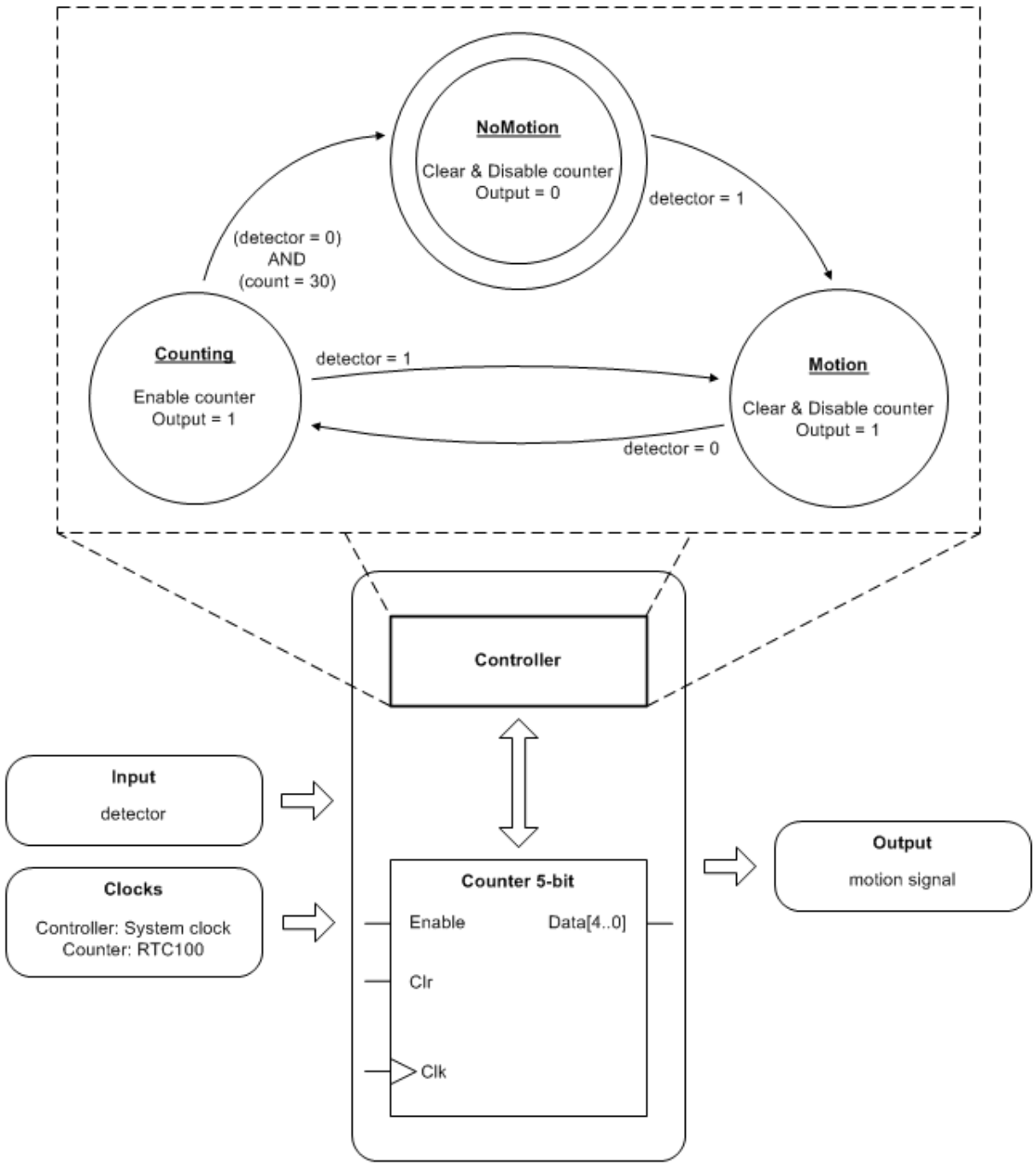


Figure 4.3.0.2: Structure of each subcontroller inside the motion decoder

4.4. Light controller

Inputs and outputs

The light controller receives external inputs and inputs from the PC. It produces outputs to the PC and external outputs. This is shown in figure 4.4.0.0.

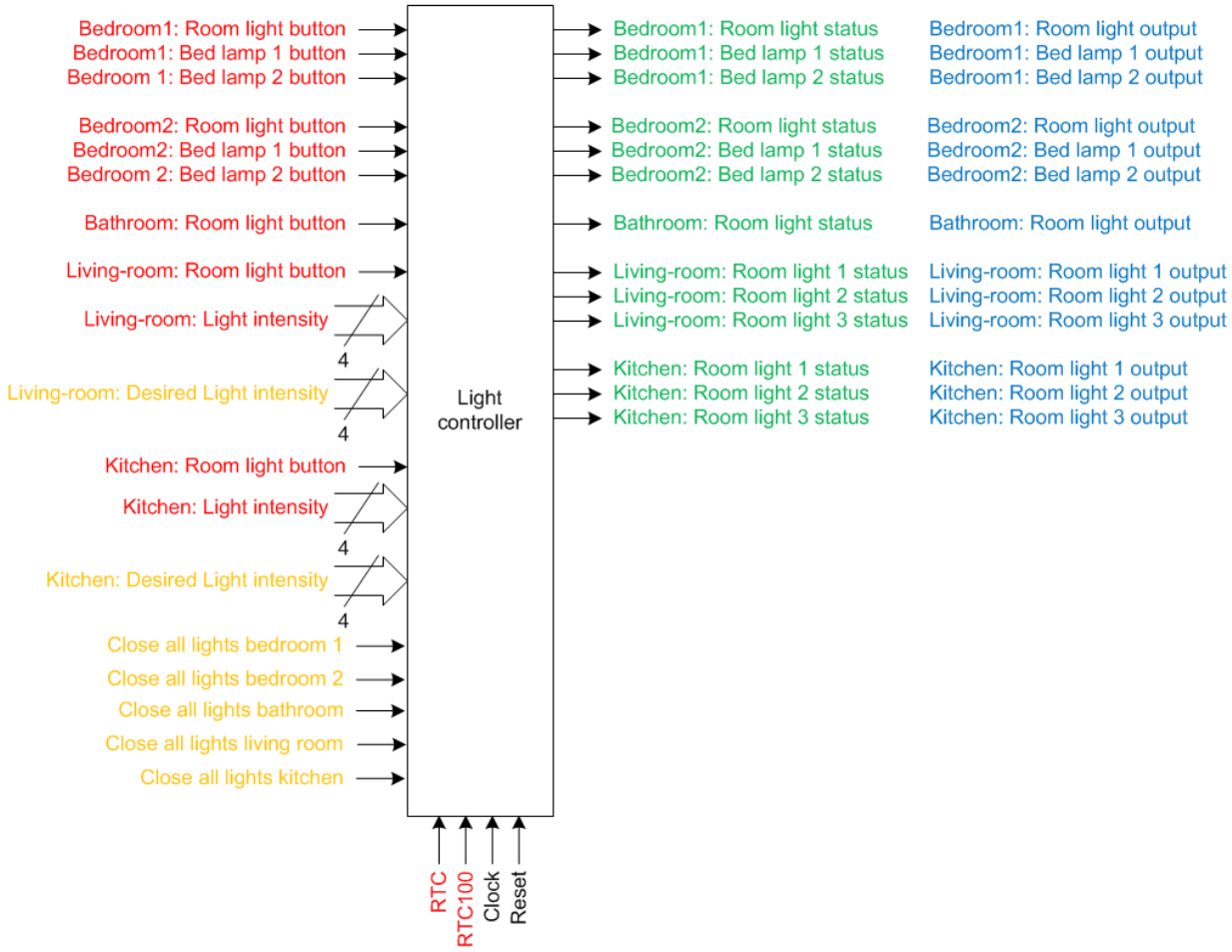


Figure 4.4.0.0: Inputs and outputs of the light controller

Internal design

The light controller is made of 12 sub-controllers of 4 different types as shown in figure 4.4.0.1.

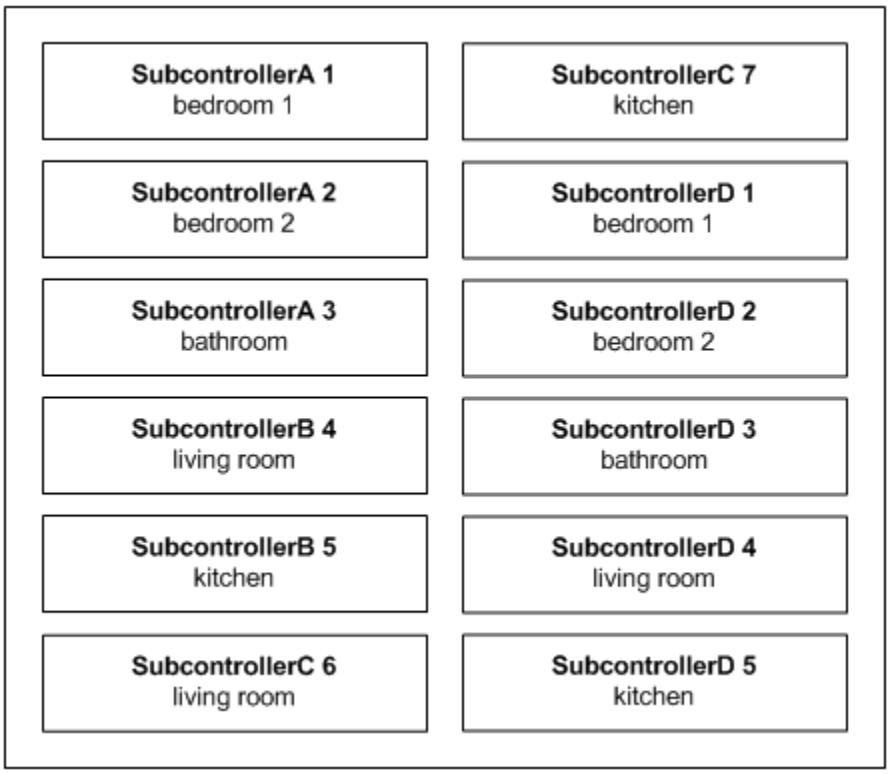


Figure 4.4.0.1: Subcontrollers of the light controller

Subcontrollers

4.4.1. Subcontroller A: Bedrooms and bathroom room lights

This subcontroller is used to control the room light in both bedrooms and in the bathroom. It is made of a controller as shown in figure 4.4.1.0.

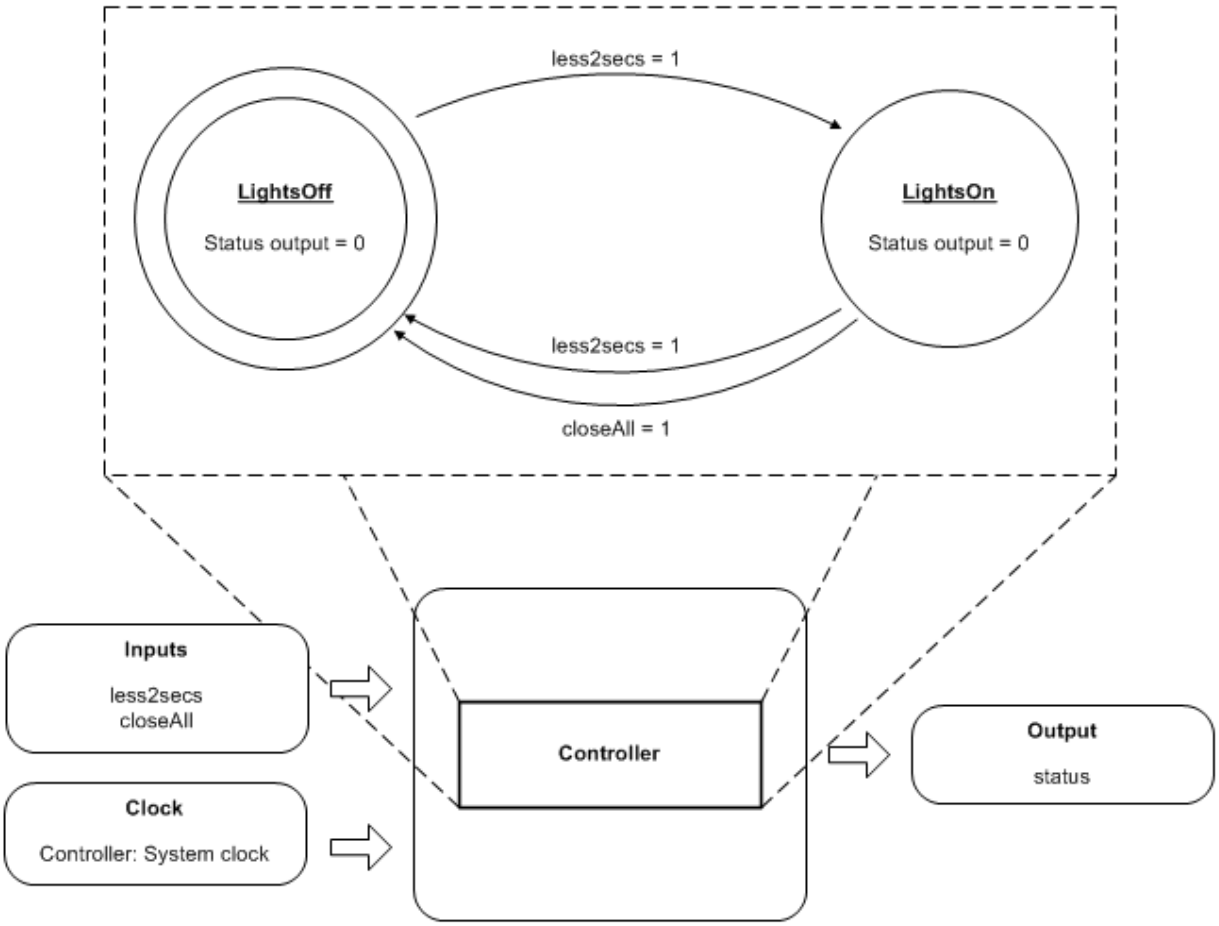


Figure 4.4.1.0: Structure of each room light subcontroller for bedrooms and bathroom

4.4.2. Subcontroller B: Living room and kitchen room lights

This subcontroller is used to control the room light in the kitchen and the living room. The RL1Status, RL2Status and RL3Status signals are multiplexed. When mux=0, these signals are controlled by subcontroller B. When mux=1, these signals are controlled by subcontroller C. Subcontroller B is made of a controller as shown in figure 4.4.2.0.

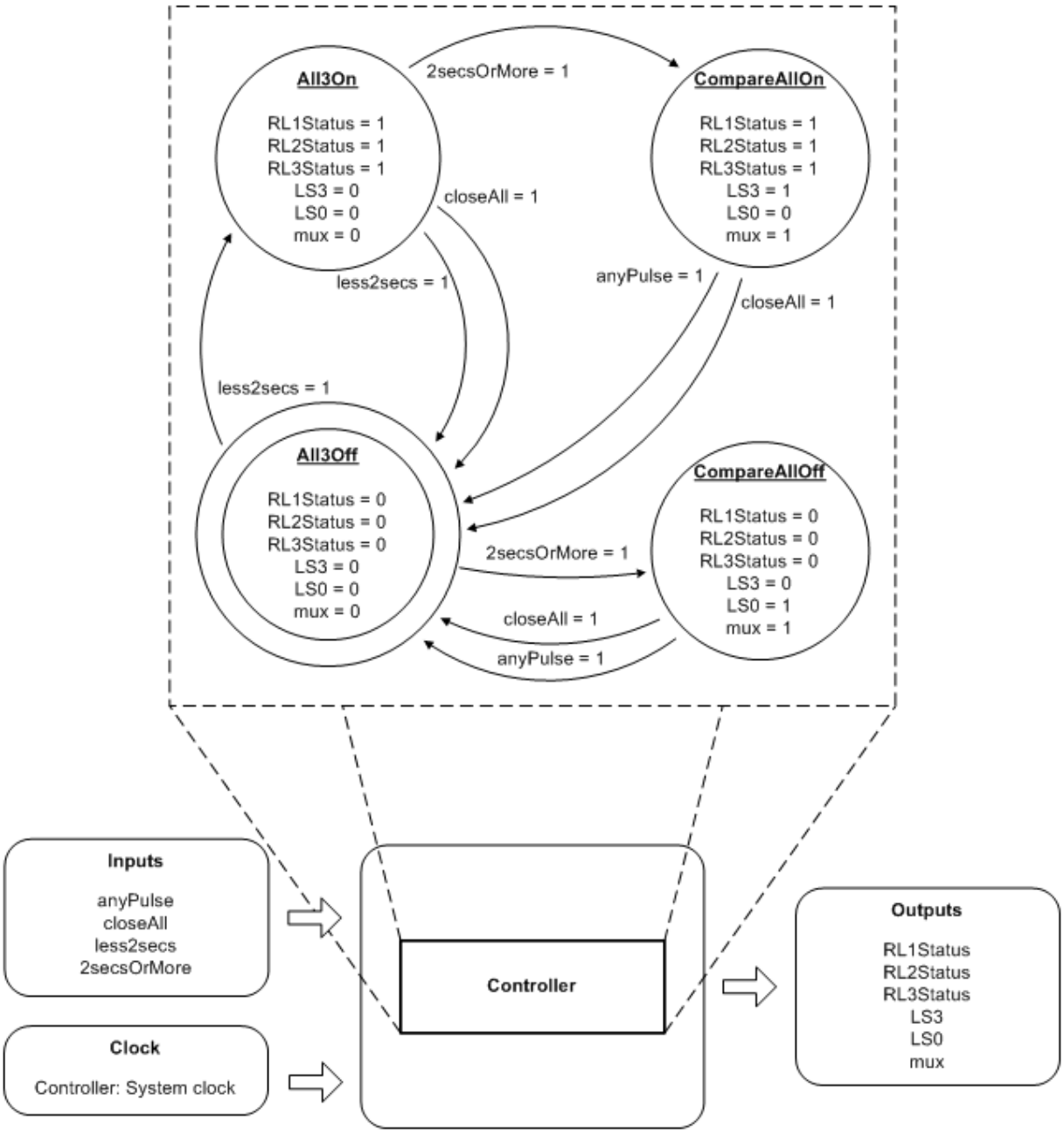


Figure 4.4.2.0: Structure of each room light subcontroller for kitchen and living room

4.4.3. Subcontroller C: Light intensity regulator for living room and kitchen

This subcontroller is used to control the light intensity in the kitchen and the living room. It is made of two counters and one controller as shown in figures 4.4.3.0, 4.4.3.1, 4.4.3.2, 4.4.3.3 and 4.4.3.4.

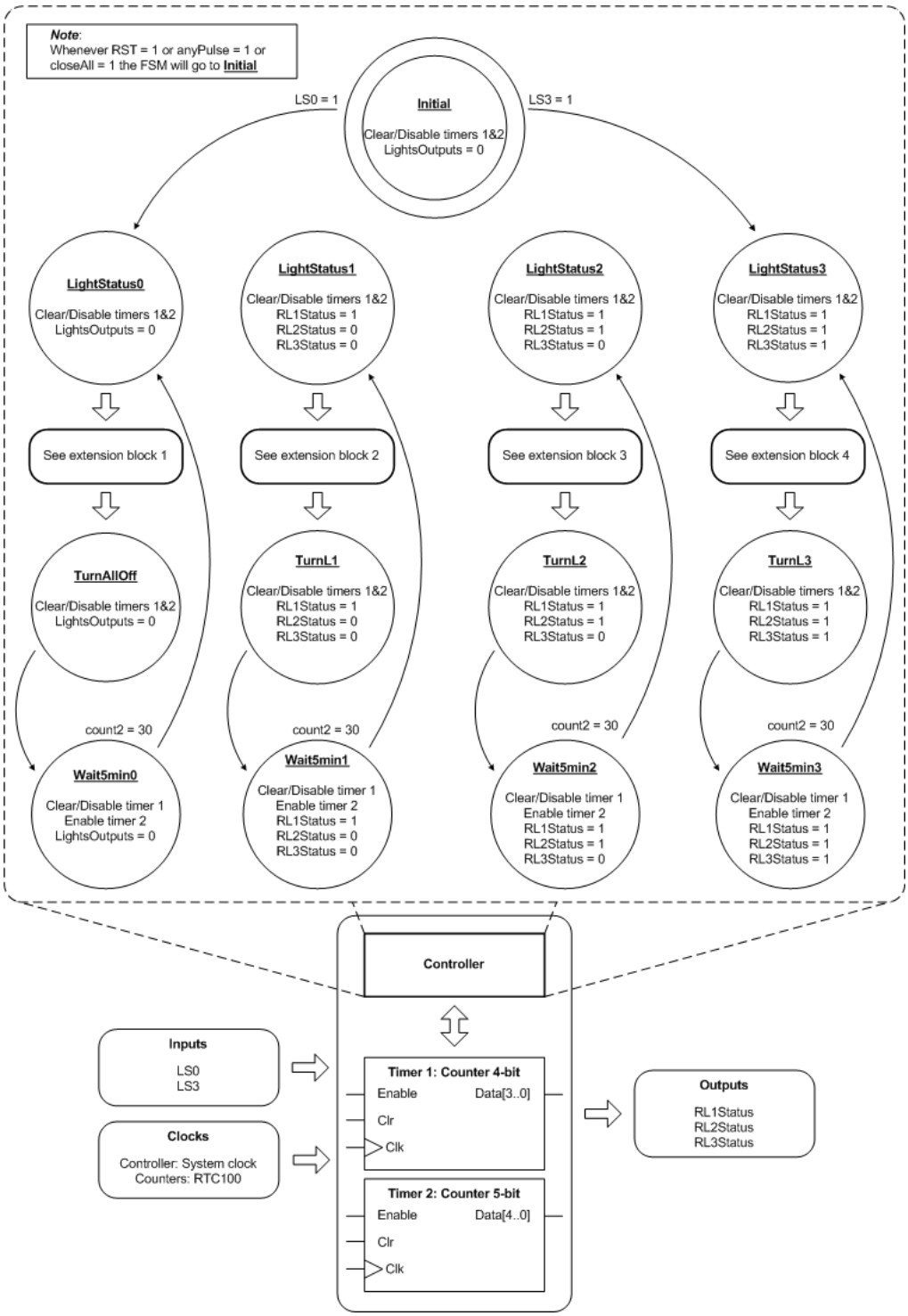


Figure 4.4.3.0: Structure of each room light intensity regulator for kitchen and living room

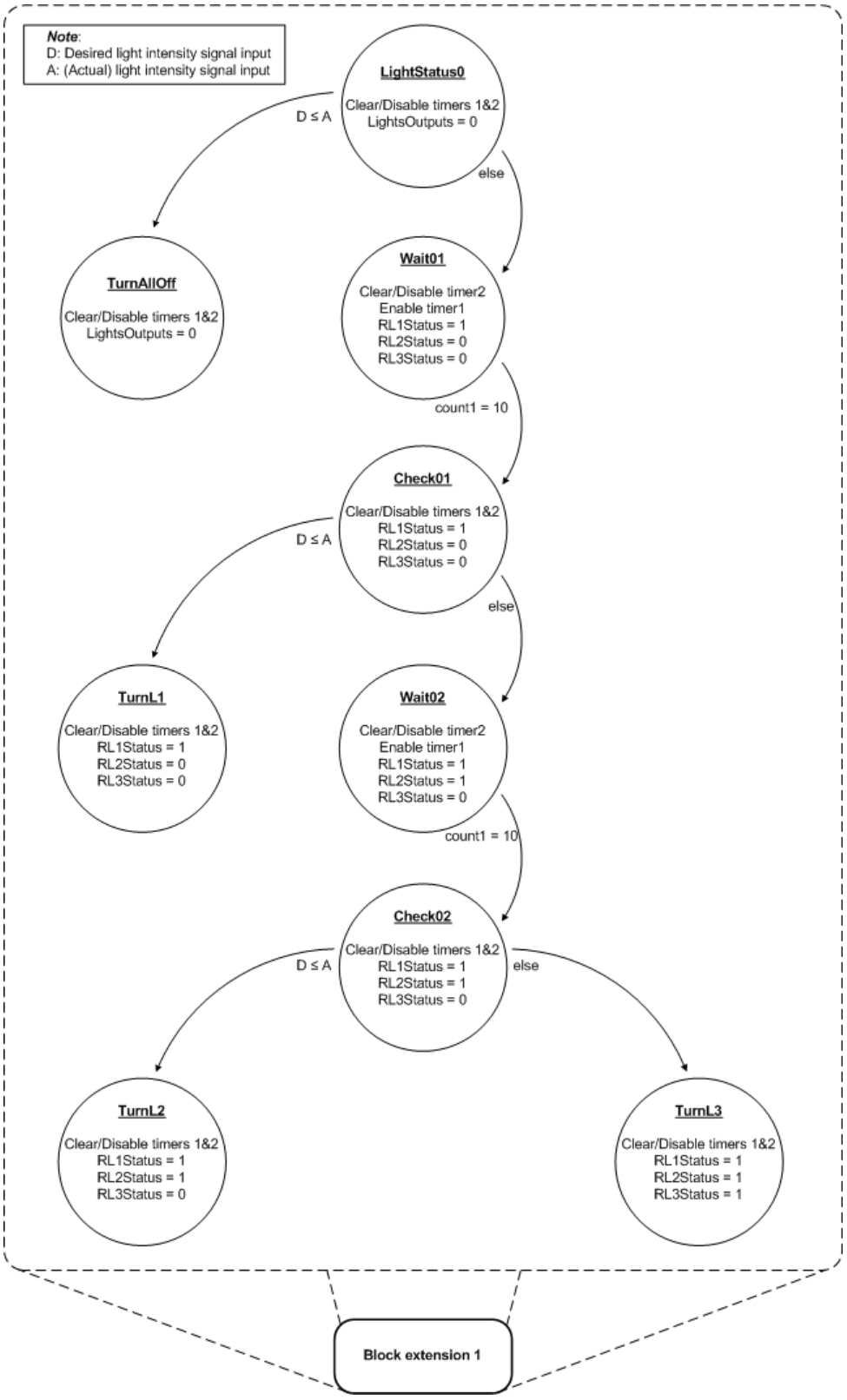


Figure 4.4.3.1: Block extension 1 of the light intensity regulator

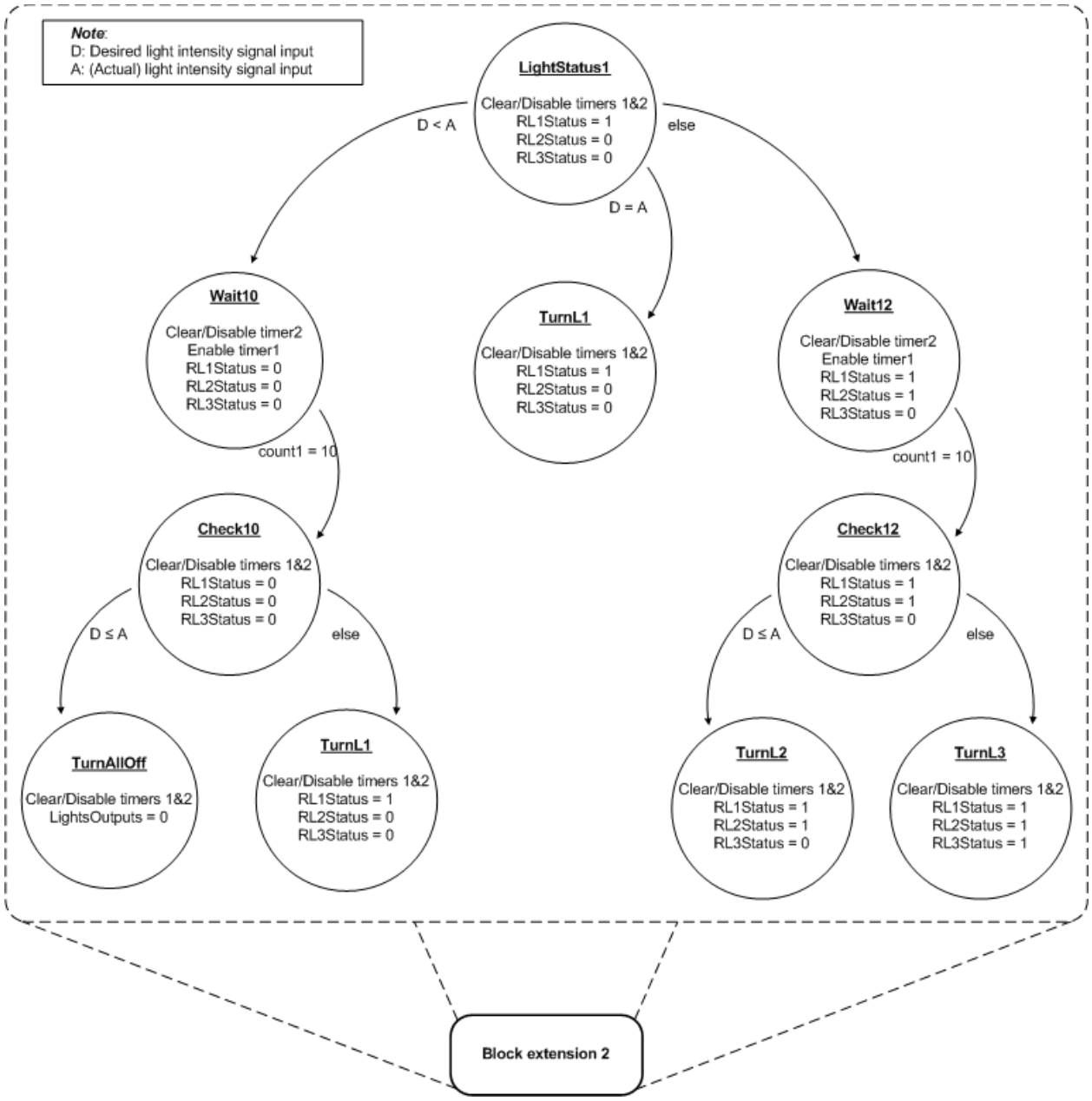


Figure 4.4.3.2: Block extension 2 of the light intensity regulator

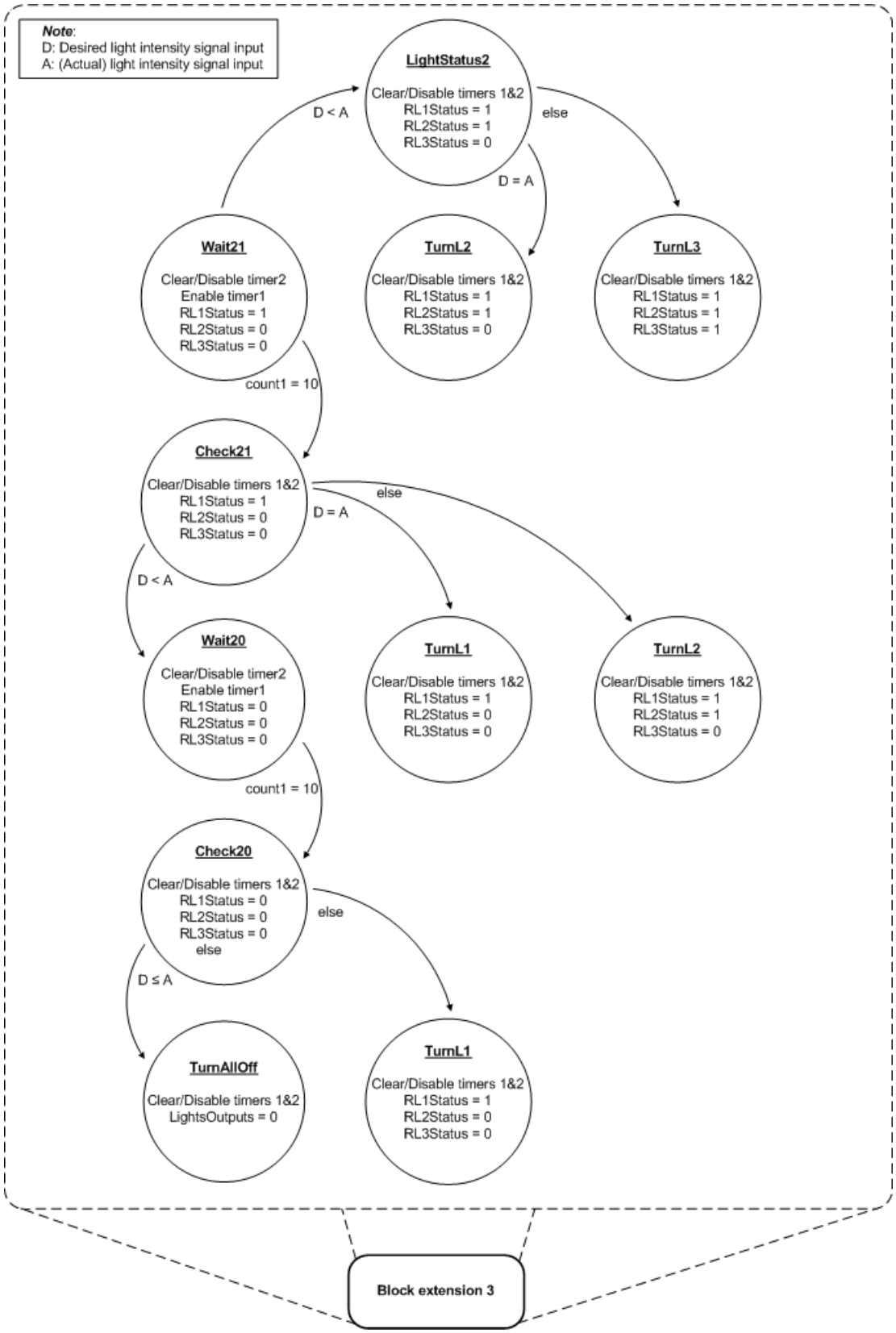


Figure 4.4.3.3: Block extension 3 of the light intensity regulator

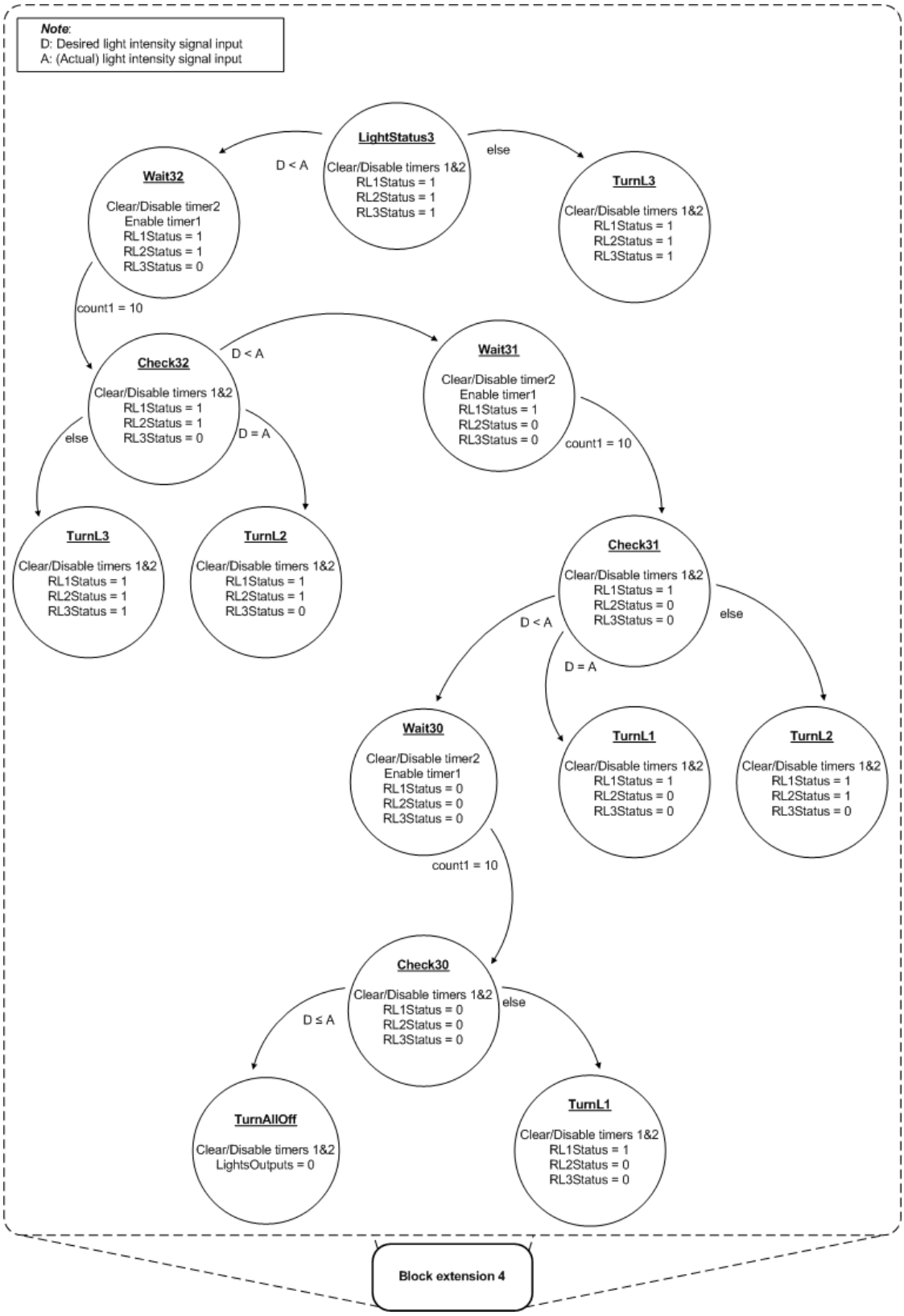


Figure 4.4.3.4: Block extension 4 of the light intensity regulator

4.4.4. Subcontroller D: Debouncer

This subcontroller is used to debounce each button and calculate the time when the button is kept pressed. It is made of one controller as shown in figures 4.4.4.0 and 4.4.4.1.

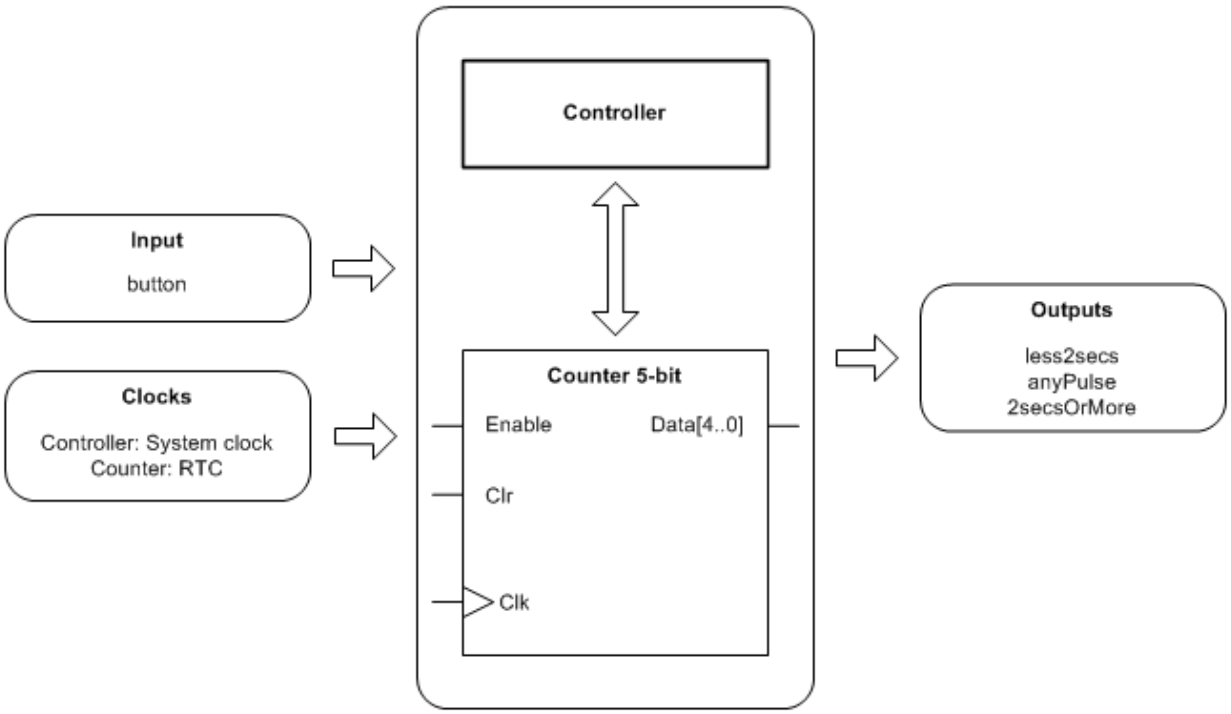


Figure 4.4.4.0: Debouncer architecture

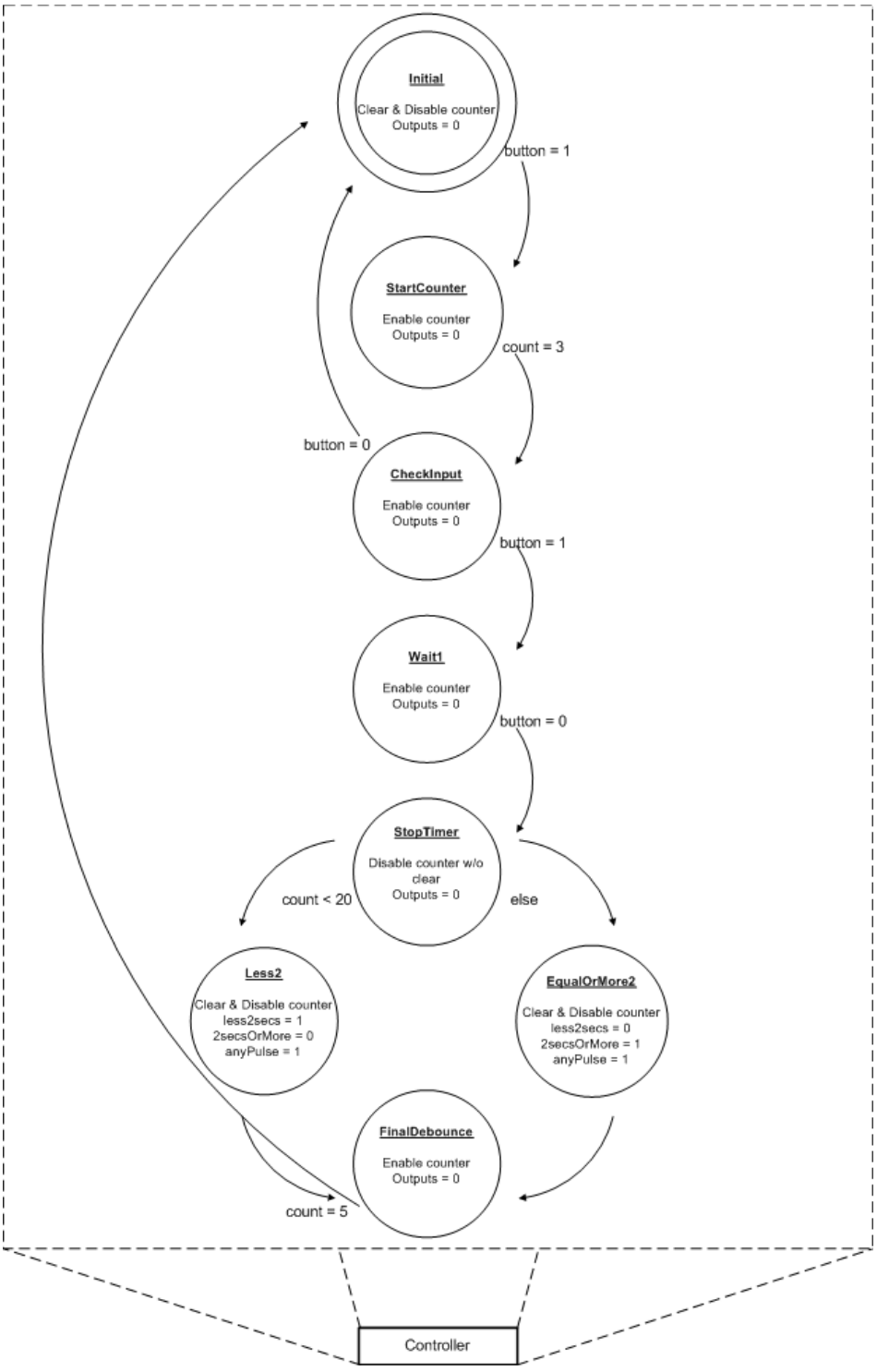


Figure 4.4.4.1: Debouncer controller finite-state machine

4.5. Heating controller

Inputs and outputs

The heating controller receives external inputs and inputs from the PC. It produces outputs to the PC and external outputs. This is shown in figure 4.5.0.0.

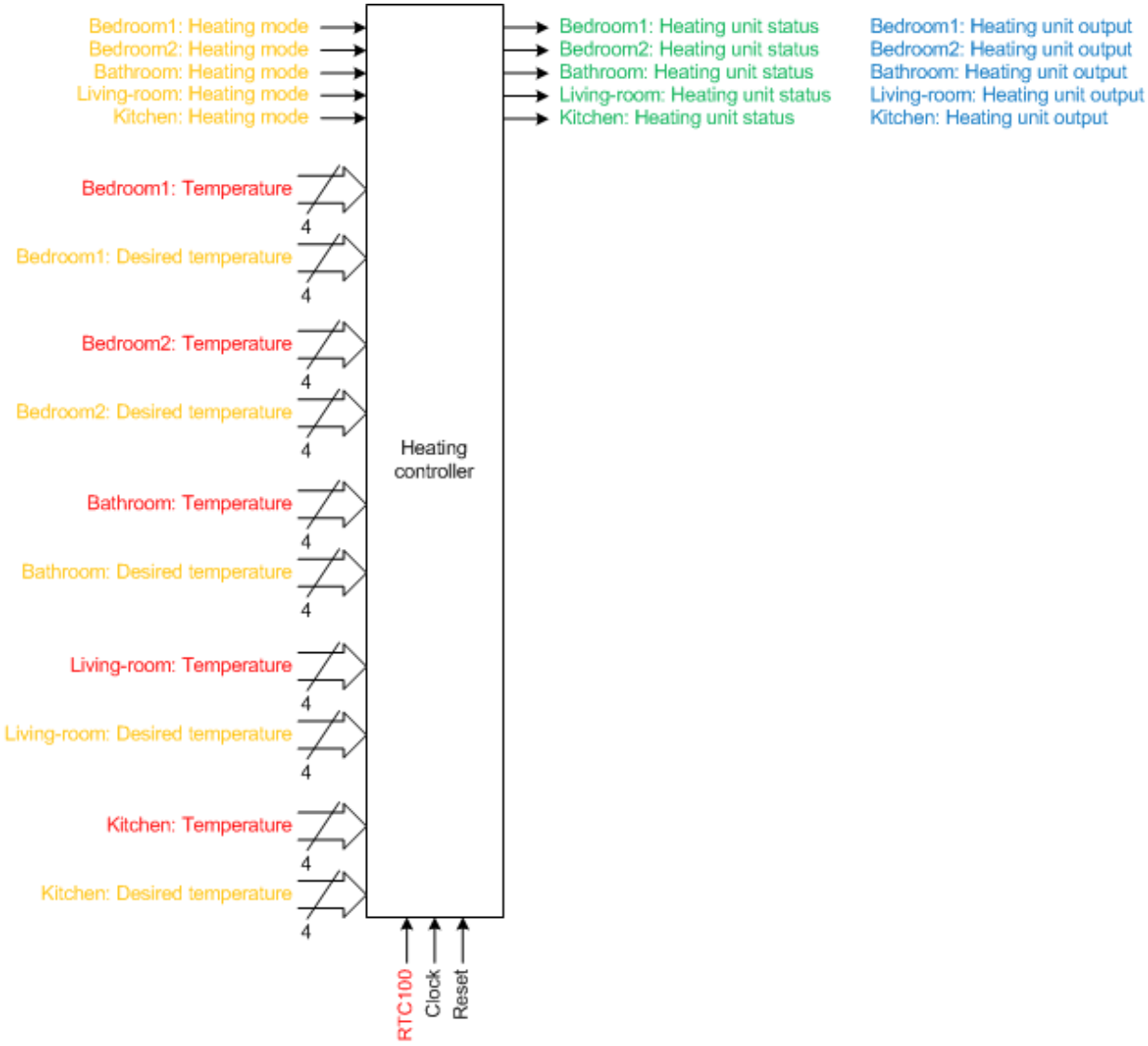


Figure 4.5.0.0: Inputs and outputs of the heating controller

Internal design

The lights controller is made of 5 subcontrollers of the same type as shown in figure 4.5.0.1.

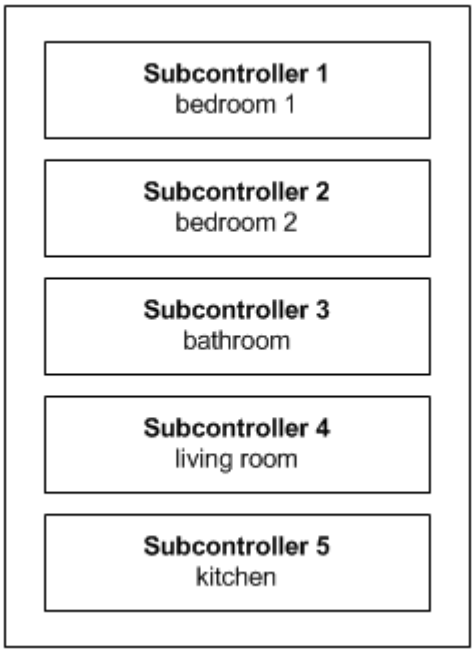


Figure 4.5.0.1: Subcontrollers of the heating controller

Subcontrollers

Each subcontroller is made of a counter and a controller as shown in figure 4.5.0.2.

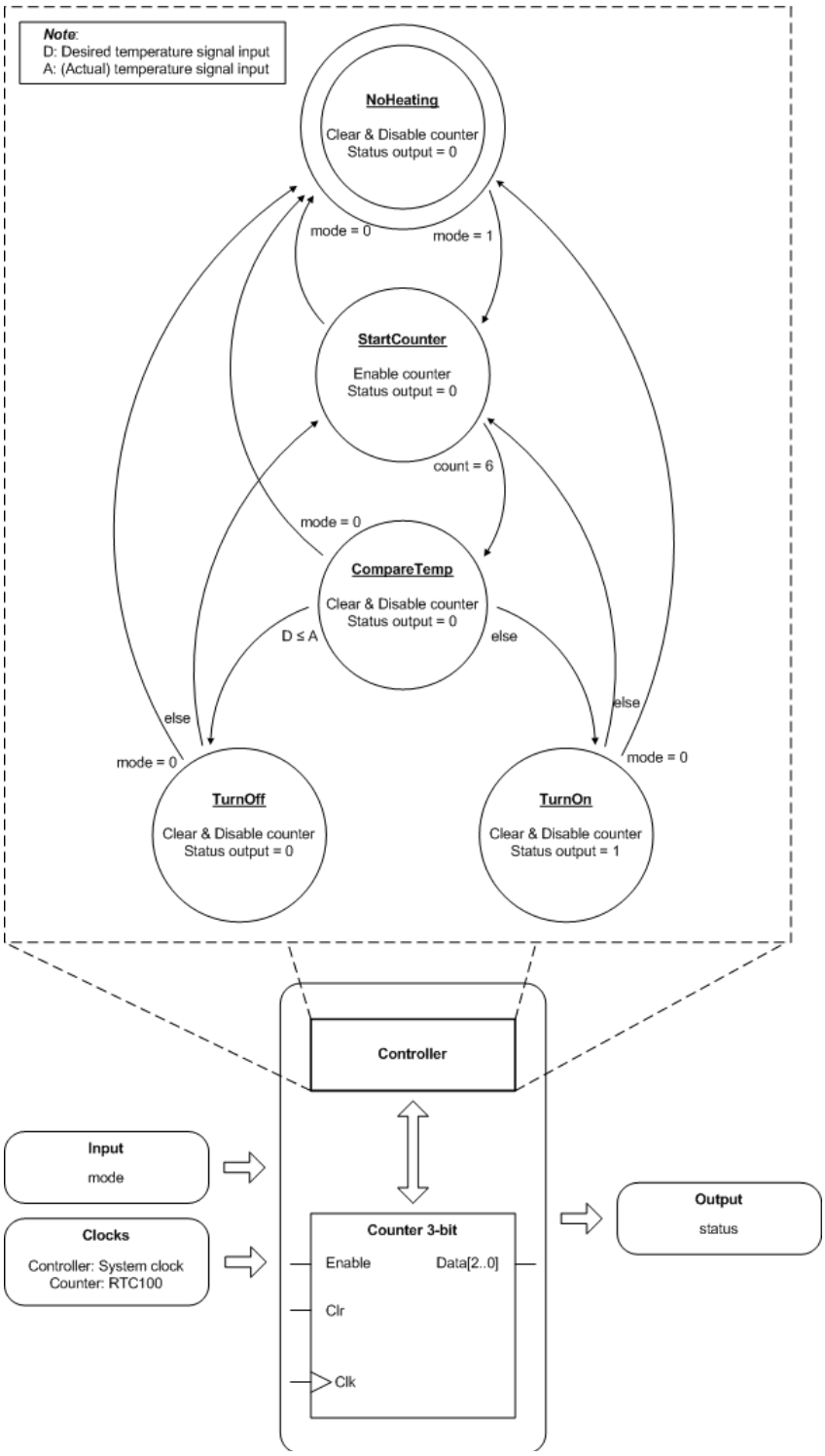


Fig 4.5.0.2: Structure of each subcontroller inside the heating controller

4.6. Refrigerator controller

Inputs and outputs

Initially, the refrigerator has to be turned ON for 30 minutes and waits for any special request (to be turned ON or OFF). If it gets a request to be turned ON, it automatically turns ON, but if it is asked to be turned OFF, it checks to see if the ratio time has been satisfied (If the refrigerator has stayed ON for at least 30 minutes out of one hour). According to the result, the refrigerator is either turned OFF as required or kept ON to satisfy the requirements. The inputs and outputs of the refrigerator controller are shown in figure 4.6.0.0.

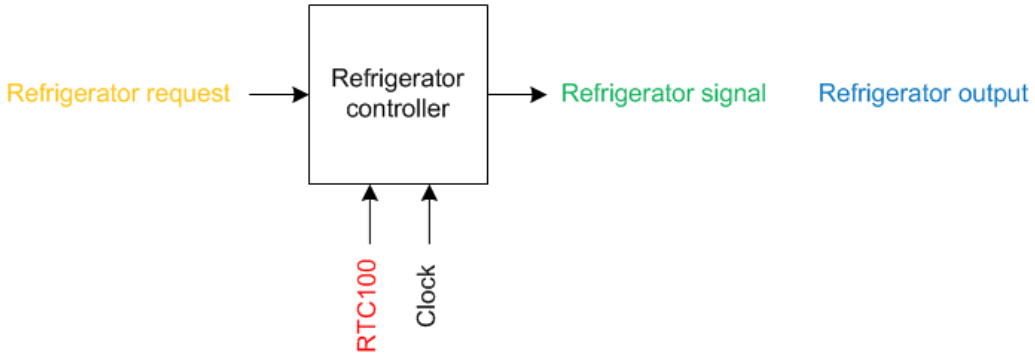


Figure 4.6.0.0: Inputs and outputs of the refrigerator controller

Internal design

The refrigerator controller is using two different counters and a controller. The design of this control is shown in figure 4.6.0.1.

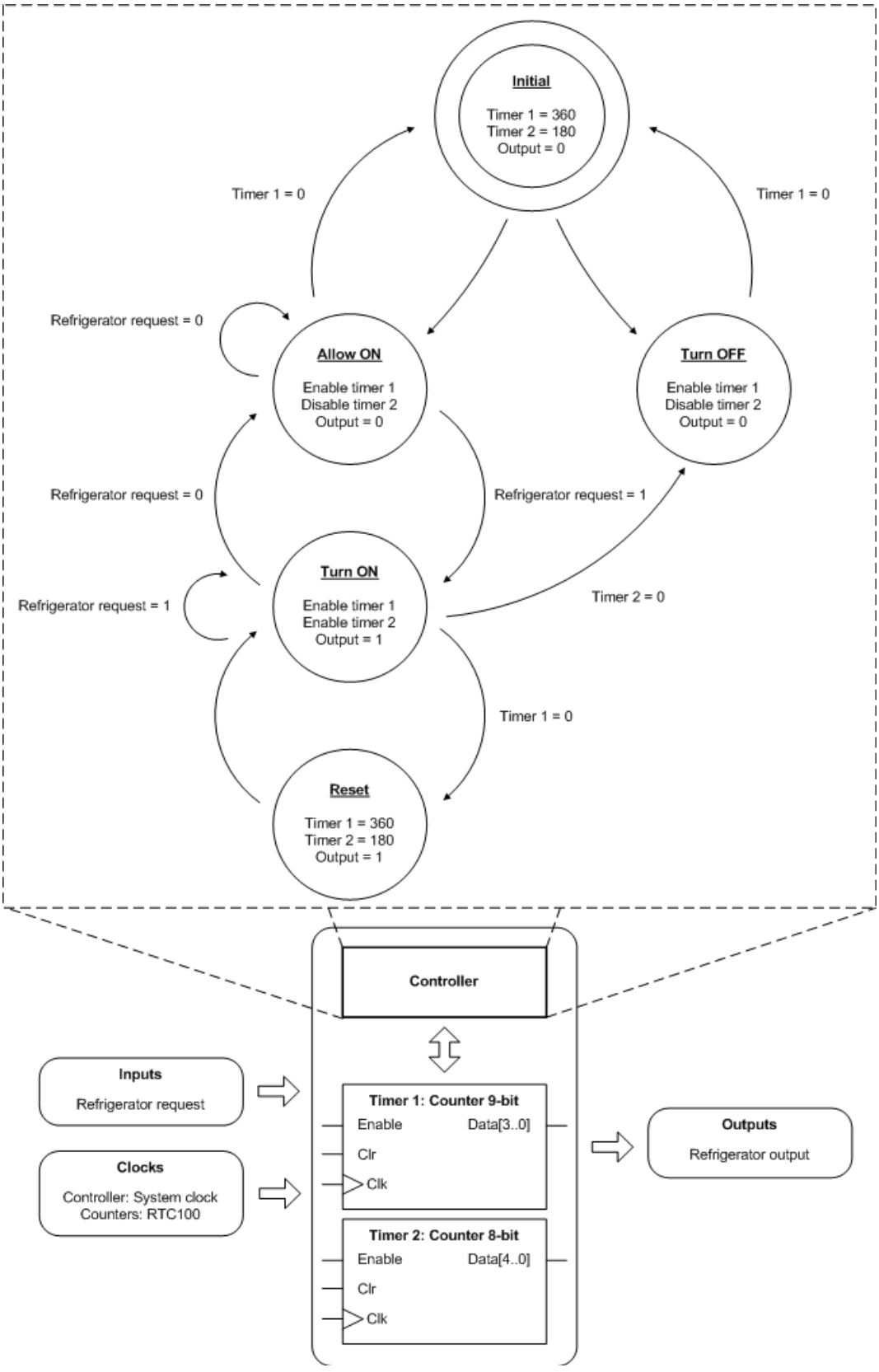


Figure 4.6.0.1: Structure of the refrigerator controller

4.7. Devices controller

Inputs and outputs

There is no logic in this controller. All inputs are connected directly to outputs. This is used only for signal organizational purposes. Inputs and outputs are shown in figure 4.7.0.0.

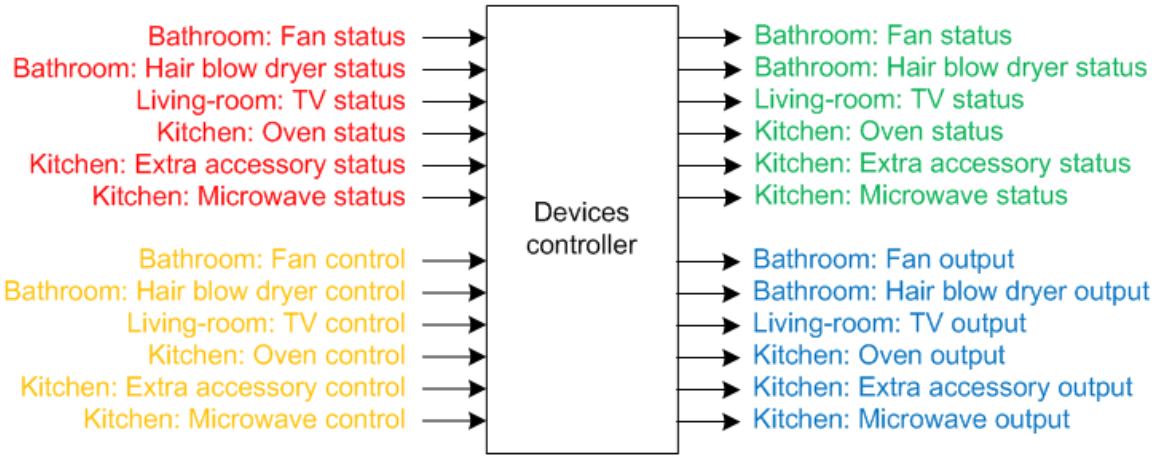


Figure 4.7.0.0: Inputs and outputs of the devices controller

4.8. Power controller

Inputs and Outputs

There is no logic in this controller. All inputs are connected directly to outputs. The “use battery” input is connected to the “discharge battery” output. This is used only for signal organizational purposes. Inputs and outputs are shown in figure 4.8.0.0.

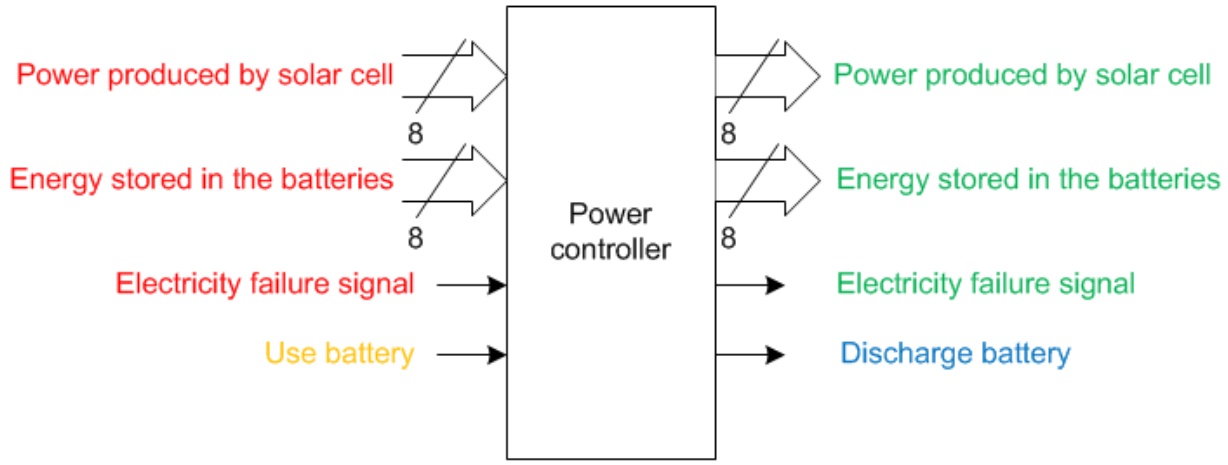


Figure 4.8.0.0: Inputs and outputs of the power controller

4.9. PC Backup

Inputs and outputs

The PC Backup controller produces signals to be read by controllers in the case where the PC is no longer responding. Its inputs and outputs are shown in figure 4.9.0.0.

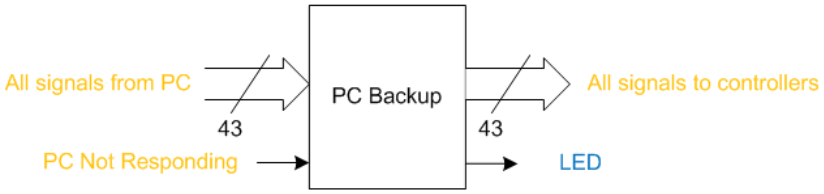


Figure 4.9.0.0: Inputs and outputs of PC Backup

Internal design

The controller connects signals from the PC to controllers if the PC is responding. If it is not responding, default values are sent to the controllers. It is made of one controller as shown in figure 4.9.0.1.

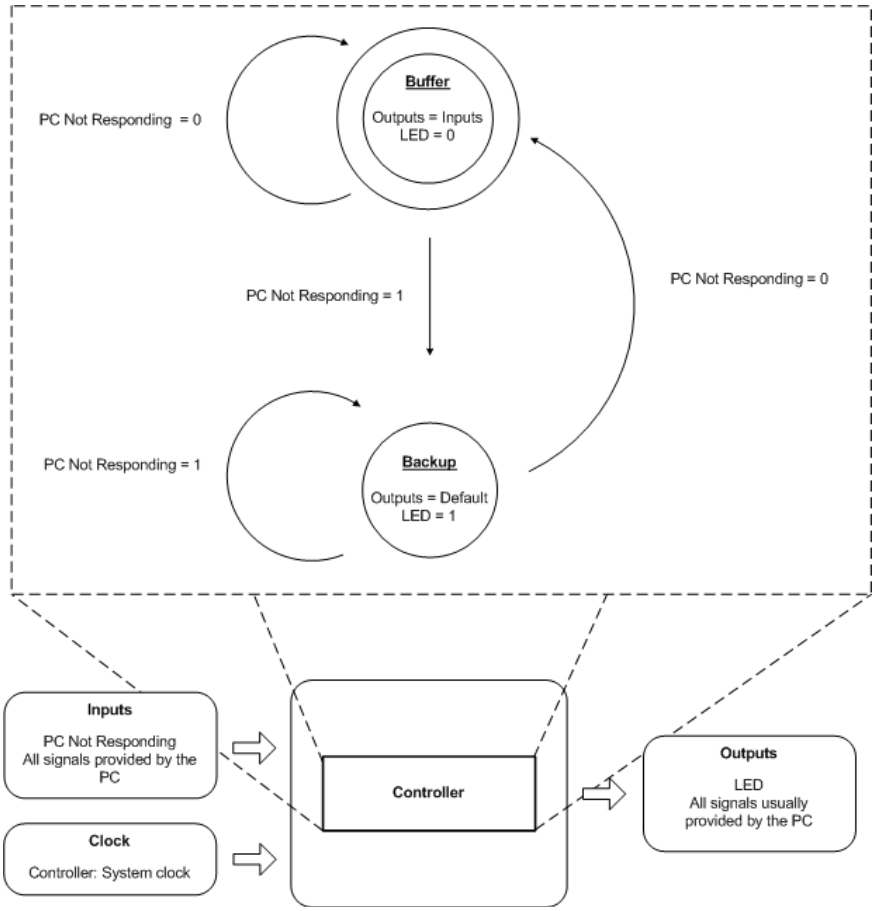


Figure 4.9.0.1: Structure of the PC Backup

5. Simulator hardware

The simulator hardware is made of two components: the real-time clock generator and the UART controller. The UART controller is described in details in the communication section.

A color coding scheme is used for the system hardware as shown in table 5.0.0.0

Color	Description
Red	Input from the board
Blue	External output signal to the system
Orange	Signal provided by the UART controller and that was received from the PC

Table 5.0.0.0: Color codes used for controllers

5.1. RTC Generator

The RTC Generator is the part of the simulator responsible for controlling the current time of the system.

Inputs and outputs

The inputs and outputs of the real-time clock generator are shown in figure 5.1.0.0.

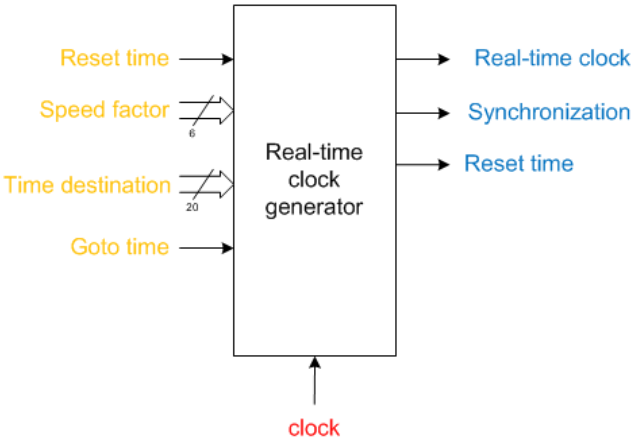


Figure 5.1.0.0: Inputs and outputs of real-time clock generator

Internal design

The real-time clock generator contains four counters, one each for the seconds, minutes, hours, and day of the week. Figure 5.1.0.1 shows the layout of the internal clocks.

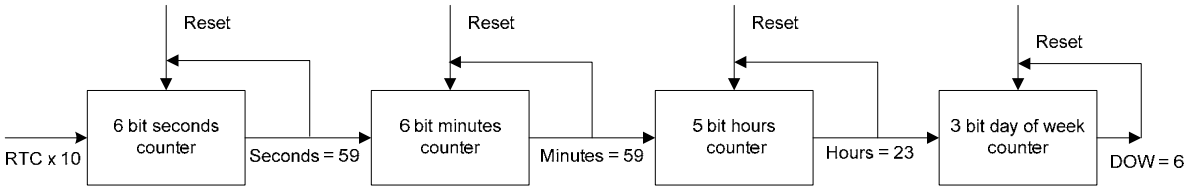


Figure 5.1.0.1: Structure of the real-time clock generator

Figure 5.1.0.2 shows the finite state machine of the RTC generator. Manual mode simply outputs an RTC signal of 10 Hz, times a speed factor set by the simulation software. If the simulation software wants to update the time, the generator enters update mode to copy a new time into the counter registers. Once copied, the generator sends the new time to the RTC receiver on the system.

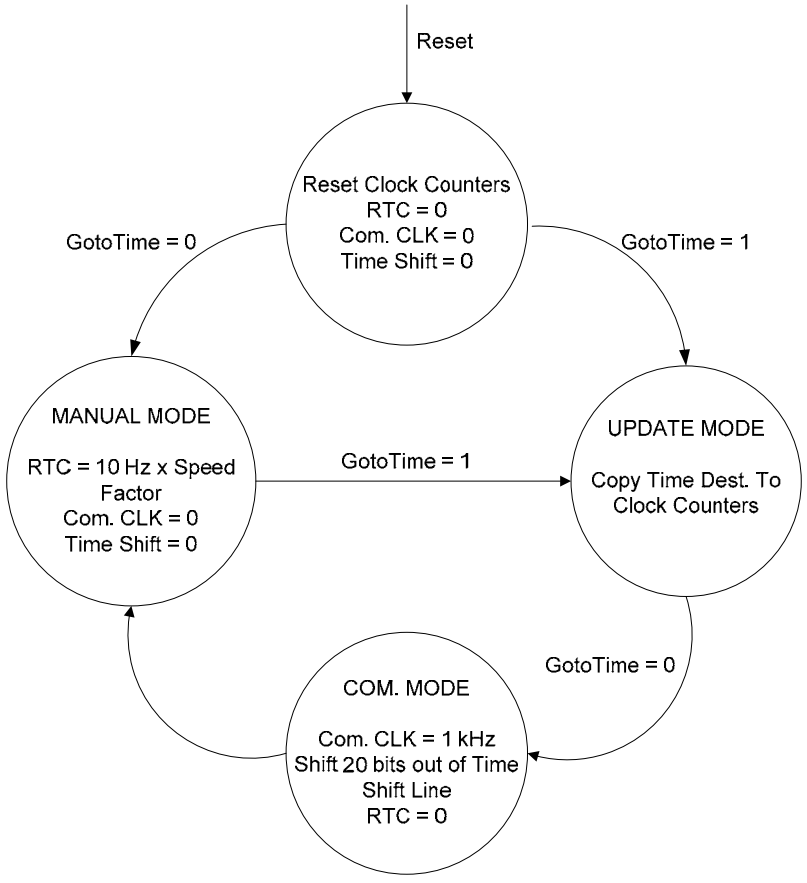


Figure 5.1.0.2: Finite-state machine of the RTC generator

6. System software

6.1. Classes diagram

The system software classes diagram is shown in figure 6.1.0.0.

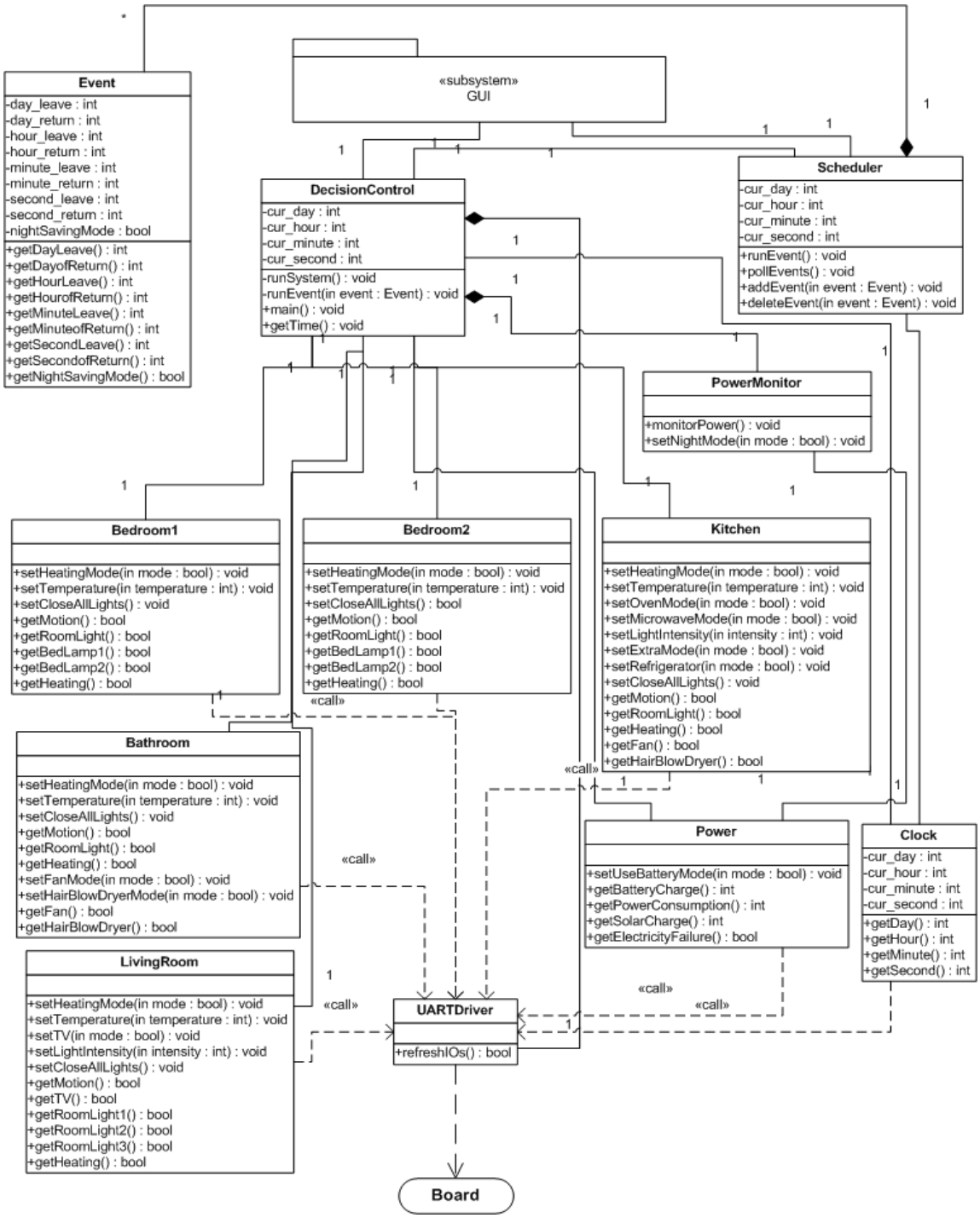


Figure 6.1.0.0: Software classes diagram for the decision-making software

6.2. Classes table

6.2.1 Decision making software

The different classes used in the decision making software have the different methods, attributes and roles as shown in table 6.2.1.0.

Name of Class	Attributes	Methods	Role
DecisionControl	**int cur_day;** Integer to store the current day **int cur_hour;** Integer to store the current hour **int cur_minute;** Integer to store the current minute **int cur_second;** Integer to store the current second	**private void runSystem();** This method is used to run the whole system. **private void runEvent(Event event);** Uses the attributes of a given event to change the system accordingly. **private void getTime();** This method is used to update the stored time attributes. **public static void main();** The main method runs the whole system process.	This class is the crux of the decision making architecture. It has knowledge of all the other classes specified below. It is contains the *main()* function which essentially runs the whole system process using the *runSystem()* method. This method calls on functions of the Scheduler and PowerMonitor classes below in seperate threads to control the whole system. Decisions are made as specified in the SRS.
Scheduler	**private ArrayList<Event> Events;** ArrayList used to store all the scheduled events. **int cur_day;** Integer to store the current day **int cur_hour;** Integer to store the current hour **int cur_minute;** Integer to store the current minute **int cur_second;** Integer to store the current second	**public void pollEvents();** Continuously polls events to check if it is time for them to be run **addEvent(Event event);** Add an event to our Event list **deleteEvent(Event event);** Delete an event from our Event list **private void getTime();** This method is used to update the stored time attributes.	This class is where all the scheduling is done. It stores a list of events, of the Event class and simply checks to see when an event is to be run (using *pollEvents()*).
Event	**int day_leave, day_return;** Integer to store the day of departure and return **int hour_leave, hour_return;** Integer to store the hour of departure and return **int minute_leave,**	**public int getDayLeaving();** **public int getDayofReturn();** Returns the day of the event **public int getHourLeaving();** **public int getHourofReturn();** Returns the hour of the event **public int getMinuteLeaving();** **public int getMinuteofReturn();** Returns the minute of the event	This class is used to represent a scheduled event. Based on the SRS, this is either an event where the user states when he is not away from home, or wishes to start night saving mode.

Name of Class	Attributes	Methods	Role
	minute_return; Integer to store the minute of departure and return **int second_leave, second_return;** Integer to store the second of departure and return **boolean nightSavingMode;** Boolean to store if night saving mode has been set or not	**public int getSecondLeaving();** **public int getSecondofReturn();** Returns the second of the event **public boolean getNightSavingMode();** Returns if night saving mode has been set to true or false	
PowerMonitor	**None**	**public void monitorPower();** This method is used to constantly monitor the power consumption in the house, and set the battery usage as decribed in the SRS **public void setNightMode(boolean mode);** Used to set night saving mode, by modulating the power usage	This class is used to monitor the power consumption in the house. The class uses the Power class to access the power usage information on the board and make the necessary decisions.
Bathroom	**None**	The following *Mutator* methods are used to change the room attributes: **static public void setHeatingMode(boolean mode);** False: Heating unit is OFF / True: Heating unit is set to maintain temperature to desired level **static public void setFanMode(boolean mode);** Desired temperature (any positive or negative number) **static public void setHairBlowDryerMode(boolean mode);** False: Turn off fan and force to stay off / True: Allow fan to be on **static public void setTemperature(int temperature);** False: Turn off Hair blow dryer and force to stay off / True: Allow hair blow dryer to be on The following *Accessor* methods are used to access the room attributes: **static public Boolean getMotion();** False: No motion in the room (for more than 5 min) / True: Motion in the room **static public Boolean getRoomlight();** False: Room light is OFF / Room light is ON **Static Public Boolean getFan();** False: Fan is OFF / True: Fan is ON	This is a static class used to represent the Bathroom and its corresponding attributes. It uses the RS232 driver to communicate with the System board.

Name of Class	Attributes	Methods	Role
		Static Public Boolean getHairBlowDryer(); False: Hair blow dryer is OFF / True: Hair blow dryer is ON	
Bedroom1	**None**	The following *Mutator* methods are used to change the room attributes: **static public void setHeatingMode(boolean mode);** False: Heating unit is OFF / True: Heating unit is set to maintain temperature to desired level **static public void setTemperature(int temperature);** Desired temperature (any positive or negative number) The following *Accessor* methods are used to access the room attributes: **static public boolean getMotion();** False: No motion in the room (for more than 5 min) / True: Motion in the room **static public boolean getRoomlight();** False: Room light is OFF / Room light is ON **static public boolean getBedlamp1();** False: Bed Lamp 1 is OFF / True: Bed Lamp 1 is ON **static public boolean getBedlamp2();** False: Bed Lamp 2 is OFF / True: Bed Lamp 2 is ON **static public boolean getHeating();** False: Heating is OFF / True: Heating is ON	This is a static class used to represent the first Bedroom and its corresponding attributes. It uses the RS232 driver to communicate with the System board.
Bedroom2	**None**	The following *Mutator* methods are used to change the room attributes: **static public void setHeatingMode(boolean mode);** False: Heating unit is OFF / True: Heating unit is set to maintain temperature to desired level **static public void setTemperature(int temperature);** Desired temperature (any positive or negative number) The following *Accessor* methods are used to access the room attributes: **static public boolean getMotion();** False: No motion in the room (for more than 5 min) / True: Motion in the room **static public boolean getRoomlight();** False: Room light is OFF / Room light is ON **static public boolean getBedlamp1();** False: Bed Lamp 1 is OFF / True: Bed Lamp 1 is	This is a static class used to represent the second Bathroom and its corresponding attributes. It uses the RS232 driver to communicate with the System board.

Name of Class	Attributes	Methods	Role
		ON **static public boolean getBedlamp2();** False: Bed Lamp 2 is OFF / True: Bed Lamp 2 is ON **static public boolean getHeating();** False: Heating is OFF / True: Heating is ON	
Kitchen	**None**	The following *Mutator* methods are used to change the room attributes: **static public void setHeatingMode(boolean mode);** False: Heating unit is OFF / True: Heating unit is set to maintain temperature to desired level **static public void setTemperature(int temperature);** Desired temperature (any positive or negative number) **static public void setLightIntensity(int intensity);** Desired light intensity (any positive number between 0 and 100) **static public void setOvenMode(boolean mode);** False: Turn off oven and force to stay off / True: Allow oven to be on **static public void setMicrowaveMode(boolean mode);** False: Turn off microwave and force to stay off / True: Allow microwave to be on **static public void setExtraMode(boolean mode);** False: Turn off extra accessory and force to stay off / True: Allow extra accessory to be on The following *Accessor* methods are used to access the room attributes: **static public boolean getMotion();** False: No motion in the room (for more than 5 min) / True: Motion in the room **static public boolean getRoomlight1();** False: Room light 1 is OFF / True: Room light 1 is ON **static public boolean getRoomlight2();** False: Room light 2 is OFF / True: Room light 2 is ON **static public boolean getRoomlight3();** False: Room light 3 is OFF / True: Room light 3 is ON **static public boolean getOven();** False: Oven is OFF / True: Oven is ON **static public boolean getExtra();**	This is a static class used to represent the Kitchen and its corresponding attributes. It uses the RS232 driver to communicate with the System board.

Name of Class	Attributes	Methods	Role
		False: Extra accessory is OFF / True: Extra accessory is ON **static public boolean getMicrowave();** False: Microwave is OFF / True: Microwave is ON **static public boolean getRefrigerator();** False: Refrigerator is OFF / True: Refrigerator is ON	
LivingRoom	**None**	The following *Mutator* methods are used to change the room attributes: **static public void setHeatingMode(boolean mode);** False: Heating unit is OFF / True: Heating unit is set to maintain temperature to desired level **static public void setTemperature(int temperature);** Desired temperature (any positive or negative number) **static public void setLightIntensity(int intensity);** Desired light intensity (any positive number between 0 and 100) **static public void setTV(boolean mode);** False: Turn off TV and force to stay off / True: Allow TV to be on The following *Accessor* methods are used to access the room attributes: **static public boolean getMotion();** False: No motion in the room (for more than 5 min) / True: Motion in the room **static public boolean getRoomlight1();** False: Room light 1 is OFF / True: Room light 1 is ON **static public boolean getRoomlight2();** False: Room light 2 is OFF / True: Room light 2 is ON **static public boolean getRoomlight3();** False: Room light 3 is OFF / True: Room light 3 is ON **static public boolean getTV();** False: TV is OFF / True: TV is ON	This is a static class used to represent the Living room and its corresponding attributes. It uses the RS232 driver to communicate with the System board.
Power	**None**	The following *Mutator* methods are used to change the power attributes: **static public void setUseBattery(boolean mode);** True: battery is discharged / False: battery is charged	This is a static class is used to represent the attributes of the power consumption in the house. It uses the RS232 driver to communicate with the System board.

Name of Class	Attributes	Methods	Role
		The following *Accessor* methods are used to access the room attributes: **static public int getBatteryCharge();** Return the battery charge (in kWh) **static public int getPowerConsumption();** Return the power consumption (in Watts) **static public int getSolarCell();** Return the power produced by solar cell (in Watts) **static public boolean getElectricityFailure();** Return electricity failure signal (true: failure, false: normal)	
Clock	**int cur_day;** **int cur_hour;** **int cur_minute;** **int cur_second;** The above attributes are used to represent the current time.	The following *Accessor* methods are used to access the current time: **static public int getDay();** **static public int getHour();** **static public int getMinute();** **static public int getSecond();**	This static class is used to represent the Real time clock on the board. It uses the RS232 driver to communicate with the System board.
UARTDriver		**refreshIOs();** Communicate with the board to apply modifications and retrieve updated status Returns true if the operation was done normally, returns false if an error occured	

Table 6.2.1.0: Classes table for the system decision-making software

6.2.2 GUI

The different classes used in the GUI software have the different methods, attributes and roles as shown in table 6.2.2.0.

Name of class	Methods	Role
SS	**-public static void main()** This method runs the whole system ***** The following moethods set the color, size and title of the page: **-public void-setDefaultCloseOperation()** **-public void getContentPane()** **-public void setSize()** **-public void setVisible()** *****The following methods are used to set 'System Manager' as the title of the interface with the proper alignments: **-public void setHorizontalTextPosition()** **-public void setHorizontalAlignment()** **-public void setForeground()** **-public void setFont()** **-public void getContentPane()** **-public void setBounds()** ***** The following methods are used create a label for the controls of Bedroom1, Bedroom2, Bathroom, Kitchen, Living Room, Power display box, Power produced by solar cell display box, Power from Hydro Quebec display, Energy stored in battery display, Total power consumption display, Set the text filed for the user to enter the desired temperature in a certain location in the appretment : **-public void setHorizontalTextPosition()** **-public void setText()** **-public void setForeground()** **-public void setFont(new Font())** **-public void getContentPane()** **-public void setBounds()** *****The following methods are used to set the progress bars and set the checkbox, button and scroll bar : **-public void setMinimum()** **-public void setMaximum()** **-public void setValue()** **-public void getValue()**	**This class is used to create a panel for the system simulator interface where the user can insert the required input, and in turn the system simulator reads and displays outputs of the system.**

	***The following were used to catch any movement of the mouse: **-public void adjustmentValueChanged()** **-public void getSource()** **-public void setLabelFor()** **-public void setDisabledTextColor()**	
MyComponent	***** The following methods are used to create oval and rectangular shapes seen in the system manager interface: **-public void drawOval()** **-public void drawRect()** ***** The following methods are used to fill the ovular and rectangular shapes with color: **-public void setColor()** **-public void fillOval()**	**This class is also used to set the panel for the interface where a design for the interface the user will see is created using different methods**

Table 6.2.2.0: Classes table for the system GUI

6.3. Interface

The interface used to communicate with the user is shown in figure 6.3.0.0.

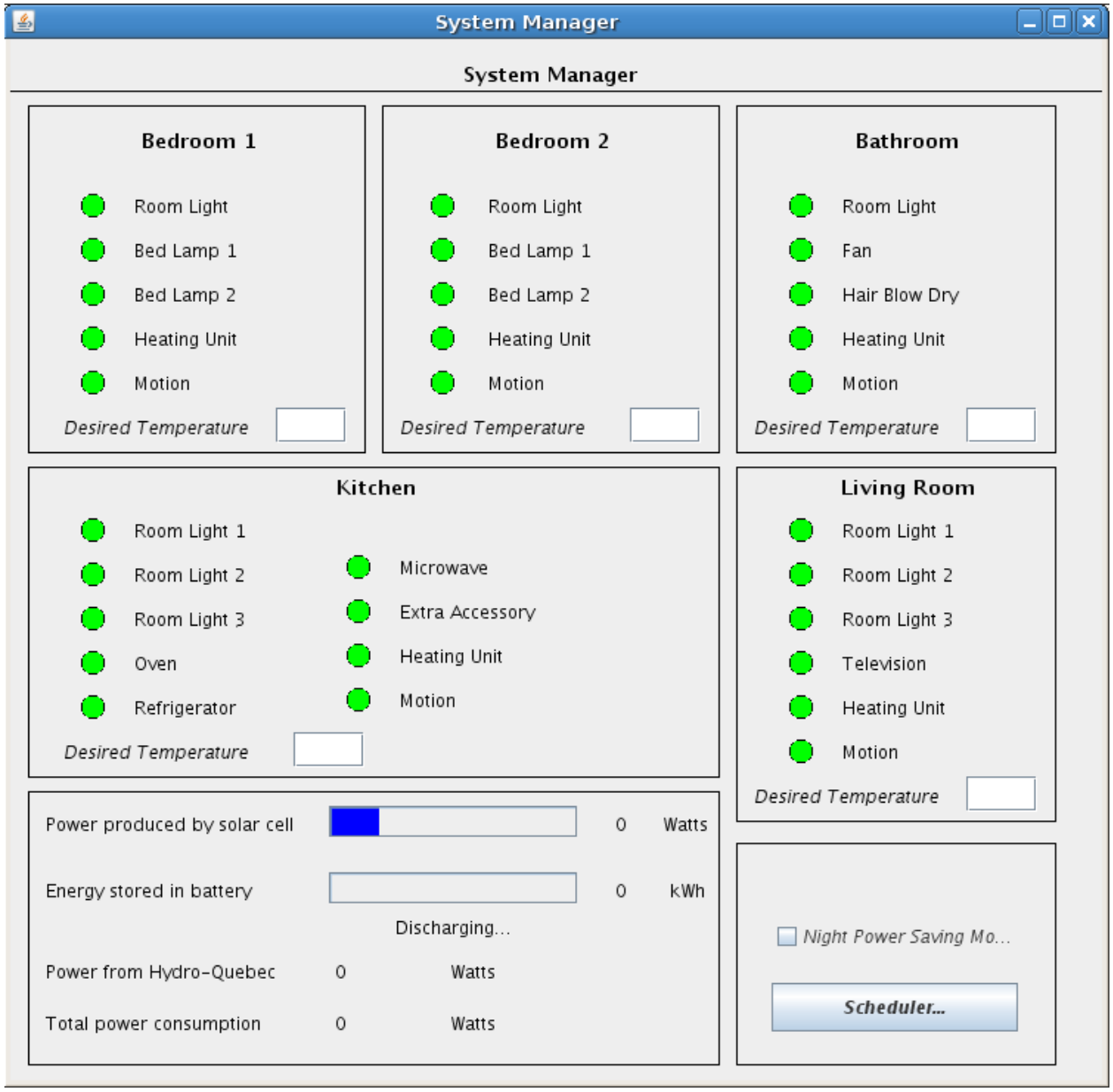


Figure 6.3.0.0: Interface preview for the decision making system

6.4. Program flow

The general program flow is shown in figure 6.4.0.0.

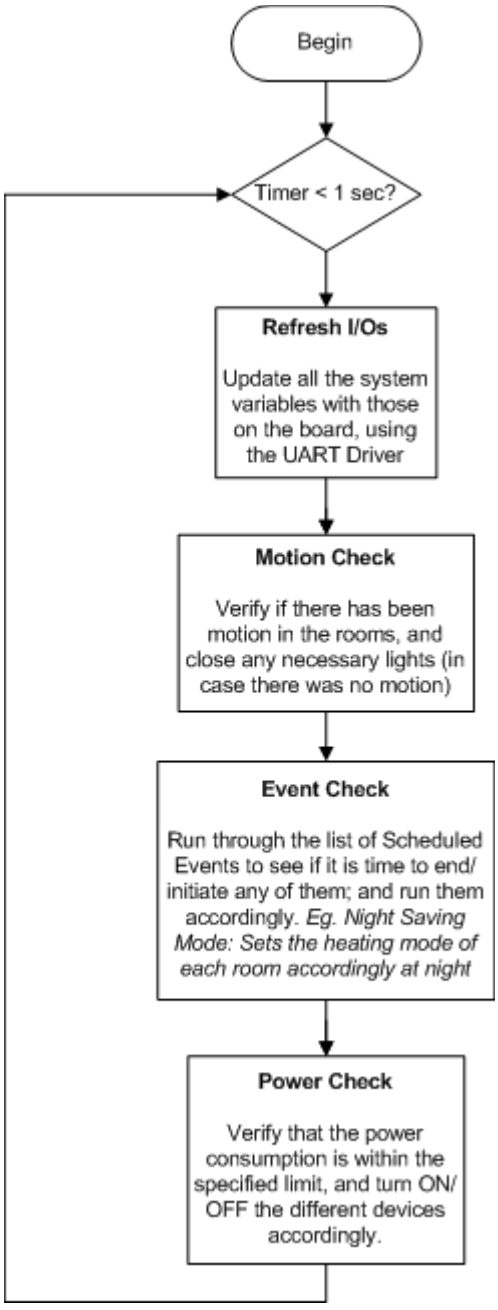


Figure 6.4.0.0: Decision-making software program flow

7. Simulator software

7.1. Classes diagram

The system software classes diagram is shown in figure 7.1.0.0.

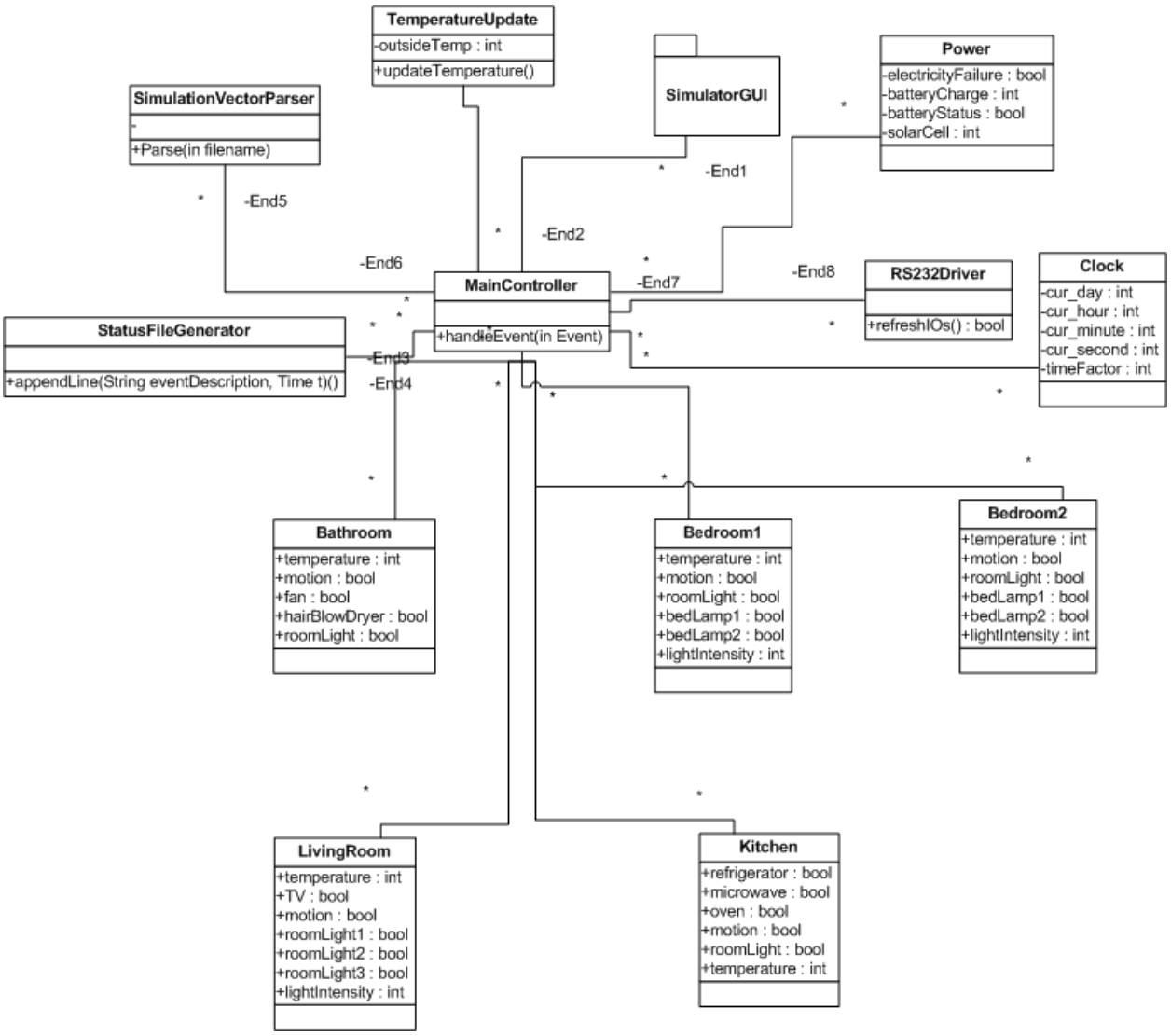


Figure 7.1.0.0: Classes diagram for the environment simulation software

7.2. Classes table

7.2.1 Environment simulation software

The different classes used in the environment simulation software have the different methods, attributes and roles as shown in table 7.2.1.0.

Name of Class	Attributes	Methods	Role
MainController		**public static handleEvent()**	sets the appropriate attributes based on the event
SimulationVectorParses		**public static parse(String filename)**	Parses the simulation vector and generates events to be handled by the controller
StatusFileGenerator		**public static appendLine(String event, Time t)**	creates the status file based on the system
Temperature Update	**int outsideTemperature;**	**public static updateTemperature()**	thread that periodically updates the temperature in each room, based on Newton's Law of Heating/Cooling
Bathroom	**static bool motion;** Boolean used to store if there is motion in the room **static bool roomLight;** Boolean used to store if the room light is on or off **static bool fan;** Boolean used to store if the fan is on or off **static bool hairBlowDryer** Boolean used to store if the hair dryer is on or off **static int temperature;** Integer used to store the temperature in the room	**None**	This is a static class used to represent the Bathroom and its corresponding attributes. It uses the RS232 driver to communicate with the System board.
Bedroom1	**static bool motion**; Boolean used to store if there is motion in the room **static bool roomLight;** Boolean used to store if the room light is on or off **static bool bedlamp1;** Boolean used to store if the first bed lamp is on or off **static bool bedlamp2;** Boolean used to store if the second bed lamp is on or off **static int temperature ;**	**None**	This is a static class used to represent the first Bedroom and its corresponding attributes.

Name of Class	Attributes	Methods	Role
	Integer used to store the temperature in the room **static int lightIntensity;** Integer used to store the light intensity in the room		
Bedroom2	**static bool motion;** Boolean used to store if there is motion in the room. **static bool roomLight;** Boolean used to store if the room light is on or off **static bool bedlamp1;** Boolean used to store if the first bedlamp is on or off **static bool bedlamp2;** Boolean used to store if the second bedlamp light is on or off **static int temperature ;** Integer used to store the temperature in the room **static int lightIntensity;** Integer used to store the light intensity in the room	**None**	This is a static class used to represent the second Bedroom and its corresponding attributes.
Kitchen	**static bool refrigerator;** Boolean used to store if the refrigerator is on or off **static bool oven;** Boolean used to store if the oven is on or off **static bool microwave;** Boolean used to store if the microwave is on or off **static bool motion** ; Boolean used to store if there is motion in the room **static bool roomLight1;** Boolean used to store if the first room light is on or off **static bool roomLight2** Boolean used to store if the second room light is on or off **static bool roomLight3;** Boolean used to store if the third room light is on or off **static int temperature;** Integer used to store the temperature in the room	**None**	This is a static class used to represent the Kitchen and its corresponding attributes.
LivingRoom	**static bool TV;** Boolean used to store if the TV is on or off **static bool motion;**	**None**	This is a static class used to represent the Livingroom and its corresponding attributes.

Name of Class	Attributes	Methods	Role
	Boolean used to store if there is motion in the room **static bool roomLight1;** Boolean used to store if the first room light is on or off **static bool roomLight2** Boolean used to store if the second room light is on or off **static bool roomLight3;** Boolean used to store if the third room light is on or off **static int temperature ;** Integer used to store the temperature in the room **static int lightIntensity;** Integer used to store the light intensity in the room		
Power	**static bool electricityFailure;** False if HydroQuebec is able to provide power, true if there is an electricity failure **static int batteryCharge**; Battery charge remaining (in Wh) **static int solarCellPower;** Power produced by the SolarCell (in Wats) **static bool batteryStatus;** The state of the battery, true if charging, false if discharging	**None**	This is a static class used to encapsulate the power producing and consuming units in the house.
Clock	**static int cur_day;** The current day **static int cur_hour;** The current hour **static int cur_minute;** The current minute **static int cur_second;** The current second **static int timeFactor**; The time rate multiplier	**None**	This class represents the time and the day, as well as the rate of simulation.
Event (alternate)	**Time start_time** **Time end_time** **String room** **String eventAction**	**None**	This class represents an event.(For example, at time 12:40:25, the light is turned on in the kitchen.)

Table 7.2.1.0: Classes table for the environment simulation software

7.2.2 GUI

The different classes used in the GUI software have the different methods, attributes and roles as shown in table 7.2.2.0.

Name of class	Methods	Role
SM	**public static void main();** This method runs the whole system Set the color, size and title of the page: **-public void-setDefaultCloseOperation()** **-public void getContentPane()** **-public void setSize()** **-public void setVisible()** Set 'System Manager' as the title of the interface with the proper alignments: **-public void setHorizontalTextPosition()** **-public void setHorizontalAlignment()** **-public void setForeground()** **-public void setFont()** **-public void getContentPane()** **-public void setBounds()** Create a label for the controls of Bedroom1, Bedroom2, Bathroom, Kitchen, Living Room, Power display box, Power produced by solar cell display box, Power from Hydro Quebec display, Energy stored in battery display, Total power consumption display, Set the text filed for the user to enter the desired temperature in a certain location in the apartment: **-public void setHorizontalTextPosition()** **-public void setText()** **-public void setForeground()** **-public void setFont(new Font())** **-public void getContentPane()** **-public void setBounds()** Create the buttons for Bedroom1, Bedroom2, Bathroom, Kitchen, Living Room, Reset time, load simulation and radio buttons: **-public void setText()** **-public void setFont(new Font())** **-public void getContentPane()** **-public void setBounds()**	This class is used to create a panel for the system manager interface where the room desired temperature, night power saving mode and scheduler can be modified. While all other values are status values and are read-only values. It contains the main function that runs the interface.

Name of class	Methods	Role
	Catch any movement of the mouse: **-public void adjustmentValueChanged()** **-public void getSource()** **-public void setLabelFor()** **-public void setDisabledTextColor()**	
My Component	Create rectangular shapes seen in the system manager interface: **public void drawRect()** Fill the rectangular shapes with color: **public void setColor()**	This class is also used to set the panel for the interface where a design for the interface the user will see is created using different methods

Table 7.2.2.0: Classes table for the simulator GUI

7.3. Interface

The interface used to communicate with the user is shown in figure 7.3.0.0.

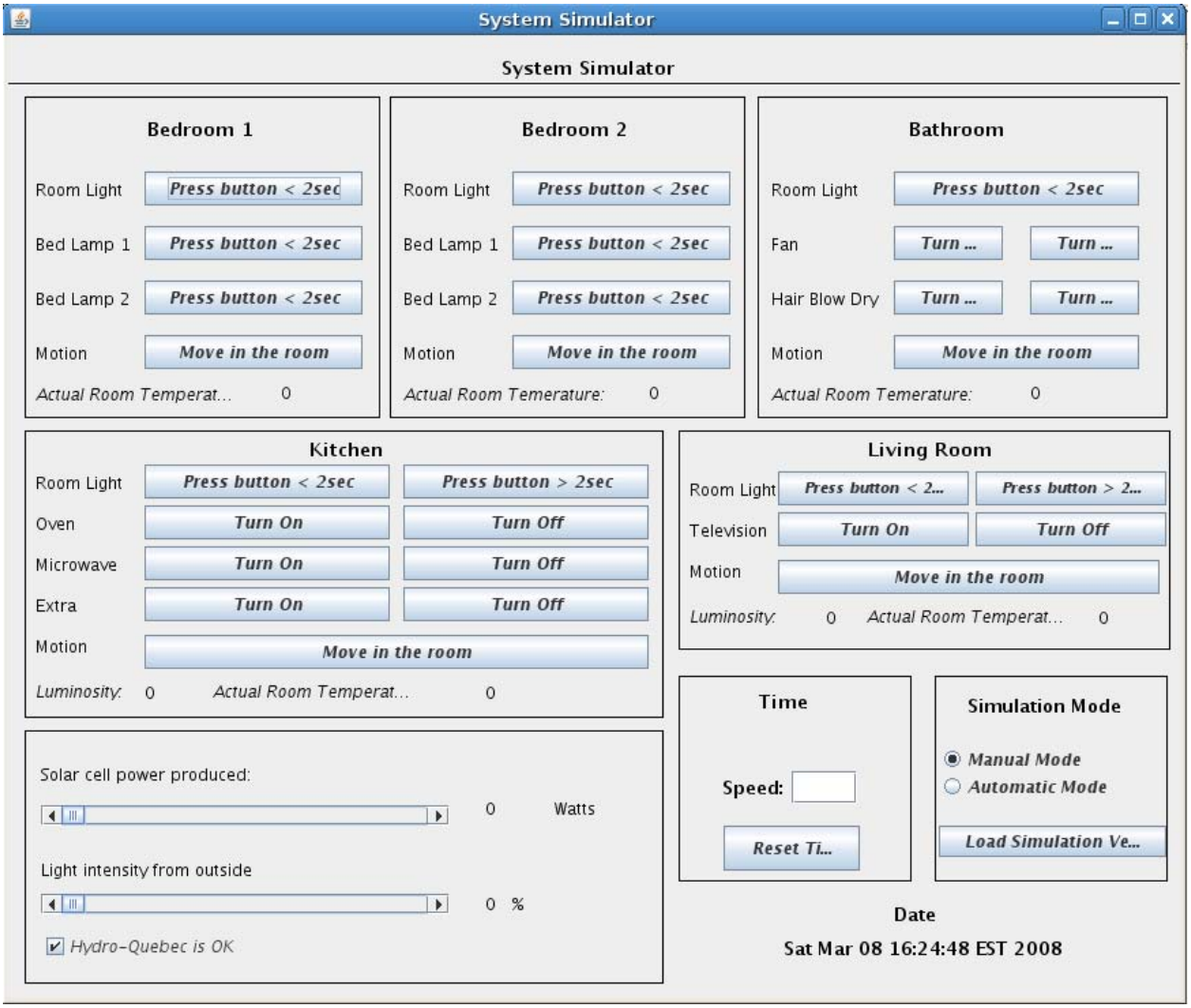


Figure 7.3.0.0: Interface preview for the simulator

7.4 Program flow

The general program flow is shown in figure 7.4.0.0.

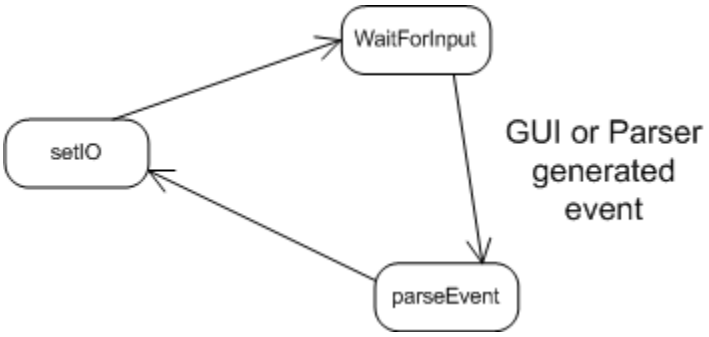


Figure 7.4.0.0: Environment simulation software program flow

8. Communication

8.1. Communication protocol

The communication between the PC and the board is done using the UART protocol. There is one start bit, two stop bits, no parity and 8 bits of data. At the end of each string there is a checksum byte. The baud rate is 19200. When the line is idle, it is set to 1. The communication is always initiated by the PC and the board replies according to the message received. There are 3 types of replies possible. The protocol is the same for both the system and the simulator.

8.1.1 Write new values to the board

The PC sends a string with different values to be updated on the board. The board receives the string, verifies the checksum and then updates its registers. When this is done, it sends a write acknowledge to the PC to confirm that the update was performed correctly. This is shown in the following figure 8.1.1.0.

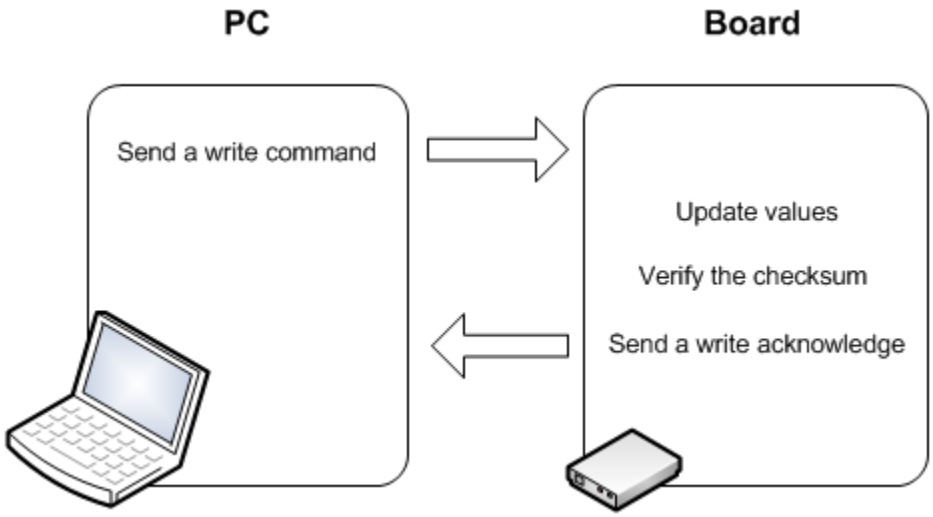


Figure 8.1.1.0: Communication sequence when writing new values to the board

8.1.2 Read values from the board

The PC sends a read request to the board. The board receives the string, verifies the checksum and then loads the values to be sent in its buffer. When this is done, it sends a string containing these values to the PC. This is shown in figure 8.1.2.0.

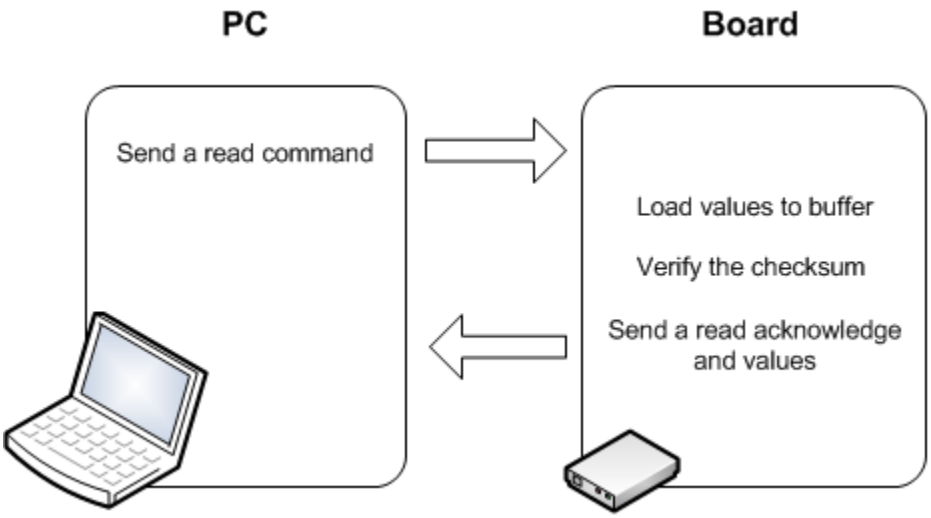


Figure 8.1.2.0: Communication sequence when reading values from the board

8.1.3 Error during communication

If the PC sends a string to the board and the checksum does not match what was received, then an error occurred during the transfer. The board then produces an error command and sends this back to the PC. The software then has to handle this case. It can resend the command to the board. This error handling is shown in figure 8.1.3.0.

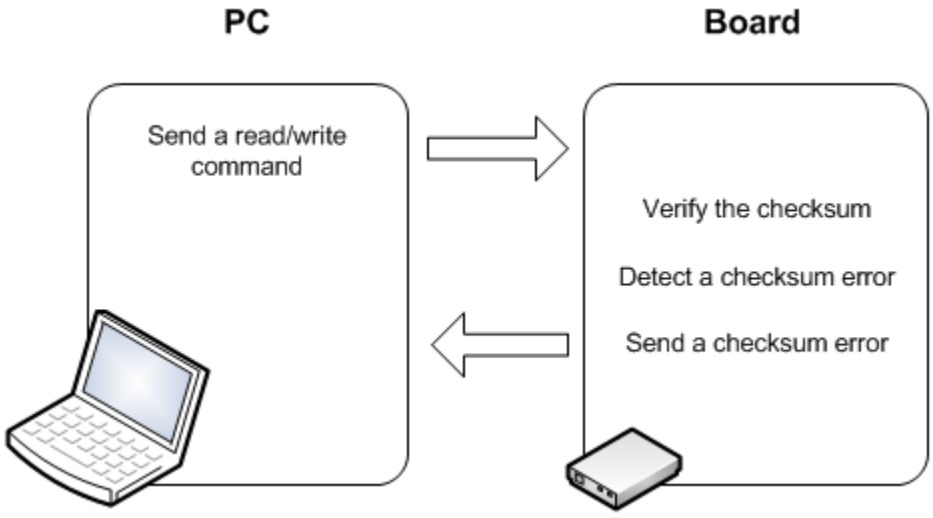


Figure 8.1.3.0: Communication sequence when there is an error during communication

8.2. Message encoding for the system

All the signals from the PC that are updated on the board are encoded in 6 bytes. There is also 1 byte for the command to be performed and 1 byte for the checksum. The total message length is therefore 8 bytes. All the signals from the board to the PC are encoded in 9 bytes. There is also 1 byte for the command to be performed and 1 byte for the checksum. The total message length is therefore 11 bytes.

8.2.1 Write new values to the board

When the PC needs to update the board registers, it sends a message of 8 bytes as shown in figure 8.2.1.0.

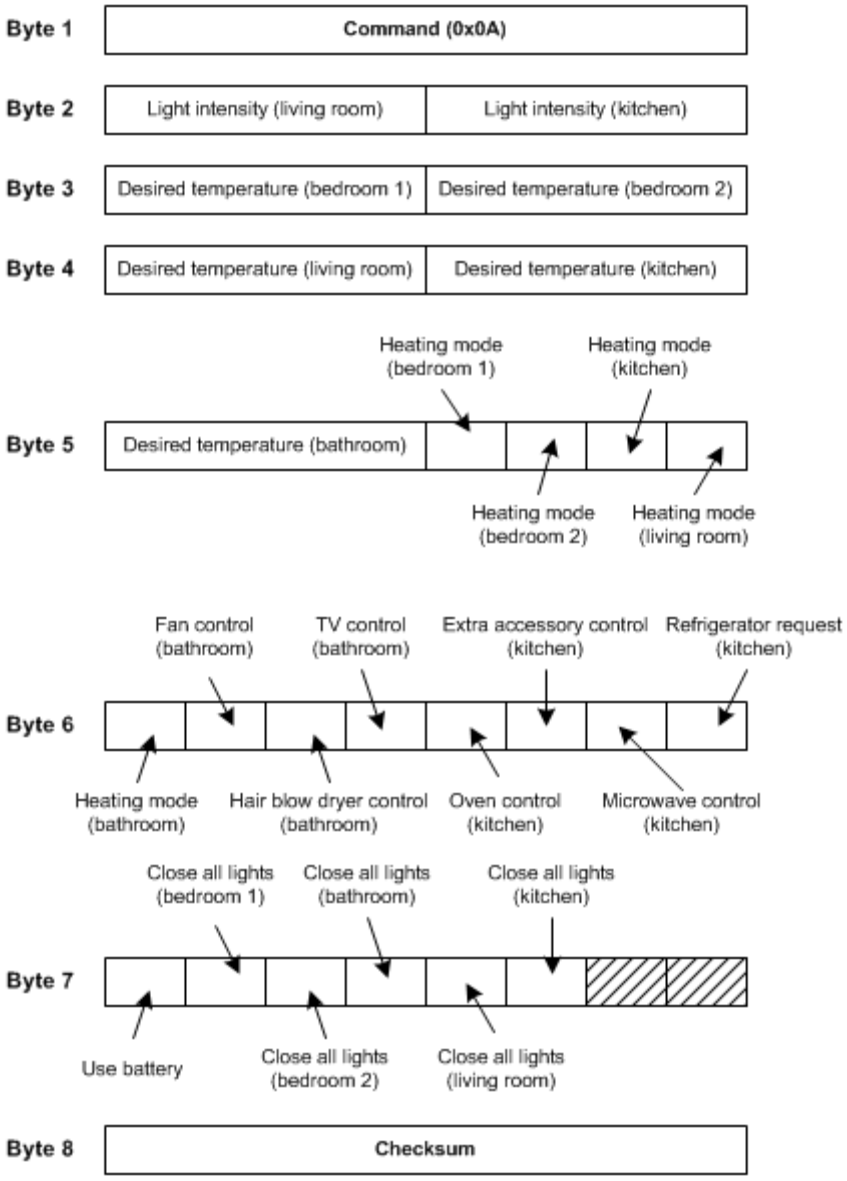


Figure 8.2.1.0: String for writing new values to the board

When this is done, the PC sends a write acknowledge. Since the usual format is 11 bytes, all the non used bytes are padded with zeros as shown in figure 8.2.1.1.

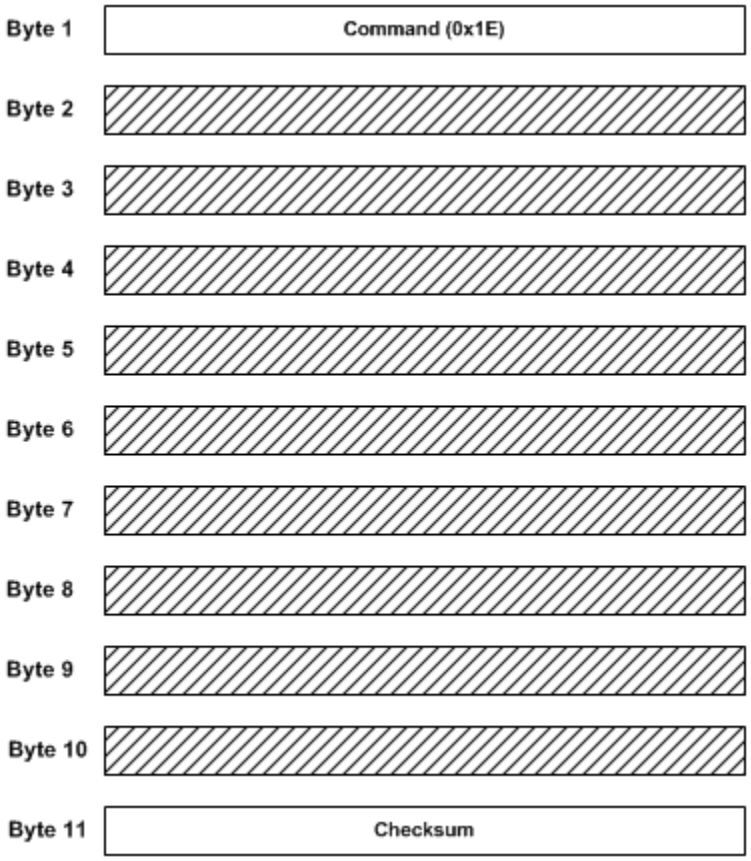


Figure 8.2.1.1: String for a write acknowledge

8.2.2 Read values from the board

When the PC needs to read the board registers, it sends a message of 8 bytes as shown in figure 8.2.2.0.

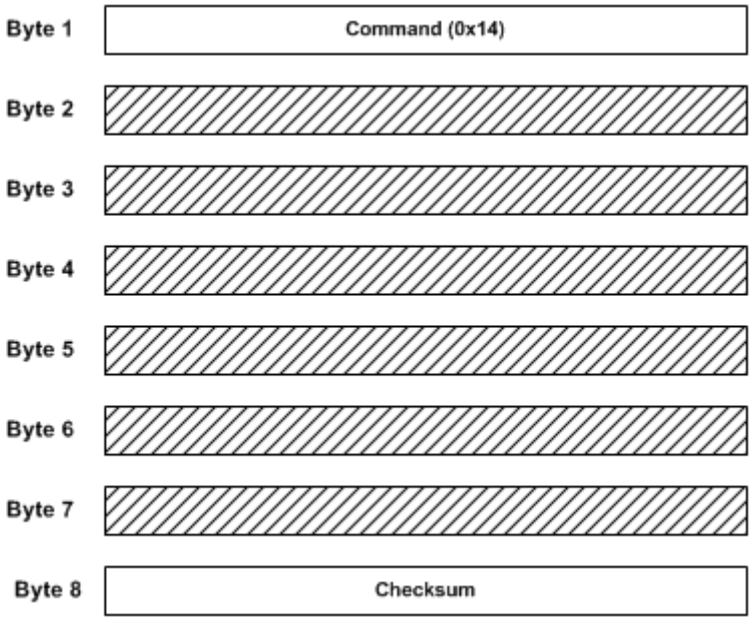


Figure 8.2.2.0: String for reading values from the board

The board then sends 11 bytes to the PC to provide the information required, as shown in figure 8.2.2.1.

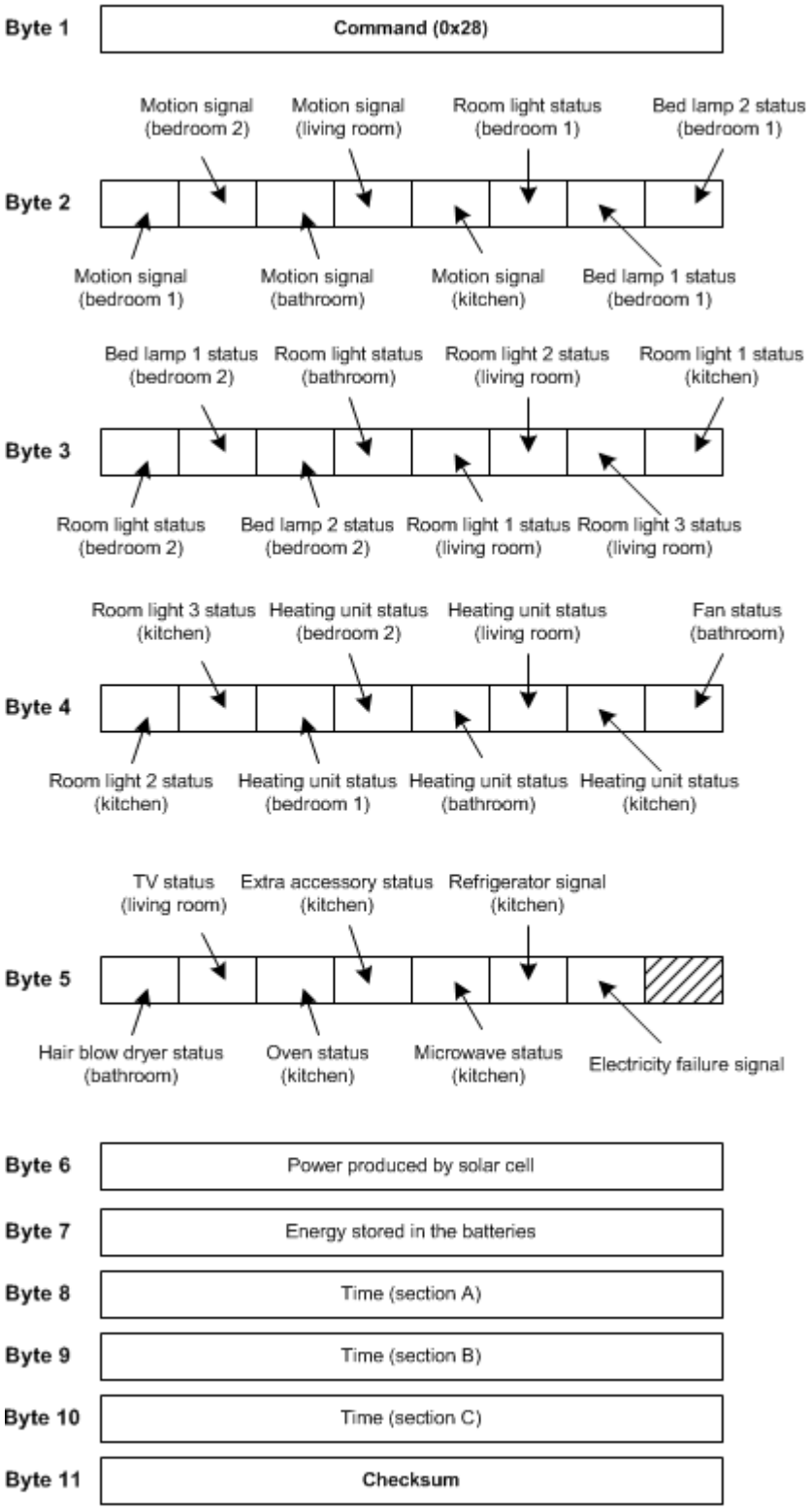


Figure 8.2.2.1: String for returning values to the board

8.2.3 Error during communication

When the PC sends a message to the board and the checksum does not match, the board sends a message of 11 bytes, as shown in figure 8.2.3.0.

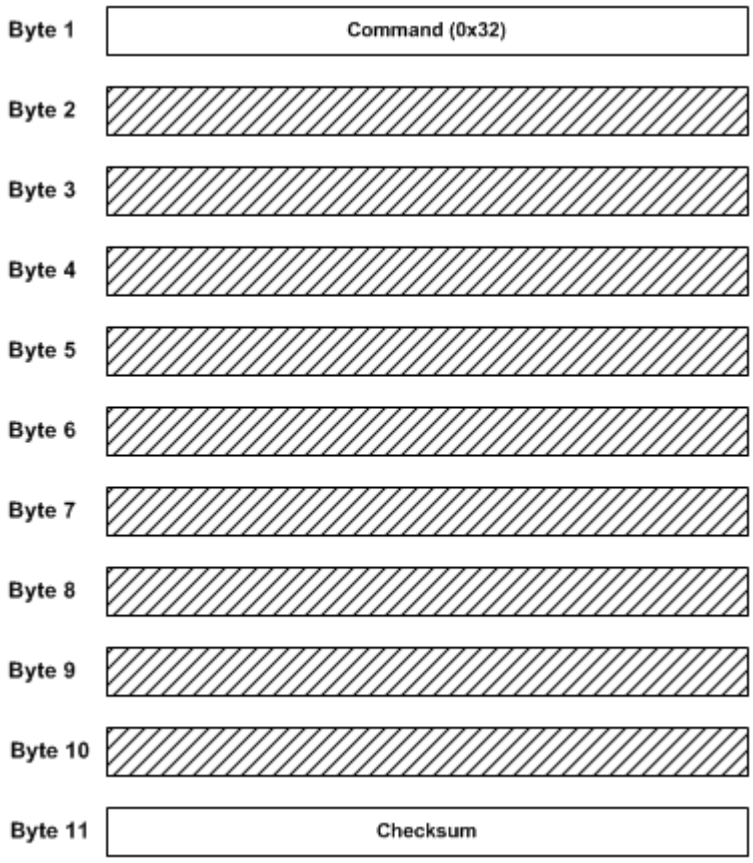


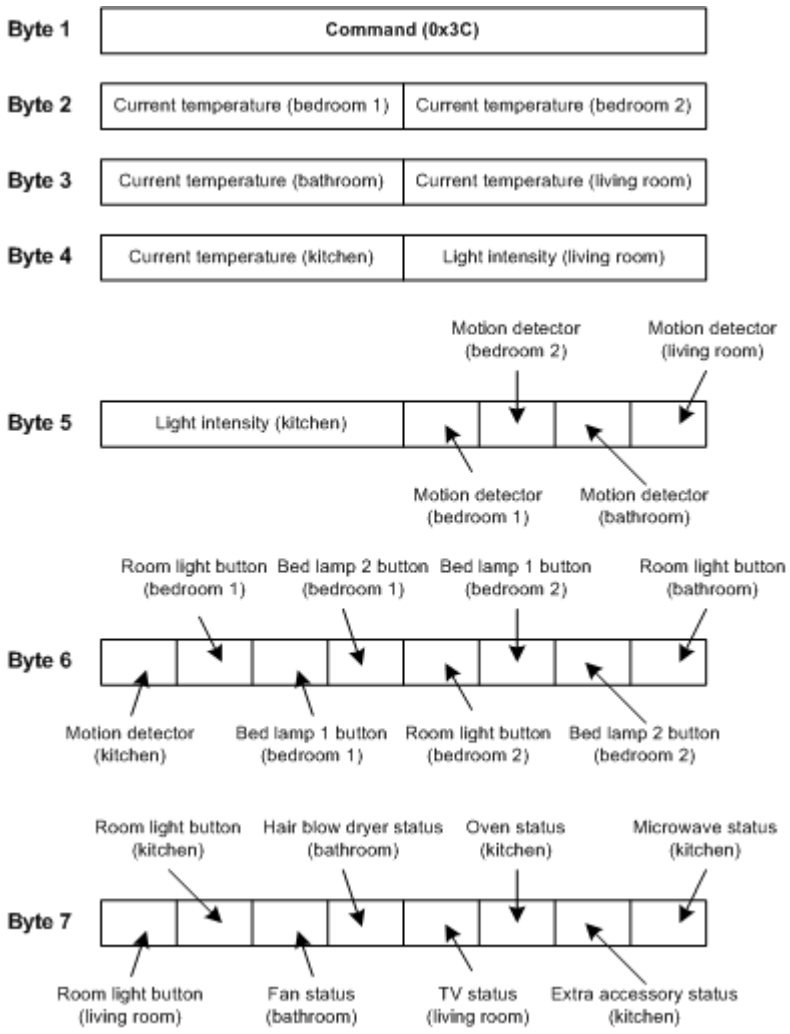
Figure 8.2.3.0: String for flagging a communication error

8.3. Message encoding for the simulator

All the signals from the PC that are updated on the board are encoded in 12 bytes. There is also 1 byte for the command to be performed and 1 byte for the checksum. The total message length is therefore 14 bytes. All the signals from the board to the PC are encoded in 4 bytes. There is also 1 byte for the command to be performed a 1 byte for the checksum. The total message length is therefore 6 bytes.

8.3.1 Write new values to the board

When the PC needs to update the board registers, it sends a message of 14 bytes as shown in figure 8.3.1.0.



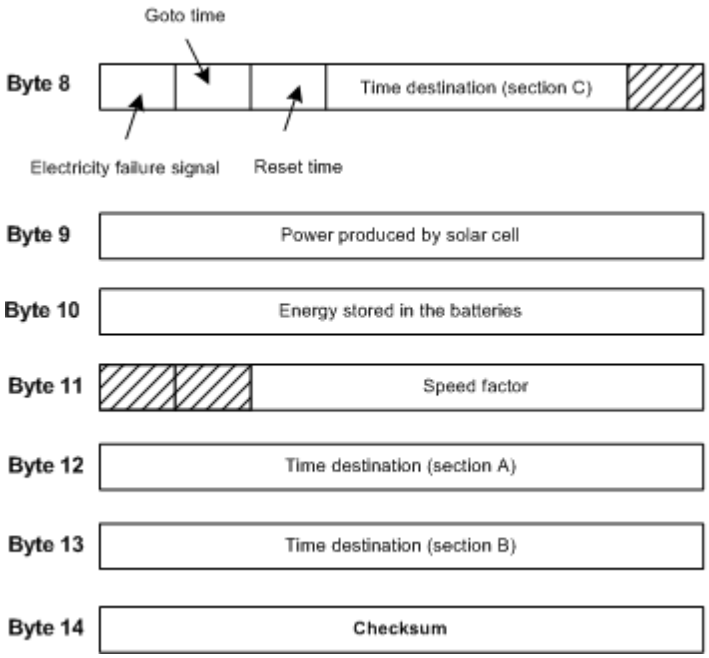


Figure 8.3.1.0: String for writing new values to the board

When this is done, the PC sends a write acknowledge. Since the usual format is 6 bytes, all the non used bytes are padded with zeros as shown in figure 8.3.1.1.

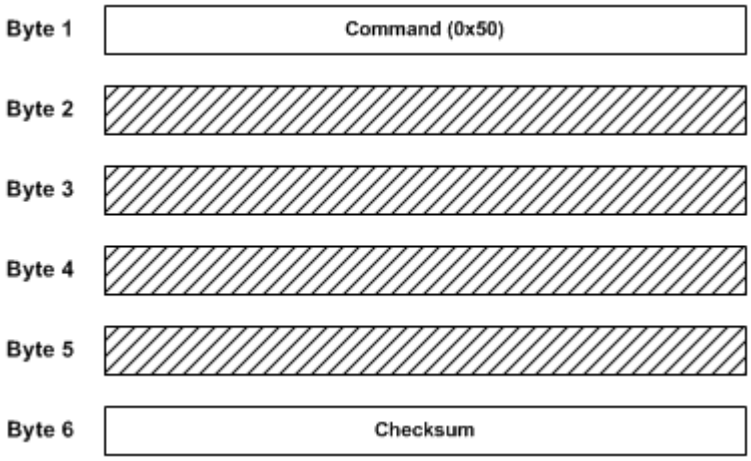


Figure 8.3.1.1: String for a write acknowledge

8.3.2 Read values from the board

When the PC needs to read the board registers, it sends a message of 14 bytes, as shown in figure 8.3.2.0.

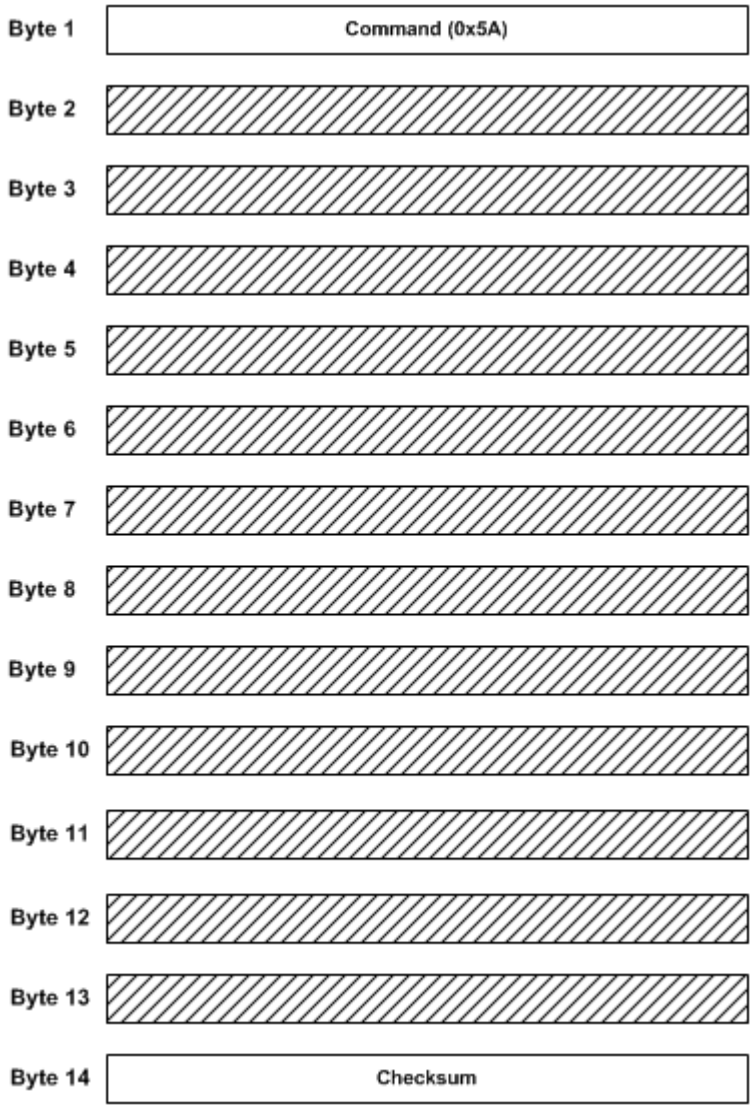


Figure 8.3.2.0: String for reading values from the board

The board then sends 6 bytes to the PC to provide the information required, as shown in figure 8.3.2.1.

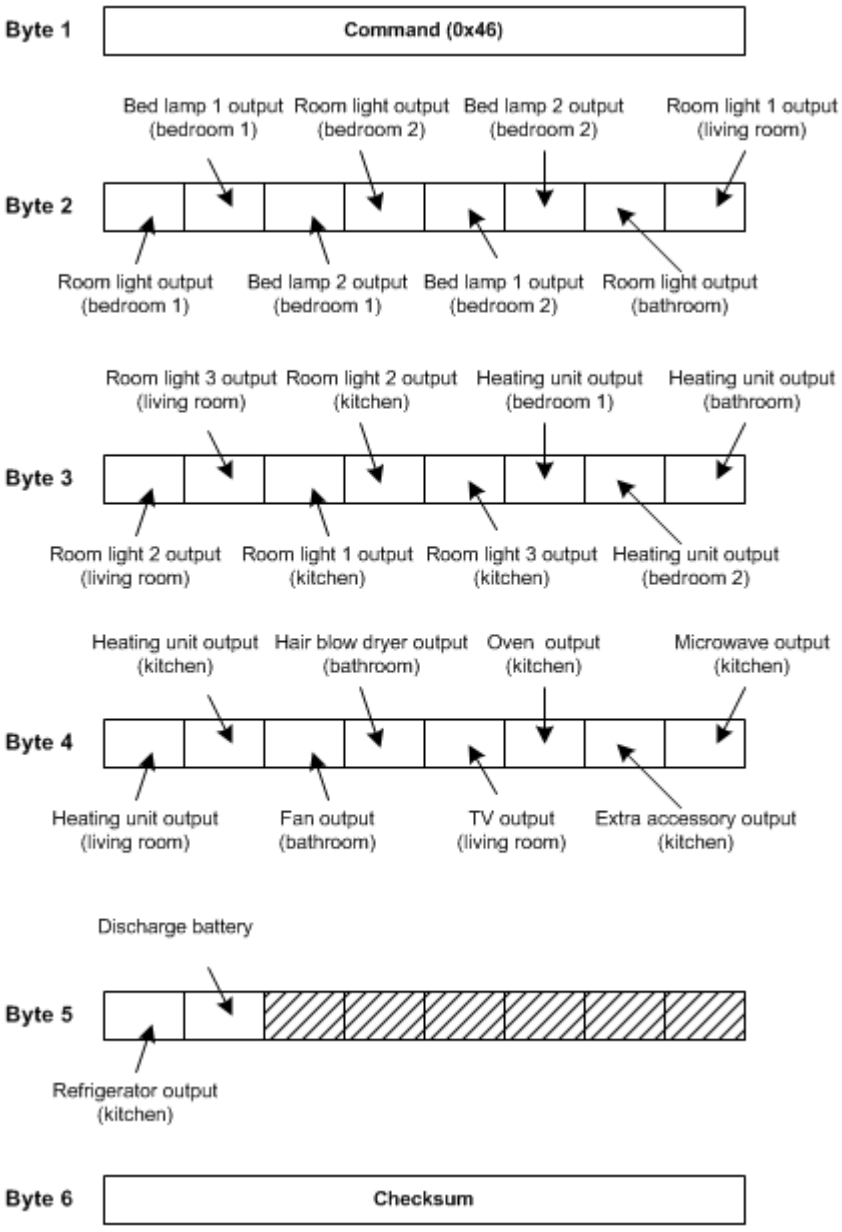


Figure 8.3.2.1: String for returning values to the board

8.3.3 Error during communication

When the PC sends a message to the board and the checksum does not match, the board sends a message of 6 bytes as shown in figure 8.3.3.0.

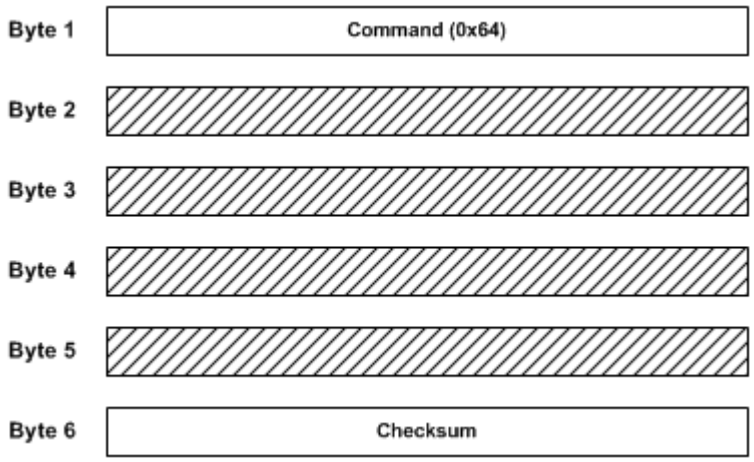


Figure 8.3.3.0: String for flagging a communication error

8.4. Implementation on the board

Inputs and outputs

The implementation of this serial communication is done with a controller. The implementation is the same for the system and the simulator except that the number of registers changes in the buffers. This is because the lengths of the messages are different. Except for that, everything else stays the same.

The controller is shown in figure 8.4.0.0. Inputs are data from the board to be sent to the PC. Outputs are data received from the PC. Each subcontroller is connected to an asynchronous reset pin (this is not showed on the figures).

System

The system UART controller communicates with the PC by receiving and sending bytes. Each bit of each byte is mapped to a specific signal. All these signals are sent to other controllers on the board and these controllers then process the given signals. The produced signals by the controllers are then sent to the UART controller and these are transmitted to the PC.

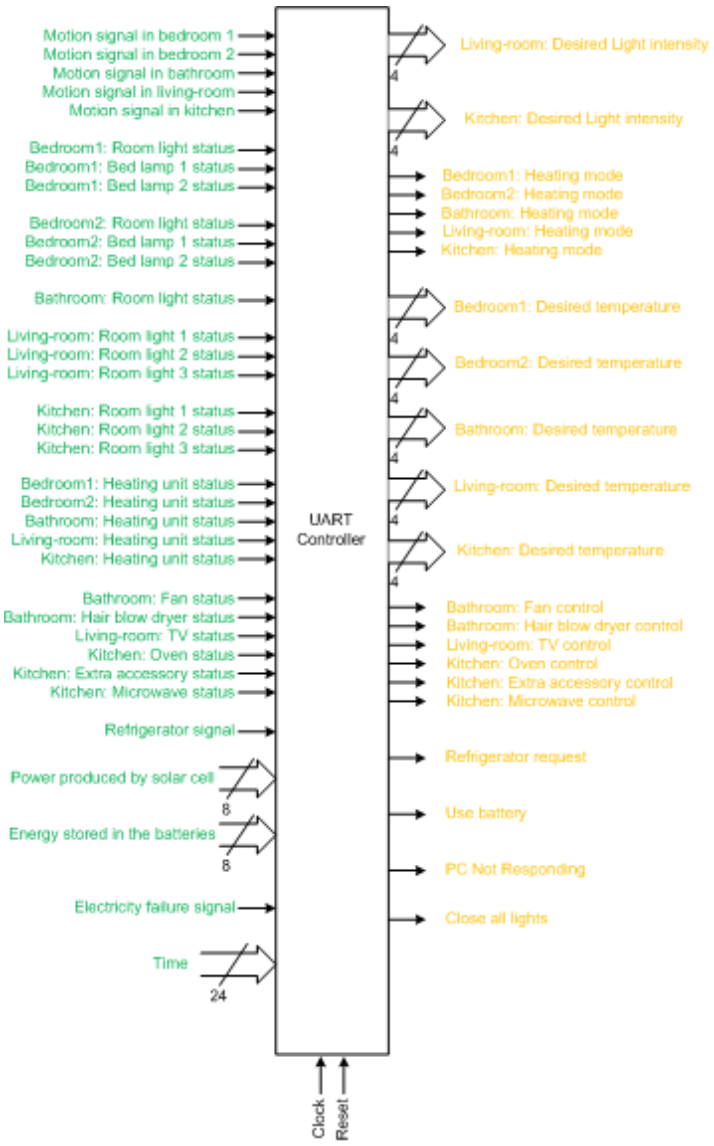


Figure 8.4.0.0: System UART controller inputs and outputs

Simulator

The simulator UART controller communicates with the PC by receiving and sending bytes. Each bit of each byte is mapped to a specific signal. It is similar to the system UART controller except that the number of bytes is not the same and the mapping is different. Most these signals are sent directly to outputs. Similarly, inputs are directly mapped to bytes and sent to the PC. The overall picture of the UART controller for the simulator is shown in figure 8.4.0.1.

Figure 8.4.0.1: Simulator UART controller inputs and outputs

Internal design

The UART controller is made of a main controller and many subcontrollers. The overall design is shown in figure 8.4.0.2.

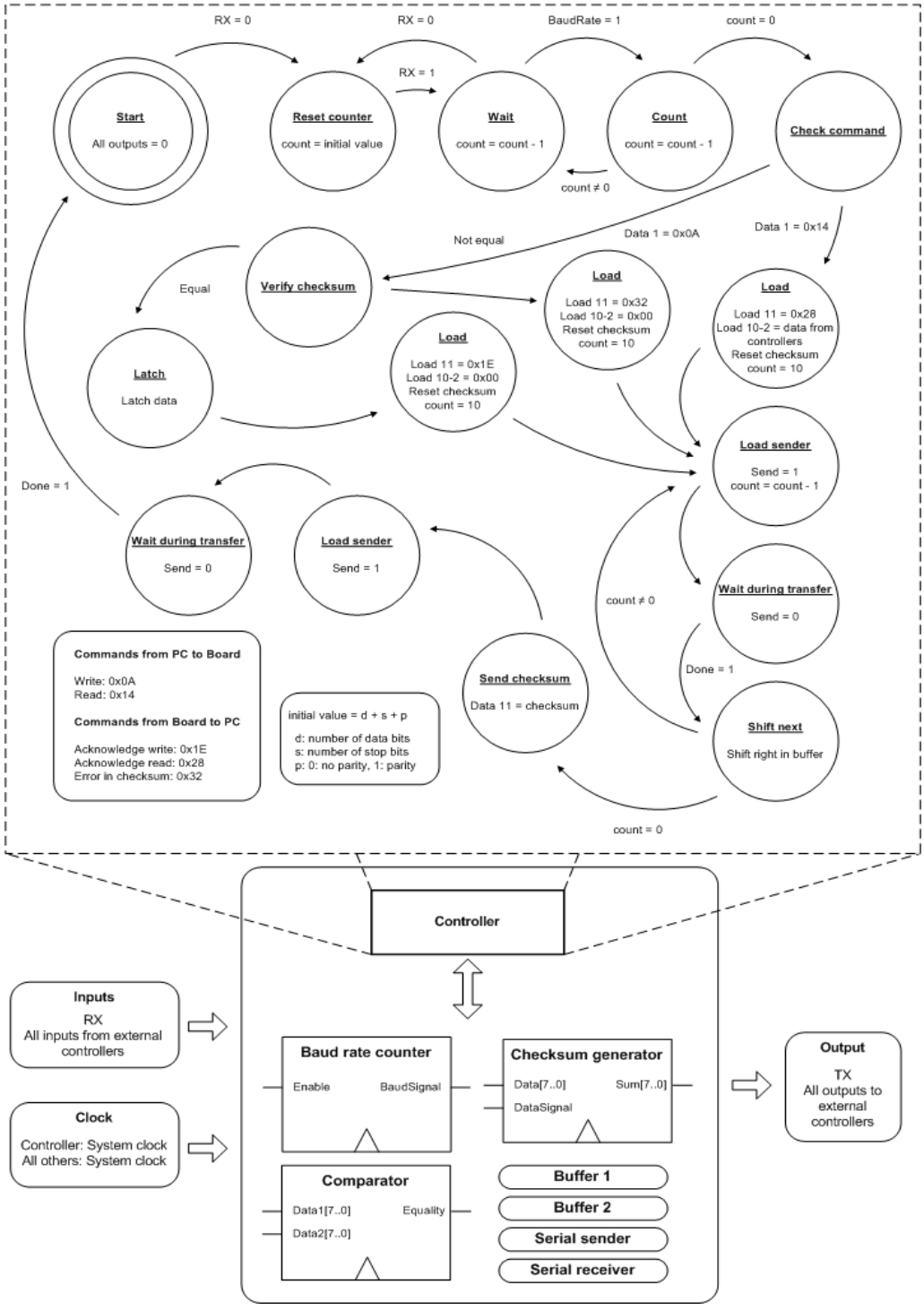


Figure 8.4.0.2: Structure of the UART controller

8.4.1. Baud rate counter

The baud rate counter produces a signal for each bit that is shifted in or out. The design is shown in figure 8.4.1.0.

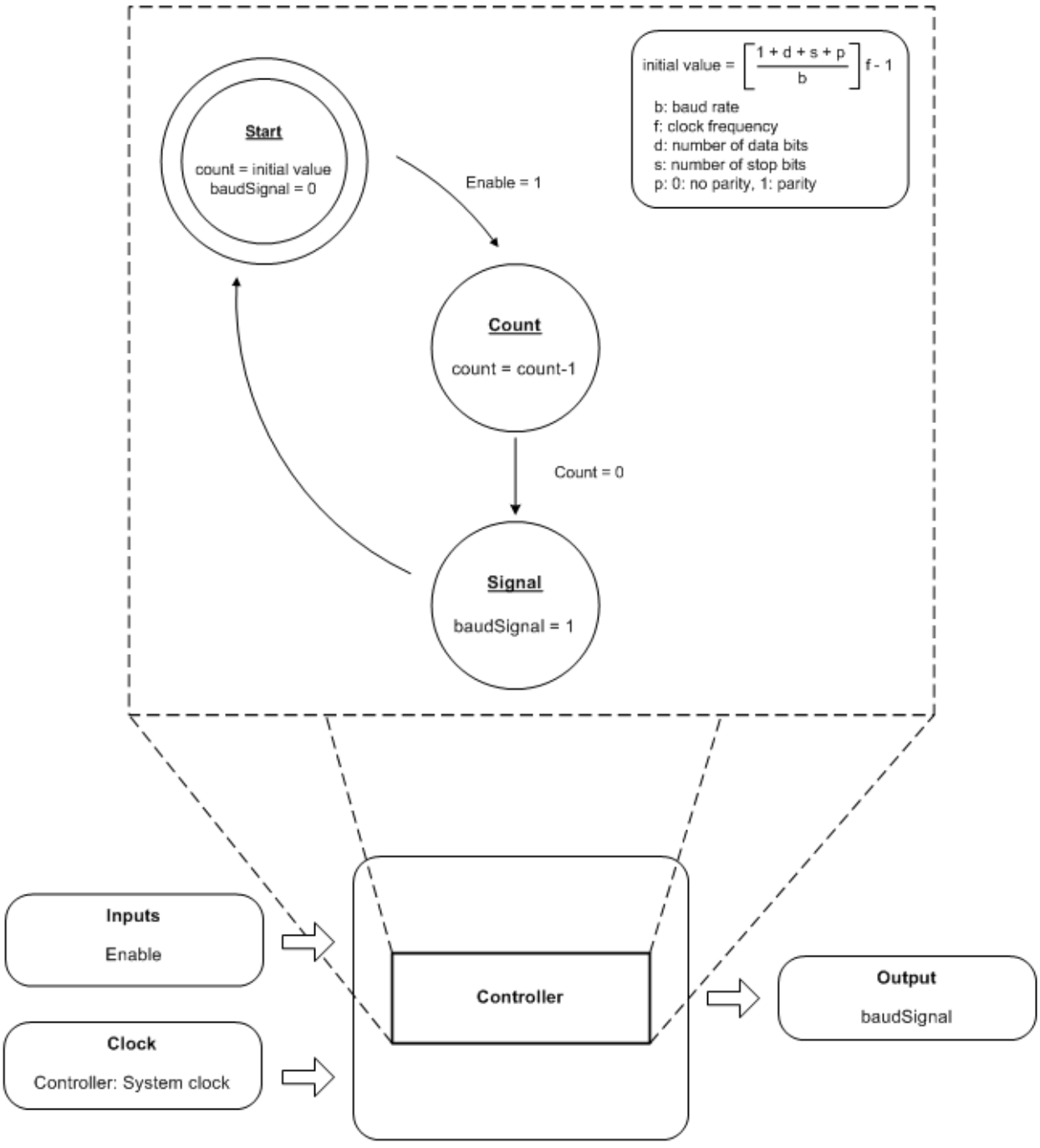


Figure 8.4.1.0: Baud rate controller architecture

8.4.2. Checksum generator

The checksum generator produces a sum from many 8-bit words that have been loaded one after the other. Its design is shown in figure 8.4.2.0.

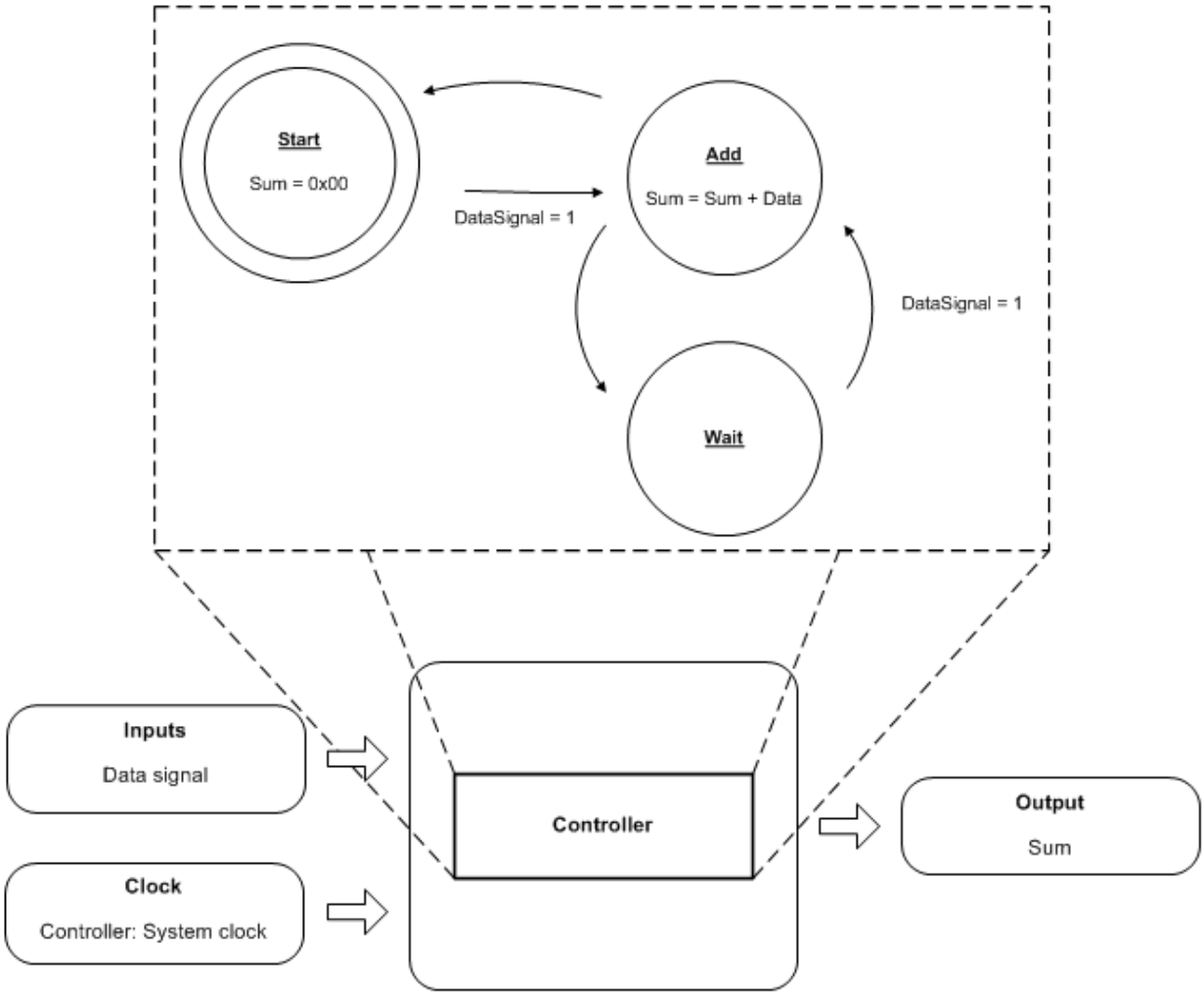


Figure 8.4.2.0: Checksum generator architecture

8.4.3. Serial receiver

The serial receiver receives 1 byte at a time. It is synchronized with the baud rate generator in order to shift in values at the appropriate time. Its design is shown in figures 8.4.3.0 and 8.4.3.1.

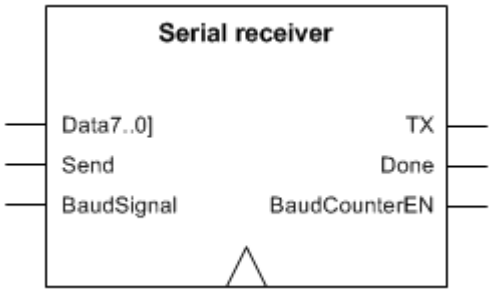


Figure 8.4.3.0: Inputs and outputs of the serial receiver

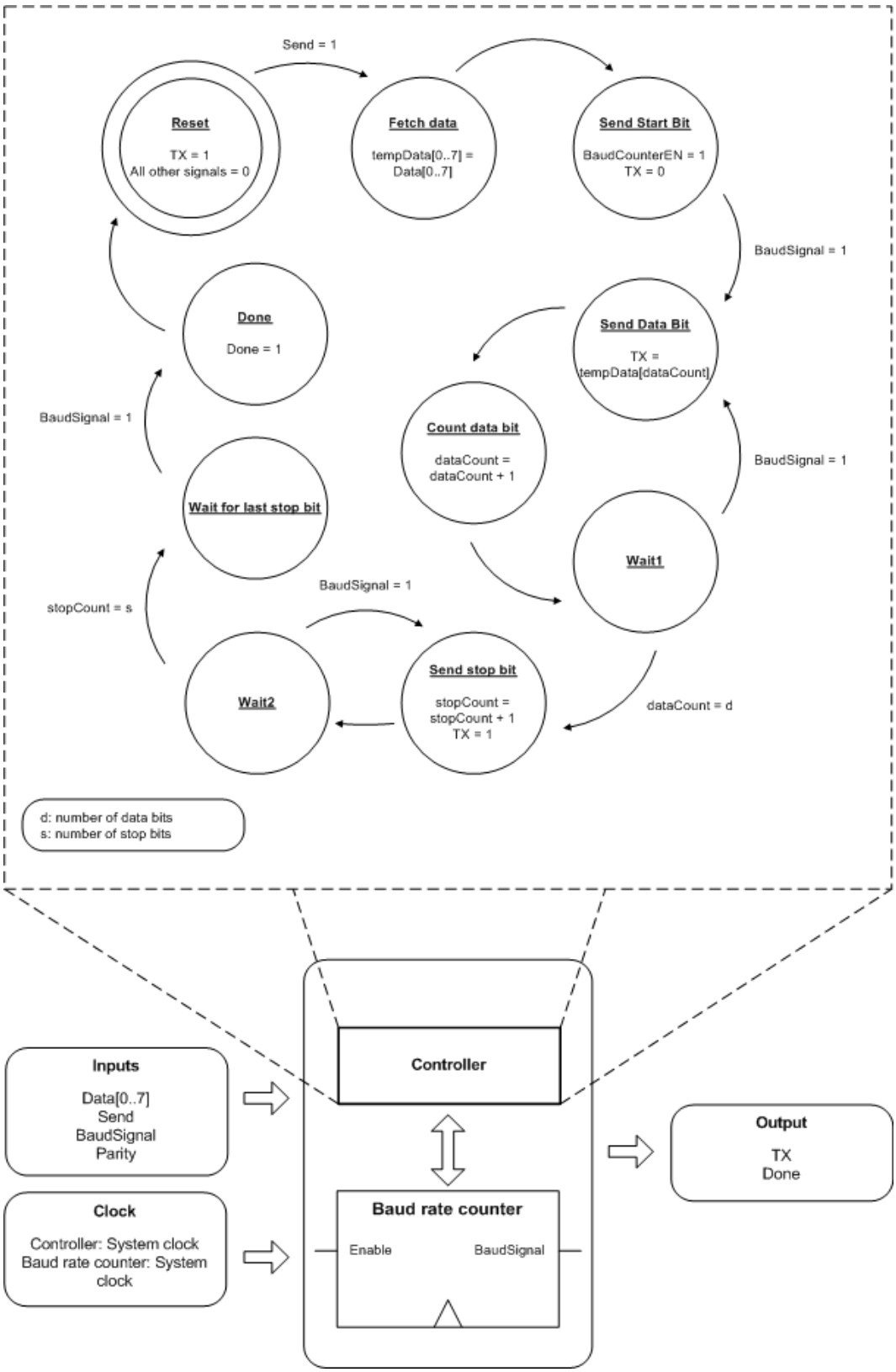


Figure 8.4.3.1: Serial sender architecture

8.4.4. Serial sender

The serial sender sends 1 byte at a time. It is synchronized with the baud rate generator in order to shift out values at the appropriate time. Its design is shown in figures 8.4.4.0 and 8.4.4.1.

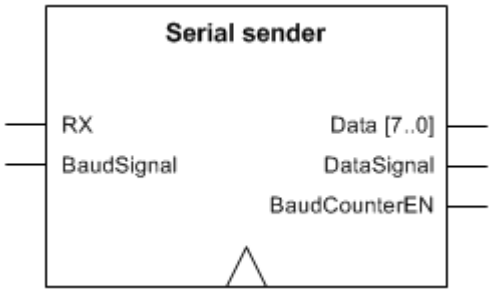


Figure 8.4.4.0: Inputs and outputs of the serial sender

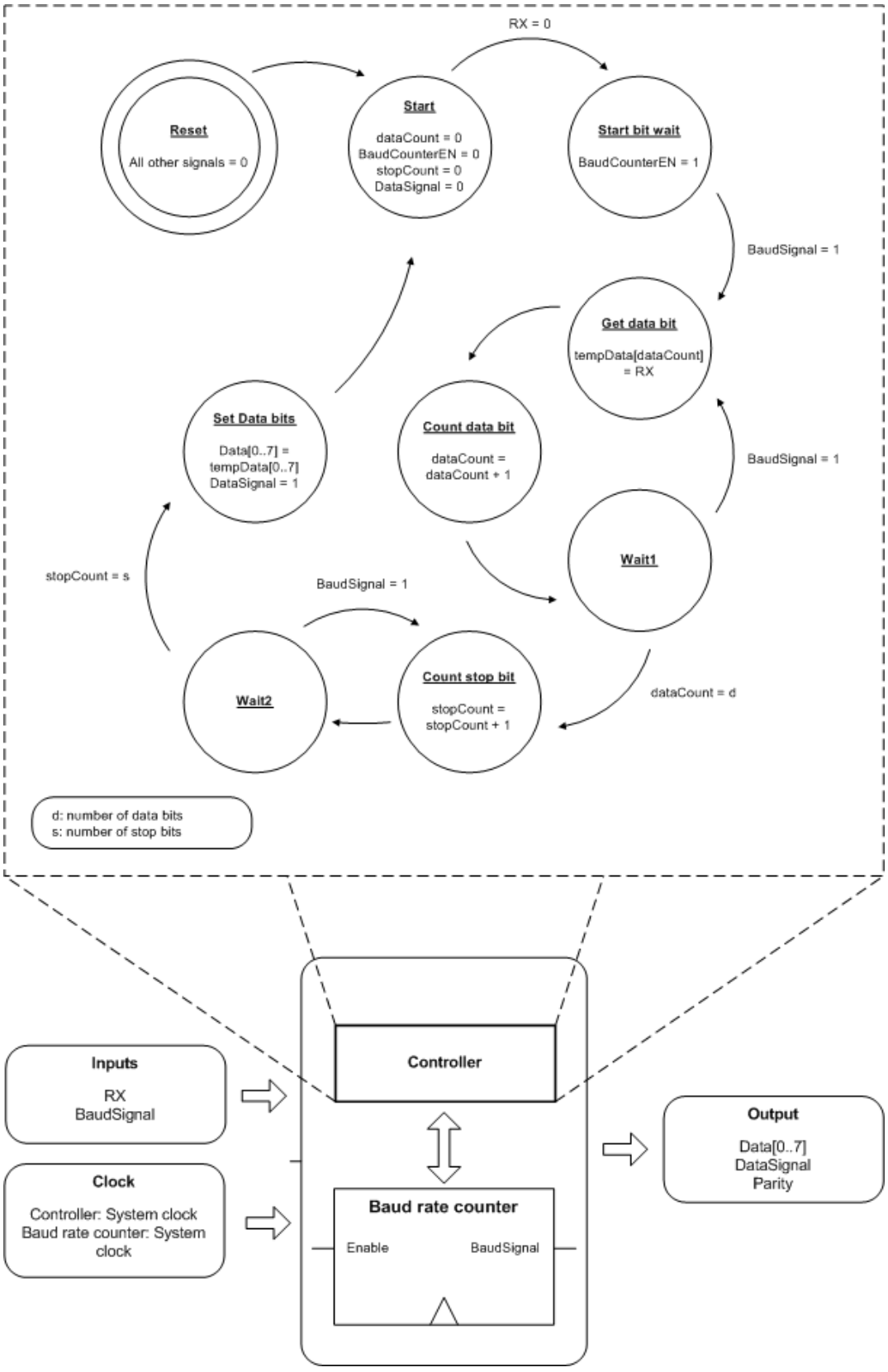
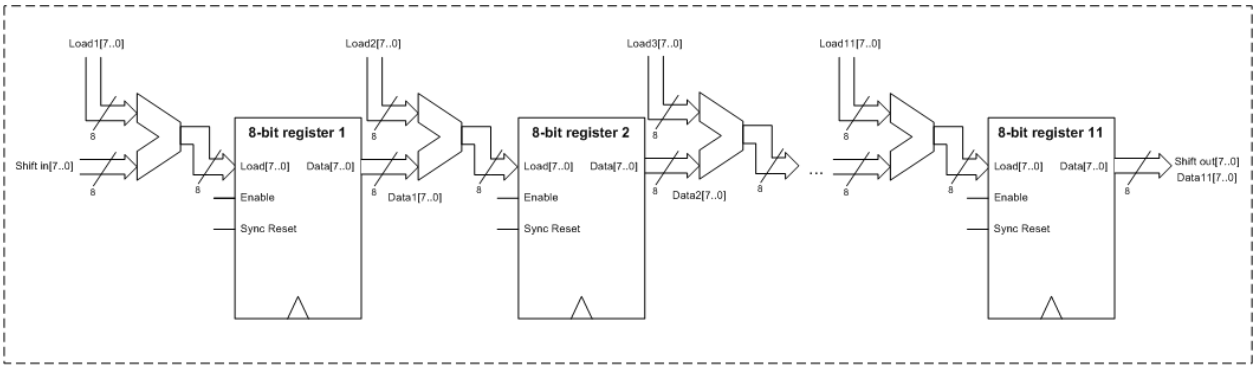


Figure 8.4.4.1: Serial sender architecture

8.4.5. Buffer

Two buffers are used in this design. One is used to shift in and out the values to RX and TX. The other is used to maintain fixed values when shifting occurs and then latch the shifted values when reception is over. The design of the buffer is showed in figure 8.4.5.0.



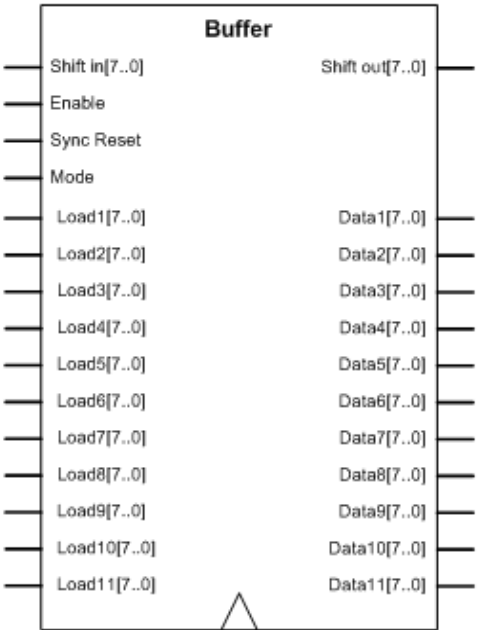


Figure 8.4.5.0: Buffer architecture

8.4.6. PC Failure detector

The PC Failure detector flags a signal when no data has been received from the PC for a given amount of time. When data is received, the flag is unset. The design is shown in figure 8.4.6.0.

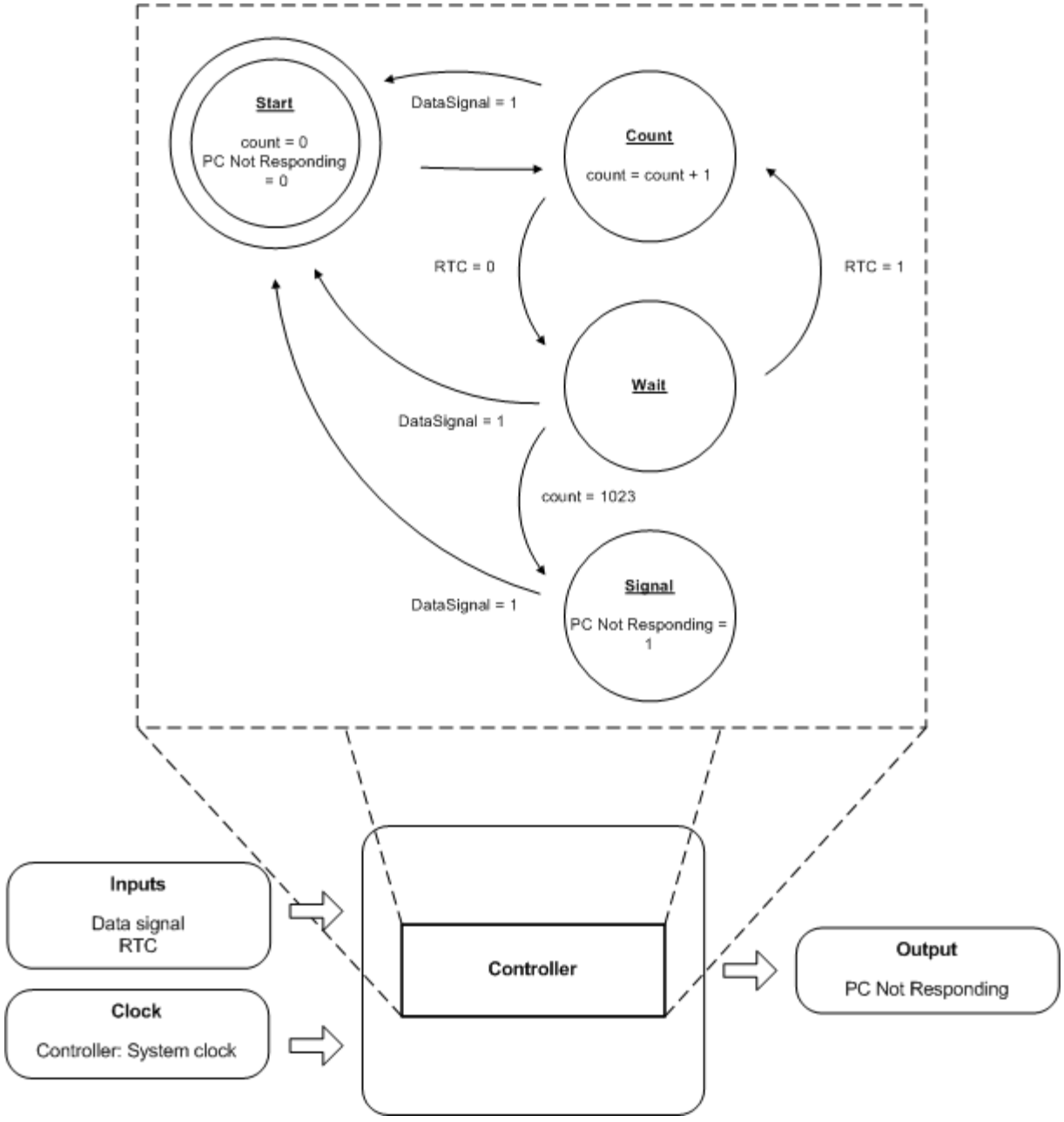


Figure 8.4.6.0: PC Failure detector architecture