

Real Time System Architectures

HOMEWORK 1

FURKAN ÖZEV 161044036

OBJECTIVE:

Keep gas temperature and pressure constant.

TASKS:

TASK1 : PRESSURE CONTROL

- ➔ The function of this task is to manage the DAC value given to the Pump according to the pressure value read from the ADC port. The DAC value ranges from 1 to 100.
- ➔ Since this task will run 100 times a second, its period is 10 ms. So, it is periodic task.
- ➔ The ADC Pressure Port is periodically listened to, and the new DAC value is set according to the current pressure value.
- ➔ It is parallel task.

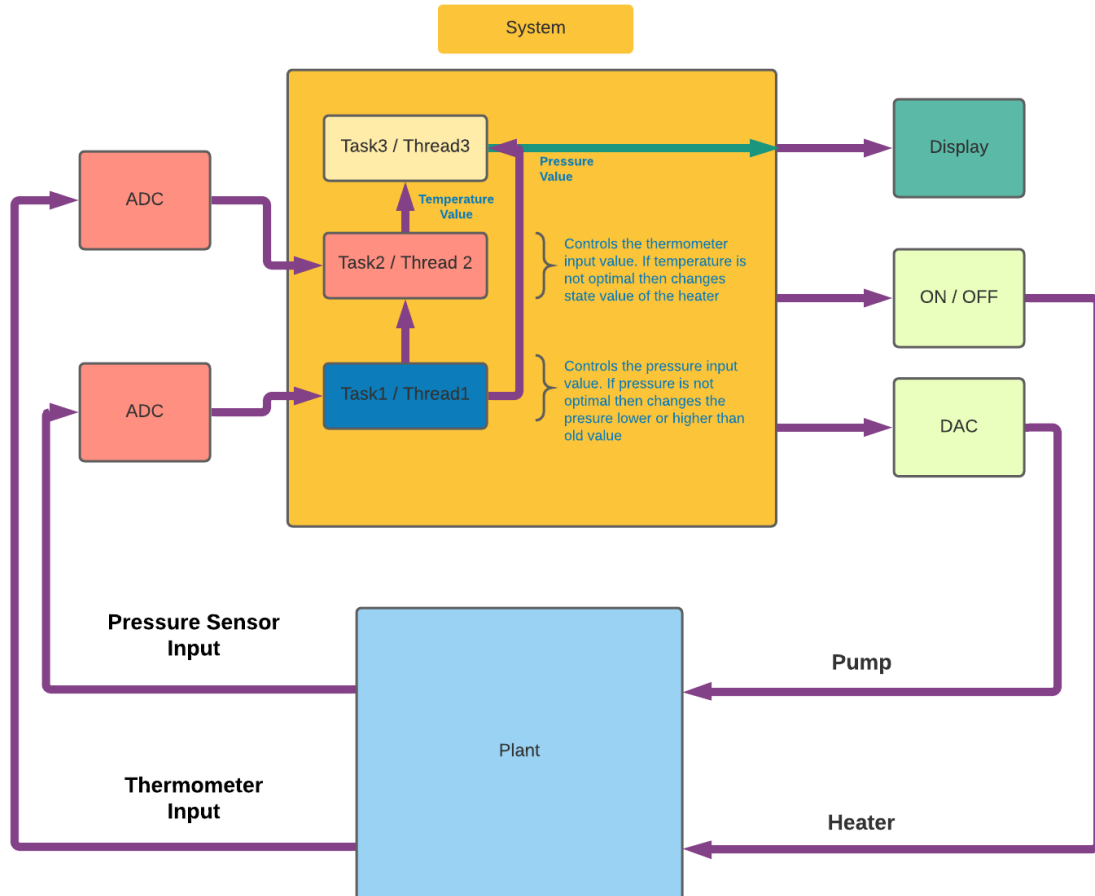
TASK2 : TEMPERATURE CONTROL

- ➔ The function of this task is to manage the Heater button as On / Off according to the thermometer value read from the ADC port.
- ➔ Since this task will run 10 times a second, its period is 100 ms. So, it is periodic task.
- ➔ The ADC Thermometer Port is periodically listened to, and the new Heater button status is set according to the current temperature value.
- ➔ It is parallel task.

TASK3 : DISPLAY

- This task function periodically prints the globally defined heater and pump values to the display.
- Since this task will run 10 times a second, its period is 100 ms. So, it is periodic task.
- It is parallel task.

BLOCK DIAGRAM:



CODES:

```
1  #include <iostream>
2  #include <thread>
3  #include <mutex>
4
5  #define PRESSURE_PIN ?;
6  #define THERMOMETER_PIN ?;
7
8  using namespace std;
9
10 class Controller{
11
12     private:
13         mutex pressureMutex;
14         mutex temperatureMutex;
15         int currentPressure;
16         int currentTemperature;
17
18         void adc_trigger(int port);
19         void read_adc(int port,int &value);
20         void write_dac(int value);
21         void write_switch(bool value);
22         bool newTemperature(int temperature);
23         int newPressure(int pressure);
24         void currentTime();
25
26     public:
27         Controller();
28         void pressureTask();
29         void temperatureTask();
30         void displayTask();
31 };
32
33 void Controller::adc_trigger(int port) { /* It was considered applied */ }
34 void Controller::read_adc(int port,int &value){ /* It was considered applied */ }
35 void Controller::write_dac(int value){ /* It was considered applied */ }
36 void Controller::write_switch(bool value){ /* It was considered applied */ }
37
```

```
38 // Controls and returns the new pressure value
39 int Controller::newPressure(int pressure){
40     int new_pressure;
41
42     if(pressure >= upper_bound){
43         new_pressure = pressure-(pressure-upper_bound);
44     }
45     else if(pressure <= lower_bound){
46         new_pressure = pressure+(lower_bound-pressure);
47     }
48     return new_pressure;
49 }
50
51 // Controls and returns the new heater state
52 bool Controller::newTemperature(int temperature){
53     bool heaterStatus;
54     if(temperature >= upper_bound)
55         heaterStatus = false;
56
57     else if(temperature <= lower_bound)
58         heaterStatus = true;
59
60     return heaterStatus;
61 }
62
63 // Gets the current system time
64 void Controller::currentTime(){ /* It was considered applied */ }
65
66 // Contrustor of the Controller class
67 Controller::Controller(){ /* Intentionally empty */ }
```

```

69 // Task1: Pressure Control
70 void Controller::pressureTask(){
71     int end_time, pressure, start_time;
72
73     for(;;){
74         start_time = this.currentTime();
75
76         // Trigger ADC to take pressure value
77         this.adc_trigger(PRESSURE_PIN);
78
79         // Read the current pressure value
80         this.read_adc(PRESSURE_PIN, pressure);
81
82         // Locks the mutex
83         this.pressureMutex.lock();
84
85         // Control the pressure value
86         this.currentPressure = this.newPressure(pressure);
87
88         // Unlocks the mutex
89         this.pressureMutex.unlock();
90
91         // Write this value to DAC
92         this.write_dac(pressure);
93
94         end_time = this.currentTime();
95
96         // Sleep to wait ADC trigger (100 Hz = 10 ms)
97         sleep(10 - (end_time - start_time));
98     }
99 }
100

```

```

101 // Task2: Temperature Control
102 void Controller::temperatureTask(){
103     int end_time, temperature, start_time;
104     bool state;
105
106     for(;;){
107         start_time = this.currentTime();
108
109         // Trigger ADC to take temperature value
110         this.adc_trigger(TEMPERATURE_PIN);
111
112         // Read the current temperature value
113         this.read_adc(TEMPERATURE_PIN, temperature);
114
115         // Locks the mutex
116         this.temperatureMutex.lock();
117
118         // Control the temperature value
119         this.currentTemperature = temperature;
120         state = this.newTemperature(temperature);
121
122         // Unlocks the mutex
123         this.temperatureMutex.unlock();
124
125         // Write this value to switch
126         this.write_switch(state);
127
128         end_time = this.currentTime();
129
130         // Sleep to wait ADC trigger (1000 Hz = 100 ms)
131         sleep(100 - (end_time - start_time));
132     }
133 }
134

```

```

135 // Task3: Display
136 void Controller::displayTask(){
137     int end_time, start_time;
138
139     for(;;){
140         start_time = this->currentTime();
141
142         // Locks the mutex
143         this->temperatureMutex.lock();
144
145         // Prints the temperature value
146         cout<<this->currentTemperature<<endl;
147
148         // Unlocks the mutex
149         this->temperatureMutex.unlock();
150
151         // Locks the mutex
152         this->pressureMutex.lock();
153
154         // Prints the pressure value
155         cout<<this->currentPressure<<endl;
156
157         // Unlocks the mutex
158         this->pressureMutex.unlock();
159
160         end_time = this->currentTime();
161
162         // Sleep to wait ADC trigger (100 Hz = 10 ms)
163         sleep(10 - (end_time - start_time));
164     }
165 }
166

```

```

167 int main (){
168     Controller obj;
169
170     thread th1 (obj.pressureTask);
171     thread th2 (obj.temperatureTask);
172     thread th3 (obj.displayTask);
173
174     th1.join();
175     th2.join();
176     th3.join();
177
178     return 0;
179 }

```