[IBM/Nackademin]

[PUMP DOCUMENTATION]

Nackademin -22 Internship

IBM

**[EMBEDDED]**

[Foreword 2](#_Toc100052722)

[Arduino code: 2](#_Toc100052723)

[Functions: 2](#_Toc100052724)

[buttonXIsTriggered(); 2](#_Toc100052725)

[debounce(); 2](#_Toc100052726)

[pumpcontroller(); 2](#_Toc100052727)

[pumpSerial(); 2](#_Toc100052728)

[commandHandler(char comdata); 2](#_Toc100052729)

[2](#_Toc100052730)

[void setPot(String potVal); 3](#_Toc100052731)

[send\_json(); 3](#_Toc100052732)

[tempReading(); 3](#_Toc100052733)

[runTime(); 3](#_Toc100052734)

[void safetyValveOFF(); 3](#_Toc100052735)

[void safetyValveON(); 3](#_Toc100052736)

[void pumpOFF(); 3](#_Toc100052737)

[void pumpON(); 3](#_Toc100052738)

[Classes: 4](#_Toc100052739)

[SensorReadings: 4](#_Toc100052740)

### Foreword

# Arduino code:

### Functions:

#### buttonXIsTriggered();

when a button is pushed, it sets the triggered state to true,

as an extra check before the debounce() function.

#### debounce();

check if the button is actually pressed with the buttonAisTriggered()

function, den it checks if it was x millis long ago. If so, it calls the function.

#### pumpcontroller();

checks if the pump is on or off, and sets it to the opposite.

#### pumpSerial();

Reads incomming data as String, converts It to char and counts amount of characters.

If the input is more than one character it’s a potentiometer value.

And send the incomming data to the setPot(cmd); function.

Else it will send it to the command handler, *commandHandler(buf);*

#### commandHandler(char comdata);

The incomming char passed from pumpSerial(); runs through a switch case, to

trigger the right function.

Eg.

#### 

void commandHandler(int comdata) {

switch (comdata) {

case 'A': // A Starts pump

pumpcontroller('A'); //Pumpcontroller in pumpOnOff.h

break;

case 'B'://B stops pump

pumpcontroller('B');//Pumpcontroller in pumpOnOff.h

break;

}

}

#### void setPot(String potVal);

Gets string passed from pumpSerial();

Checks if string starts with any of the 3 letters (X,Y,Z) that represent potentiometers 1,2&3.

Then takes the value that comes after the letter. Eg: X166 sets potentiometer 1 value to 166.

if (potVal.startsWith("X")) {

int input = potVal.substring(3).toInt();

if (input > 1 && input < 255) {

pump\_speed = input;

}

#### send\_json();

In json.h. Takes sensor readings and pump id, and turns it into a nice json object. And Serial.prints it.

#### tempReading();

Reads temperature and stores it in temperature variable.

Checks if old is different from new value. If true, call send\_json();

#### runTime();

To determine how long the arduino has been running.

Breaks down millis() into a seconds counter which is NOT affected by rollovers .

Then hours/ minutes/seconds formatted as: "%02d:%02d:%02d", saved to variable

char runTimeBuffer[21]; in declare.h

A picture containing text

Description automatically generated

Internal time counting using the on-board oscillator/or ceramic resonator as a clocking source may be wrong up to 0.8%.

So counting 1000 seconds with internal timing might be something between 992 and 1008 seconds, actually.

#### void safetyValveOFF();

sets valve\_relay to low.

Sets safety\_valve\_state to false.

#### void safetyValveON();

sets valve\_relay to high.

Sets safety\_valve\_state to true.

#### void pumpOFF();

sets pump\_relay to low.

Calls safetyValveOFF();

Sets pump\_state to false.

Sets buttonA\_ledRing to low.

#### void pumpON();

sets pump\_relay to high.

Calls safetyValveON();

Sets pump\_state to true.

Sets buttonA\_ledRing to high.

#### debounce();

if the button trigger(trigger\_X) is true, it sets the corresponding state to the opposite of what it was before.

The checkHardwareStateChange(); checks the state later and acts accordingly. (true == thing turns on, and the other way around).

We need to do this instead of calling the ON or OFF function directly, because the idlestate(); has to go to a complete stop, otherwise the ledrings on the button will shut down themselves right after they light up.

*void* debounce() {  
 *if* (trigger\_A) {  
  
 *if* ((millis() - buttonAStartMillis) > debounceDelay) {  
 *//if pumpstate is true, set to false and vice versa.  
 if* (pump\_state ? pump\_state = *false* : pump\_state = *true*);}  
 buttonAStartMillis = millis();  
 trigger\_A = *false*;  
 }

### Classes:

#### SensorReadings:

##### .h file:

*class* Sensor{  
*private*:  
 *int* flowSensor;  
 *int* lastFlowSensor;  
*public*:  
 *//SETTER  
 float* setFlowSensorValue(*float* s);  
 *float* setLastFlowSensorValue(*float* s);  
 *//GETTER  
 float* getFlowSensorValue() *const*;  
};

##### .cpp:

setFlowensorValue(int s); sets corresponding float value to each object.

setLastFlowensorValue(int s); sets corresponding value from flowSensor.

getFlowensorValue(int s); gets corresponding value from each object.

*float* Sensor::setFlowSensorValue(*float* s){  
 *this* -> flowSensor = s;  
}  
  
*float* Sensor::getFlowSensorValue() *const* {  
 *return this* -> flowSensor;  
}  
  
*float* Sensor::setLastFlowSensorValue(*float* s) {  
 *this* -> lastFlowSensor = s;  
}

#### Convert:

*The sensor:*

*The flow meter, for every liter of liquid passing through it per minute, it outputs about 4.5 pulses. This is due to the changing magnetic field caused by the magnet attached to the rotor shaft. Each pulse is approximately 2.25 milliliters.*

##### .h file:

*class* Convert{  
*private*:  
 *int* X;  
 *int* Y;  
 *float* TIME = 0;  
 *float* FREQUENCY = 0;  
 *float* WATER = 0;  
 *float* LS = 0;  
 *float* TOTAL = 0;  
*public*:  
 *float* convertToLitres(*float* inflow);  
  
}

.cpp file:

pulseIn meassures high and low.

Time is first set to equal all high-time and all low-time.

Formula:

**Litres** = Q \* time elapsed (seconds) / 60 (seconds/minute)  
**Litres** = (Frequency (Pulses/second) / 7.5) \* time elapsed (seconds) / 60  
**Litres** = Pulses / (7.5 \* 60)

Returns a float total that’s litres per minute.

*float* Convert::convertToLitres(*float* inflow){  
 *this* ->X = pulseIn(inflow, HIGH);  
 *this* ->Y = pulseIn(inflow, LOW);  
 *this*->TIME = X + Y;  
 *this*->FREQUENCY = 1000000/*this*->TIME;  
 *this*->WATER = *this*->FREQUENCY/7.5;  
 *this*->LS = *this*->WATER/60;  
 *this*->TOTAL = *this*->TOTAL + *this*->LS;  
 *return this*-> TOTAL;  
}