Home Automation - Electricity Monitoring Tutorial 2

Extend light sensor notification system to allow real time monitoring of electricity usage and get updates of current prepaid electricity level on request.

Table of Contents

1. Introduction

Welcome to part two of this tutorial series. In this tutorial we will extend our previous monitoring system to no also keep track of our real time usage of electricity. The aim of this project is to be able to get an on demand reading of where our electricity usage is at. It's all good and well to be notified when your electricity is running low, but as you reach the end of the month you might find yourself in a situation where you need to know how much electricity you have left in order to make an informed choice of how your budget will play out. Thanks to many of the modern prepaid electricity boxes having a light indicating usage by means of blinking an X amount of times per kWh, it is now possible to calculate how much electricity is being used and convert that into units. Once we know the unit amount of our usage, we can start doing all sort of things.

The purpose of this tutorial will be to implement a Light Dependant Resistor (LDR) to measure the amount of blinks happening and then in the background convert those blinks into measurable usage. For this to work however certain variables needs to be taken into account and an understanding gained on how your electricity consumption is measured. A voltage divider is made with the LDR and a 10K resistor. The voltage output from this divider is passed to an analog input on the ArduPi to be measured. When the LED is off, the light detector will be in darkness and have a resistance of 100's of KOhms so the analog input will see a voltage near to 0V. When the LED is on, the light detector (which is positioned very close to the LED) will have a resistance of well under 1 KOhm, so the analog input will see a voltage near to 5V.

Your monthly electricity consumption is calculated based on kWh – Kilowatt Hour. One Kilowatt is equal to 1000 watts and the equivalent of 1 unit of electricity displayed on your box. A standard globe used in a house is 100 Watts. If that globe is switched on for one hour it will consume 0.1 kWh (or units) of power (100 Watts / 1000watts per hour). If that same globe were to burn for 24 Hours it would therefore use 2.4 kWh or 2.4 Units of electricity.

Your prepaid electricity meter is able to count the flow of electricity used in kWh and then converts the usage as seen above into units of electricity. The meter balance therefore decreases as you use electricity. The red/green LED on the meter flashes as electricity is used: the faster it flashes, the more units are being used.

It's also important to note that different electricity boxes flashes at different rates per kWh used. For the purposes of this tutorial I will be making use of the rate that the box in my house uses, which is set to 400 flashes per kWh.

Below is a picture of the meter in my house in which you can see the indication of how many flashes it has per kWh as well as shows where my flashing light is.

Seeing as this is a follow up on our previous tutorial, you will require a working version of the previously created light sensor notification system. This includes all software and hardware. We will simply add to the existing code base as well a second sensor to our breadboard and Ardupi.

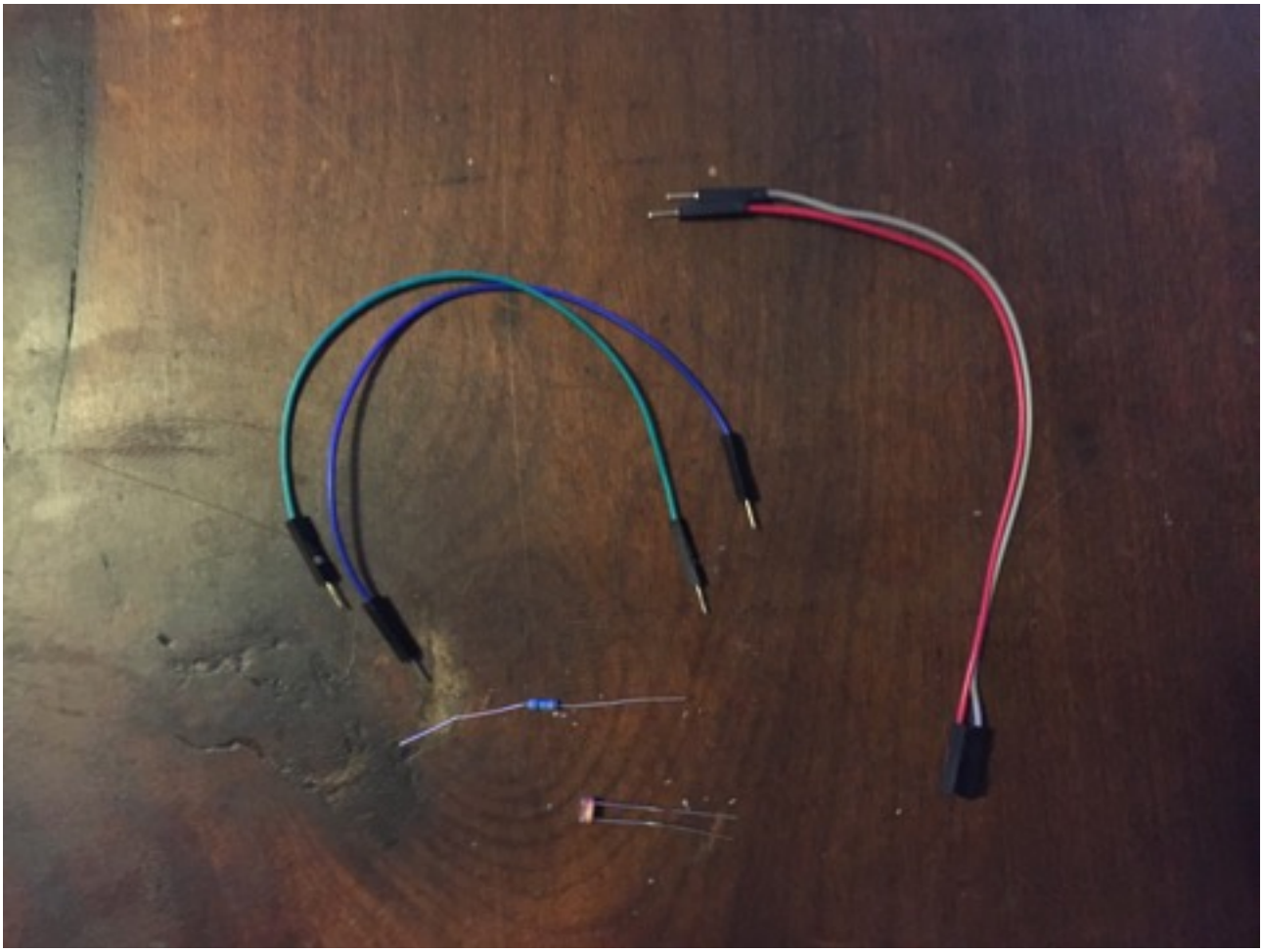
With all of that our of the way, let's jump straight into things and get going.

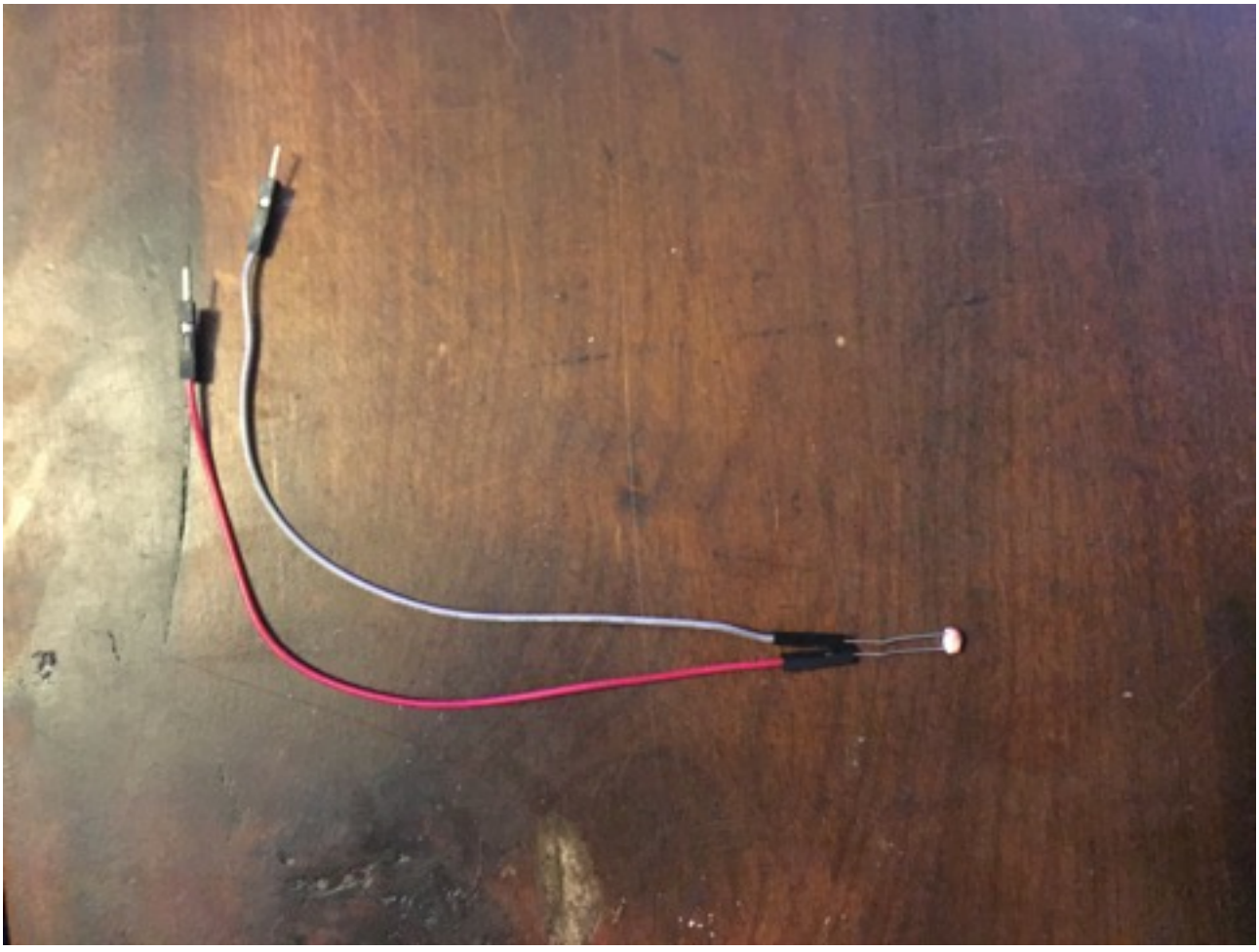
---

2. Hardware requirements

As mentioned you will require all the components assembled and working from the previous tutorial. Over and above that you will also require the following:

- Additional 11k Light Dependant Resistor (LDR)
- 10k ohm resistor
- Wires (At least 4 will be required.)
  - 2x Male to Female (I used red and grey)
  - 2x Male to Male (I used blue and green)
- heatshrink tube

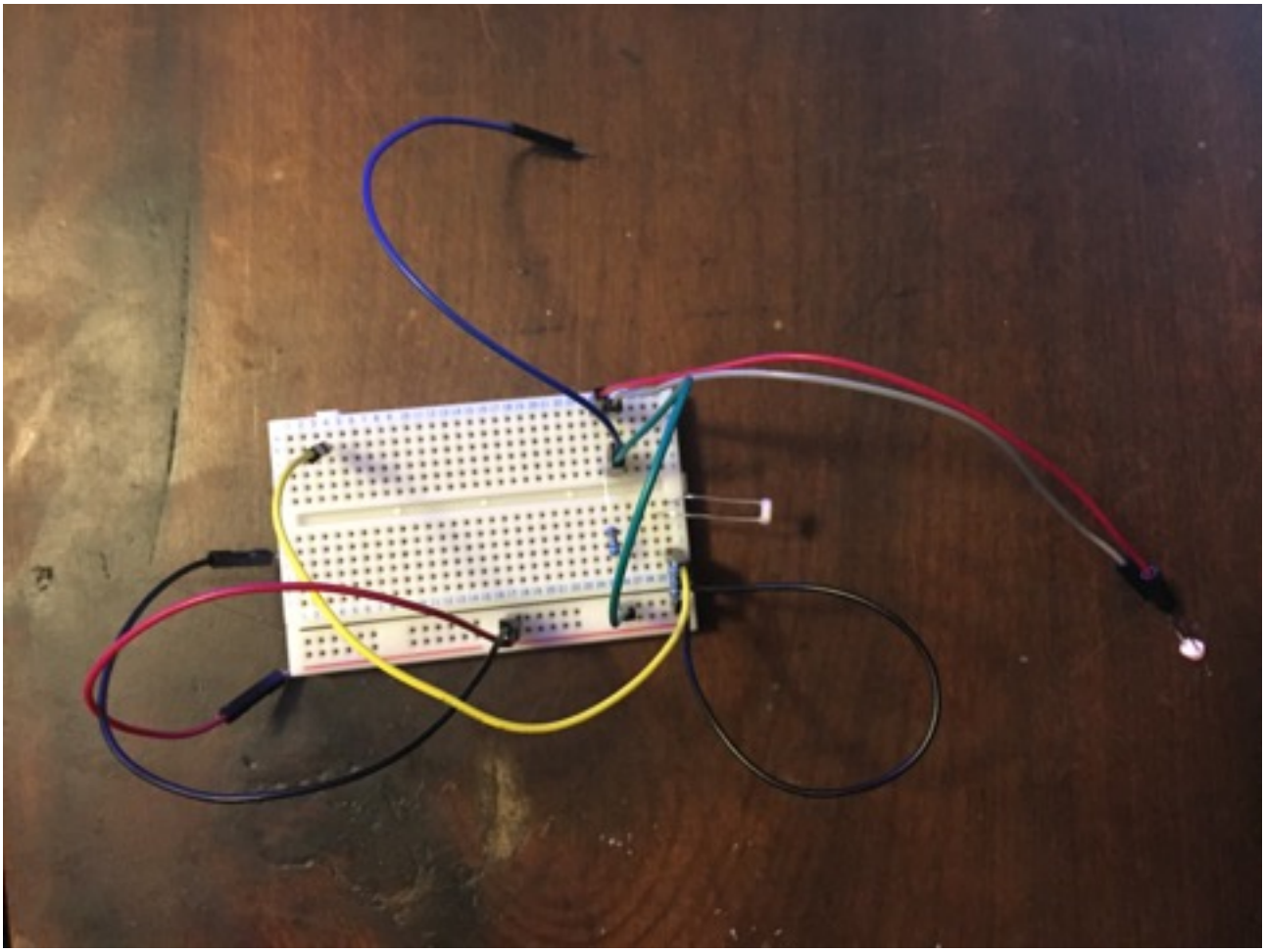




3. Hardware setup

As with the previous tutorial we will begin by adding the components into the breadboard.

1. Begin by adding your 10k ohm resistor into pin 25 of the positive row on the bus strip and the other into pin F25 on the terminal strips.
2. Take one of your male to male wires (I used my green one), and add one end into pin 26 of the negative row on the bus strip and the other end into pin F26 on the terminal strips.
3. Next take the remaining male to male wire and insert one end into pin E26 of the terminal strip and the other end into our ArduPi's number 4 analogue input
4. The two male to female wires will be used in conjunction with your second LDR in order to allow for variable placement of your LDR in connection with your flashing electricity meter light. Go ahead and connect the female parts of each of the wires to a leg of the LDR. Once done Connect the male sides of the wires, now connected to the LDR to the breadboard on pins A25 and A26.



And that's it for the hardware setup. We can now focus on getting the software setup and running before installing our project and having peace of mind.

4. Software setup

Time to update our existing code to include the new sections. This is the part where it can get a bit tricky. To make things easier I added the entire contents of the file below and then simply added the new snippets into the existing code base with red. Remember to insert your own Clickatell account credentials into the appropriate section.

Start of by opening our Final.cpp file in the arduPi directory

pi@raspberrypi ~/arduPi $ *vi Final.cpp*

Once opened you will se the below. Continue to then add in to various parts in red and then saving your file.

```
//Include ArduPi library
#include "arduPi.h"

//Include the Math library
#include <math.h>

//Clickatell Message Code
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "clickatell/src/clickatell_sms/clickatell_debug.h"
#include "clickatell/src/clickatell_sms/clickatell_string.h"
#include "clickatell/src/clickatell_sms/clickatell_sms.h"

/* ------------------------------------------------------------------------ *
 * Input configuration values                                               *
 * NOTE: Please modify these values and replace them with your own credentials.  *
 * ------------------------------------------------------------------------ */

// insert your HTTP API credentials here
#define CFG_HTTP_USERNAME          "myusernamehere" // insert your Clickatell account username here
#define CFG_HTTP_PASSWORD          "mypasswordhere" // insert your Clickatell account password here
#define CFG_HTTP_APIID             "myapiidhere"       // insert your Clickatell HTTP API ID here

// insert your destination addresses here
//For this project you only need to insert
#define CFG_MSISDN1          "" // insert your first desired destination mobile number here

// insert your SMS message sText here
#define CFG_MSG_TEXT        "WARNING: Your prepaid electricity is running low."
#define CFG_BAL_TEXT        "Your current balance is: "

// timeout values - these can be modified or left as is
#define CFG_APICALL_TIMEOUT        5 // Config: Maximum time in seconds (long value) for API call to take
#define CFG_APICALL_CONNECT_TIMEOUT 2 // Config: maximum time in seconds (long value) that API call takes to connect to Clickatell server

/* ------------------------------------------------------------------------ *
```

```
 * Fixed Macros/Types                                                    *
 * ---------------------------------------------------------------------- */

// common print functions
#define PRINT_MAIN_TEST_SEPARATOR
{ click_debug_print("\n============================================================
====================================\n"); }
#define PRINT_SUB_TEST_SEPARATOR    { click_debug_print("\n\n"); }

static void run_common_setup(eClickApi eApiType);
static void run_common_api_call(eClickApi eApiType, ClickSmsHandle *oClickSms);

#define KO 1
#define triggeredMessage 0
#define coolDown 0
#define flashesPerUnit 400


/*******************************************************
 *  IF YOUR ARDUINO CODE HAS OTHER FUNCTIONS APART FROM  *
 *  setup() AND loop() YOU MUST DECLARE THEM HERE        *
 * *****************************************************/

/*
 * Function:  run_common_setup
 * Info:      Runs series of API calls.
 * Inputs:    API type
 * Return:    void
 */
static void run_common_setup(eClickApi eApiType, type, msg = null)
{
    ClickSmsHandle *oClickSms = NULL;

    PRINT_MAIN_TEST_SEPARATOR

    switch (eApiType) {
        case CLICK_API_HTTP:
            click_debug_print("Executing HTTP API Tests with Username+Password as authentication
method\n\n");

            ClickSmsString *sHttpUser = click_string_create(CFG_HTTP_USERNAME);
            ClickSmsString *sHttpPassword = click_string_create(CFG_HTTP_PASSWORD);
            ClickSmsString *sHttpApiId = click_string_create(CFG_HTTP_APIID);

            oClickSms = clickatell_sms_handle_init(eApiType,
                                        sHttpUser,
                                        sHttpPassword,
                                        NULL,
                                        sHttpApiId,
                                        CFG_APICALL_TIMEOUT,
                                        CFG_APICALL_CONNECT_TIMEOUT);
            if (oClickSms == NULL)
            {
                click_debug_print("ERROR: Clickatell SMS Module Initialization failed\n");
            }
```

```
            if (type == common)
            {
                run_common_api_calls(eApiType, oClickSms);
            }
            else
            {
                run_balance_api_calls(eApiType, oClickSms, msg);
            }
            clickatell_sms_handle_shutdown(oClickSms);

            click_string_destroy(sHttpUser);
            click_string_destroy(sHttpPassword);
            click_string_destroy(sHttpApiId);
            break;

        default:
            click_debug_print("ERROR: Invalid API type selected!\n");
            break;
    }
}

/*
 * Function:  run_common_api_call
 * Info:      Runs common API call.
 *            Ensure function clickatell_sms_handle_init() has been called prior to this
 *            function.
 * Inputs:    eApiType  - API call type
 *            oClickSms - ClickSmsHandle handle returned from clickatell_sms_handle_init() call
 * Return:    void
 */
static void run_common_api_call(eClickApi eApiType, ClickSmsHandle *oClickSms)
{
    int i = 0;
    ClickSmsString *sResponse = NULL;
    ClickSmsString *sMsgText = click_string_create(CFG_MSG_TEXT);

    click_debug_print("[%s: Send SMS]\n\n", (eApiType == CLICK_API_HTTP ? "HTTP" : "REST"));
    ClickMsisdn *aMsisdnsSingle = (ClickMsisdn *)calloc(1, sizeof(ClickMsisdn));
    aMsisdnsSingle->aDests = calloc(1, sizeof(ClickSmsString *));
    aMsisdnsSingle->iNum = 1;       // send to 1 mobile number
    aMsisdnsSingle->aDests[0] = (ClickSmsString *)click_string_create(CFG_MSISDN1);

    ClickSmsString *sMsgIdResponse = clickatell_sms_message_send(oClickSms, sMsgText,
aMsisdnsSingle);
    PRINT_SUB_TEST_SEPARATOR

    click_string_destroy((ClickSmsString *)(aMsisdnsSingle->aDests[0]));
    free(aMsisdnsSingle->aDests);
    free(aMsisdnsSingle);

    // retrieve apiMessageId field from response
    ClickSmsString *sMsgId = NULL;
    if (eApiType == CLICK_API_HTTP) {
        /* A successful response should look like this:   ID: 205e85d0578314037a96175249fc6a2b
         * which means we need to remove the 'ID:' prefix text and space character from the response
         */
```

```
        click_string_trim_prefix(sMsgIdResponse, 4);
        sMsgId = click_string_duplicate(sMsgIdResponse);
    }

    click_string_destroy(sMsgIdResponse);
    click_string_destroy(sMsgId);
    click_string_destroy(sMsgText);
}

/*
 * Function:  run_balance_api_call
 * Info:      Runs balance API call.
 *            Ensure function clickatell_sms_handle_init() has been called prior to this
 *            function.
 * Inputs:    eApiType  - API call type
 *            oClickSms - ClickSmsHandle handle returned from clickatell_sms_handle_init() call
 * Return:    void
 */
static void run_balance_api_call(eClickApi eApiType, ClickSmsHandle *oClickSms, msg)
{
    int i = 0;
    ClickSmsString *sResponse = NULL;
    ClickSmsString *sMsgText = click_string_create(CFG_BAL_TEXT.concat(msg));

    click_debug_print("[%s: Send SMS]\n\n", (eApiType == CLICK_API_HTTP ? "HTTP" : "REST"));
    ClickMsisdn *aMsisdnsSingle = (ClickMsisdn *)calloc(1, sizeof(ClickMsisdn));
    aMsisdnsSingle->aDests = calloc(1, sizeof(ClickSmsString *));
    aMsisdnsSingle->iNum = 1;        // send to 1 mobile number
    aMsisdnsSingle->aDests[0] = (ClickSmsString *)click_string_create(CFG_MSISDN1);

    ClickSmsString *sMsgIdResponse = clickatell_sms_message_send(oClickSms, sMsgText,
aMsisdnsSingle);
    PRINT_SUB_TEST_SEPARATOR

    click_string_destroy((ClickSmsString *)(aMsisdnsSingle->aDests[0]));
    free(aMsisdnsSingle->aDests);
    free(aMsisdnsSingle);

    // retrieve apiMessageId field from response
    ClickSmsString *sMsgId = NULL;
    if (eApiType == CLICK_API_HTTP) {
        /* A successful response should look like this:   ID: 205e85d0578314037a96175249fc6a2b
         * which means we need to remove the 'ID:' prefix text and space character from the response
         */
        click_string_trim_prefix(sMsgIdResponse, 4);
        sMsgId = click_string_duplicate(sMsgIdResponse);
    }

    click_string_destroy(sMsgIdResponse);
    click_string_destroy(sMsgId);
    click_string_destroy(sMsgText);
}



/*************************
```

```
 * YOUR ARDUINO CODE HERE *
 * ************************/

int main (int argc, char *argv[]){

        setup();

        while(1){
                loop();
        }

        return (0);
}

void setup(void) {
}


void loop(int argc, char *argv[]){
  float analogReadingArduino;
  analogReadingArduino = analogRead(5);
  analogReadingArduino2 = analogRead(4);
  currentUnits = 0;
  amountOfFlashes = 0;
  msgReceived = null;
  lastState = 0;

  if(analogReadingArduino < KO){
    if(triggeredMessage == 0 && coolDown == 0){
      // start using Clickatell library
      clickatell_sms_init();

      // run Clickatell HTTP common API calls (with Username+Password as authentication)
      run_common_setup(CLICK_API_HTTP, 'common');

      // finished using Clickatell library
      clickatell_sms_shutdown();

            //Update variables
            triggeredMessage = 1;
            coolDown = 1;
    }
  } else {
    if(triggeredMessage == 1){
      triggeredMessage = 0;
    }

    if (coolDown < 100 && coolDown > 0) {
            coolDown++;
    } else {
            coolDown = 0;
    }
  }

//Check if flashing light is on flash count if off
if(analogReadingArduino > KO){
```

```
    lastState = 0;
    if (amountOfFlashes == 0 && coolDown == 0) {
        amountOfFlashes = 1;
    }
    elseif (amountOfFlashes != 0 && coolDown == 0)
    {
        amountOfFlashes++;
    }

    if (coolDown == 0) {
        coolDown = 1;
    }

    if (coolDown < 10 && coolDown > 0) {
            coolDown++;
    } else {
            coolDown = 0;
    }

}
else
{
    lastState = 1;
}

//Update Unit amount if flashes per unit has been reached. On update reset the flashed to 0
if (amountOfFlashes == flashesPerUnit)
{
    amountOfFlashes = 0;
    currentUnits - -;
}

//Check is a incoming message has been received and act accordingly

if (received == true && (msg == 'balance' ll int(msg))
{
    msgReceived = msg;
}
else
{
    msgReceived = null;
}


//Return balance
if (msgReceived == 'balance')
{
    // start using Clickatell library
    clickatell_sms_init();

    // run Clickatell HTTP common API calls (with Username+Password as authentication)
    run_common_setup(CLICK_API_HTTP, 'balance', currentUnits);

    // finished using Clickatell library
    clickatell_sms_shutdown();
```

```
    msgReceived = null;
}

//Update unit amount
if (msgReceived != 'balance' && msgReceived != null )
{
    currentUnits = msgReceived;
    msgReceived = null;
}

  delay(100);
}
```

Apart from the new additions to the code, a couple of alterations were also made. The delay time for the loop was decreased from running every 3 seconds to running every 0.1seconds. The Clickatell library script was also updated to accommodate for the new message type and msg being sent.

The rest of the code in red runs through a sequence of events from the top as follow

1. Check if flashing light is on flash count if off
2. Update Unit amount if flashes per unit has been reached. On update reset the flashed to 0
3. Check is a incoming message has been received and act accordingly
4. Return balance if requested
5. Update unit amount new unit amount is received.

Once done, you can save your file and then compile it.

pi@raspberrypi ~/arduPi $ *g++ -lrt -lpthread Final.cpp arduPi.o -o Final*

To test your end result run the below.

pi@raspberrypi ~/arduPi $ *sudo ./Final*

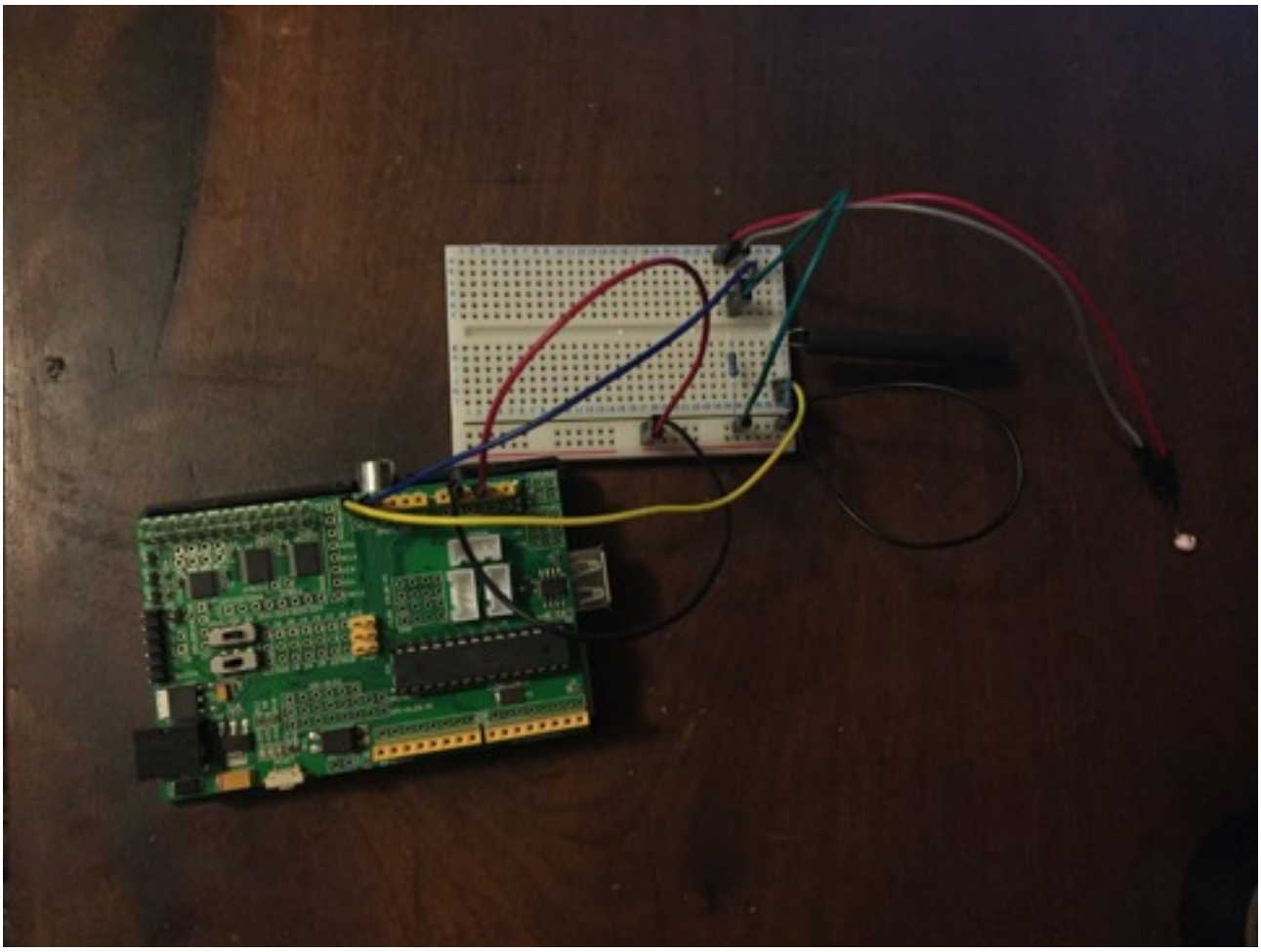
---

5. Installation of the final product.

The last thing to do is install your finished project, or reinstall if you reused the previous one. Due to the previous tutorial already setting up an on start event, you won't need to worry about setting that up again. If you do however experience a problem with your project not booting on start, head over to the previous tutorial and go through the steps again.

When installing your project it's important to remember that your LDRs are very sensitive and therefore require shielding from unwanted light. Make sure your heatshrink is attached properly and that your two sensors are place directly in from of the corresponding LED light and that now external light can effect them. As mentioned during the hardware setup depending on the placement of your flashing light, you will probably need to get a bit creative in order to make sure your sensors are set up correctly. For my meter I used a steel wire to create a little stand for myself to keep my sensor in place, but you are free to make use of what ever you like.

As before, once everything is in place and you are happy, you can go ahead and switch on the raspberry pi.



6. Helpful links

Below you can find some helpful links that can be used in this project.

- Raspberry Pi Foundation : https://www.raspberrypi.org/
- Debian WiFi : https://wiki.debian.org/WiFi
- Clickatell : https://www.clickatell.com