PERSONAL SOLAR-POWERED WATER STORAGE REGULATION

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

In partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Christopher Ryan Land

MEng Field Advisor: Bruce Robert Land

Degree Date: May 2016

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: PERSONAL SOLAR-POWERED WATER STORAGE REGULATION

Author: Christopher Ryan Land (crl233@cornell.edu)

Abstract:

An organic farm uses a 2000-gallon water tank to supply water to crops and animals. Currently, the tank is filled with rainwater, but the rain does not occur consistently enough to keep the tank full. The goal of the project is to automate the filling of the tank using water from a pond 100 meters across a densely vegetated field. The entire system runs from solar power, meaning a microcontroller and water pump must run from the same solar panel. The microcontroller must be instructed to turn on the pump from a wireless control signal from another microcontroller, which is also running on solar panel and is situated at the tank.

Executive Summary

An organic farm in Burdett, NY uses a 2000-gallon water storage tank near the crops and animals for easy access to water. To minimize environmental impact, the tank is currently replenished with rainwater. However, because rain is unpredictable and does not produce enough water to fill the tank, the goal of the project is to keep the tank full by pumping water from a pond 100 meters behind thick vegetation. When the tank is not full, a signal will be sent toward the pond, and the electronics at the pond will react and begin pumping water to re-fill the tank.

The first design decision was to use 433MHz as the carrier frequency for the signal. This frequency works well because the large wavelength is good for penetration of brush and trees. It is also a legal unlicensed frequency in the U.S., provided that it is only used for "intermittent control signals." Fortunately, this project requires very sparse intermittent control signals on the order of < 20 transmissions per minute.

The chosen microcontroller was the Moteino-USB R5 revision. Moteino is an Arduino Uno derivative designed by LowPowerLab. The Moteino works with the standard Arduino IDE and is designed to easily interface with RF transceiver modules from HopeRF. The chosen RF transceiver was the HopeRF RFM69HW V1.3. This module is designed to operate over a multitude of different unlicensed frequencies including 433MHz. LowPowerLab provides libraries to interface the Moteino with the RFM69HW.

Two solar panels were used for the project. The first panel, used at the tank-side, is rated at 5.6 W and has a voltage range from 6 to 9 V. The second panel, used for the pump-side system, is rated at 80W and has a voltage range from 24 to 42 V. The Texas Instruments LMZ14202 Evaluation Module board was used for each panel to give a solid 3.3-V output. The LMZ14202 is a switching regulator that can accept any voltage from 6 V to 42 V and was configured in this case to give an output of 3.3 V. During testing, the regulator continued to supply uninterrupted power to the Moteino even when the pump was power-cycled.

The biggest challenge was figuring out how to control the pump from the pump-side Moteino. Originally, a high-current, logic-level gate threshold N-Channel MOSFET was used as a low-side driver for the pump. However, the weak current drive strength from the Moteino I/O caused a slow transition in the turn-on of the MOSFET. During this partially-conductive phase, the motor demanded a large amount of startup current, which caused the MOSFET to burn up every time. Eventually, the CPC1709 optically coupled MOSFET with an added heatsink was the best solution. This device has an internal LED connected to an optical MOSFET gate. The Moteino is responsible for driving the 10-mA LED, and the MOSFET turns on quickly, producing no noticeable heat when the motor starts. Two 10-amp protection diodes on the pump-side circuit protect against overvoltage from the solar panel and from inductive motor voltage spikes.

The tank-side Moteino transmits a control signal every 5 seconds when the tank is empty. When the pump-side receives the signal, it makes a decision on whether to pump. Is the panel voltage high enough? Is the temperature too low? The system does not work at night or in freezing weather, but this is not an issue as the system only needs to top-off the water levels in the tank during the spring and summer months.

Introduction

Background

The project is located in Burdett, NY. The setting is a rural area that is used as a personal organic farm. The goal is to refill the water tank using water available from the pond 100 meters through the shrubbery, as seen in the satellite image below.

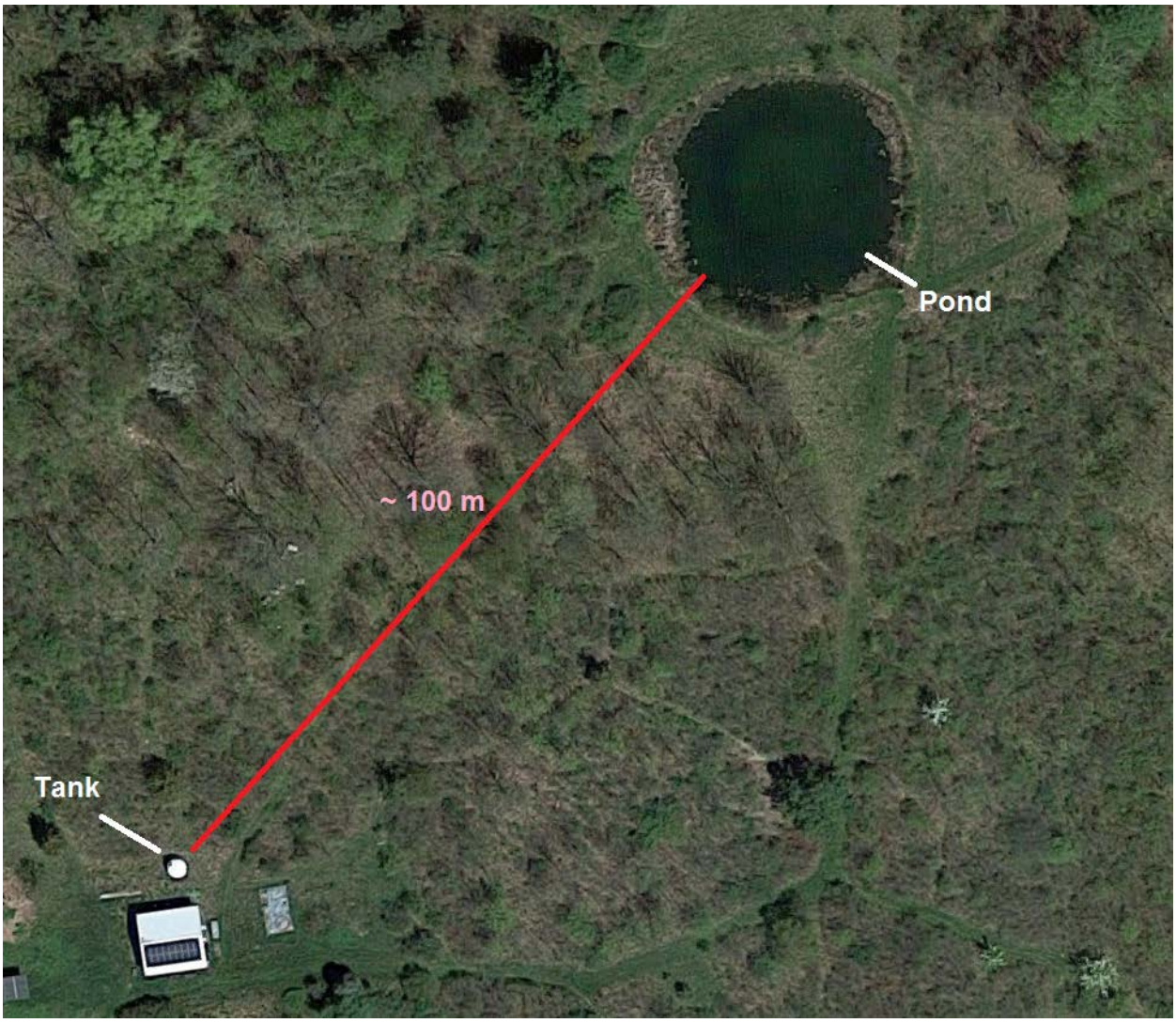

Figure 1: Google Maps satellite image of the tank and pond.

The tank only needs to be filled during the growing months in the spring and summer. Collected water cannot be used during the winter months because it is mostly frozen anyway. In addition, the pump must never be activated during the winter months because attempting to pump frozen water may cause the pump to fail and burn up.

Available Materials

A Shurflo 2088 24-VDC water pump is available from a previous attempt at this project. An SJE Micromaster float switch is also available. The float switch has two terminals. When the water tank is full and the floating part of the switch turns upward, the two terminals become a closed circuit. When the tank is not full and the switch is facing downward, the terminals are isolated as an open circuit. This will be used to control the transmission of the signal from the tank to the pump.

An [INSERT BRAND NAME] 80-W rated solar panel is available. This solar panel is rated at a 24-V output, but it has been measured to go as high as 42 V in cold, sunny weather. In fact, this 42-V was unexpected by another group and caused their voltage regulation circuit to permanently fail.

A 2-W Coleman 58012 solar panel is available and was used as a previous attempt for the transmit side. This panel has a car battery plug built in, and its intended use is to keep a car battery from dying after a long time spent away from a car. This panel is rated at 12 V.

Accessories for the Shurflo pump include a filter cap and hose that can be run for 100 meters. The filter will be placed on the input side of the pump to block as much debris as possible while absorbing water from the pond.

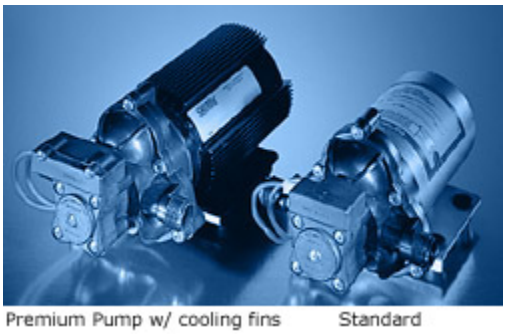

Figure 2: Shurflo 2088 series pump. The model used is the Standard on the right. (shurflo.com)

Major Component Selection

Transmit-side solar panel

The Coleman 2-W solar panel was tested using an off-brand 12V –to-5V USB car power adapter. This car adapter contained a MC34063A switching regular onboard. According to manufacturer datasheets, this industry-standard regulator is over 80% efficient at down-converting higher voltages to 5 V outputs. However, after testing the Coleman solar panel's ability to power a microcontroller through this regulator, the power was only high enough to operate the unit in full, direct sun. This panel did not do a very good job of keeping its power at a consistent 2 W. Also, the panel was designed for use inside a car, and was therefore not hardened to the rough outdoor weather conditions of upstate New York.

The solution was to choose a 6-V, 5.6-W solar panel. This device is product 1525 at Adafruit. In tests, this panel could easily power the entire transmit system even in partial shade and in evenings.

Figure 3: Adafruit "Huge 6V 5.6W Solar panel" used at the transmit side (adafruit.com)

Power Regulator

In order to conserve as much energy as possible – and avoid having to buy larger solar panels or sacrifice the power for the motor – the power solution of choice was a switching regulator. While linear regulators are the easiest to design and are the cheapest to purchase, their efficiency is equal to Vout / Vin. In the case of the 42-V panel powering a 3.3-V system, this means that at 92% of the power used for the microcontroller will be wasted as heat. Not only does this leave less power for the operation of the motor – it also creates an issue where excessive heat is being created in the pump-side electronic system.

After extensive research, the regulator of choice was the Texas Instruments LMZ14201 SIMPLE SWITCHER® series switching regulator. At a 42V input at 3.3-V output, the regulator is specified as dissipating only 200mW of energy as heat, which is low enough not to be a concern in the system. This regulator was chosen for its wide input range of 6 V to 42 V. This range will allow the regulator to operate even near the peak measured solar panel voltage. It will also prevent the problem that the previous group had – where the regulator burned up when exposed to the 42V panel. Texas Instruments offers an evaluation module for this device. The evaluation module contains a pre-built board which is configured with all capacitors and resistors needed to accept 8 V to 42 V and output 3.3 V at 1 A maximum. The 8 V minimum for this board is due to a default resistor divider setting which activates the UVLO (Under-voltage lockout) feature at 8 V or below. However, this resistor divider can be adjusted to accept voltages as low as 6 V. The UVLO feature is very useful for brownout detection, and it may help prevent unwanted behavior from a partially-powered microcontroller.

Figure 4: National Semiconductor (Acquired by Texas Instruments) LMZ14201 evaluation board. This is a picture of the 3A model, but the 1A model looks virtually identical. (ti.com)

Wireless Device and Microcontroller

Originally, the idea was to use a standard Arduino or Texas Instruments MSP430 Launchpad. The Launchpad was the initial preference because of its low power consumption and onboard temperature sensor. Simple microcontrollers are appropriate for this application because we will be sending a simple on/off signal. After choosing the microcontroller, a 433-MHz RF module would be interfaced to the board.

After substantial research, the Moteino-USB microcontroller from LowPowerLab was the best suited for this application. The Moteino is a re-designed Arduino Uno which is shaped appropriately to fit pre-built HopeRF modules, such as the RFM69HW. The Moteino is compatible with the standard Arduino GUI under the “Arduino Uno” configuration. LowPowerLab offers the option to purchase the Moteino by itself or to purchase with an RFM69HW pre-installed on the board, as shown below in Figure 5. There is also an option for pre-ordering with additional SPI flash memory, but for a simple application like this one, the on-chip flash memory is sufficient.

The Moteino-USB with the RFM69HW pre-soldered was ultimately chosen. LowPowerLab offers Arduino libraries for download that are compatible with the Moteino-USB and RFM69HW that contain simple API calls like radio.sendWithRetry() and radio.receiveDone(). They also include many sample programs. The ones used for this project were modified versions of the “Struct_send” and “Struct_receive” examples which are able to send a few bytes of data across a 433MHz channel.

The on-board 433MHz RF transceiver and built-in libraries took almost all of the headache out of designing the RF component of the project.

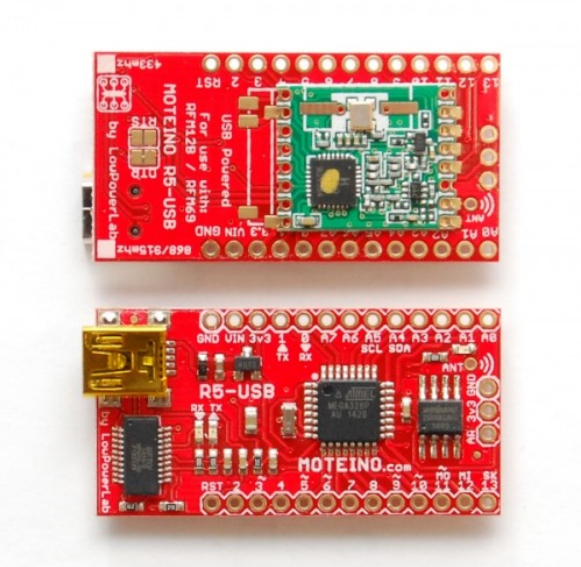

Figure 5: Moteino-USB with onboard RFM69HW (lowpowerlab.com)

Circuit Design

Pump-side Circuit

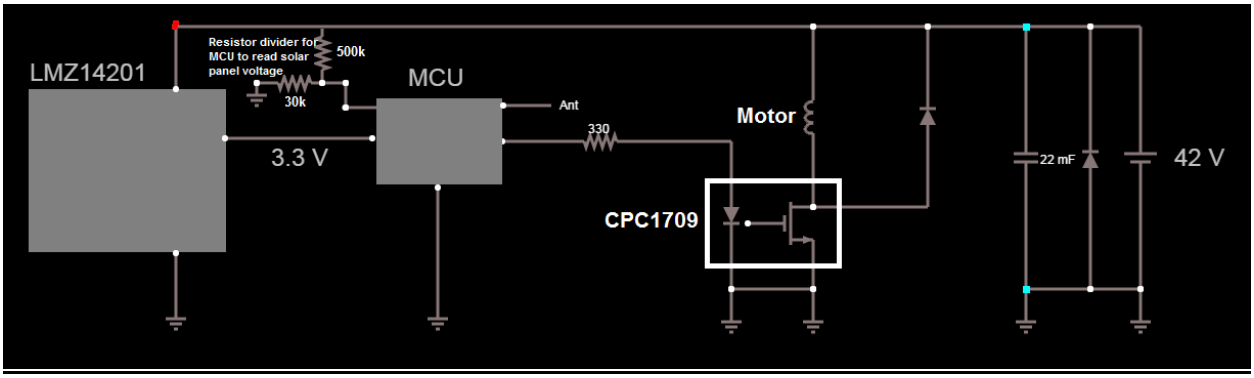

Figure 6: Pump-Side Circuit Schematic (Drawn from falstad.com/circuit)

The pump-side circuit has to accomplish two things: first, it needs to give the motor a low-resistance path to GND whenever pump operation is desired. Second, it needs to keep the MCU powered even during the transition between the on and off states of the motor.

At the right, there is the solar panel which tends to max out at 42 V. In case the voltage drifts beyond that value, there is a SR1045 10-A rated diode attached in parallel that will break down at 45 V and prevent the panel from going any higher.

To accommodate for the large initial spike in current that the motor will draw upon being turned on, there is also a 22mF capacitor in parallel with the supply. This large capacitor helps balance the load on the solar panel; without it, the motor would draw far too much current for the solar panel to handle, and the panel voltage would drop to nearly zero. Then, the MCU would lose power and turn off the motor. As a result, the panel voltage would climb back up until the MCU and motor can be powered, then the entire process will repeat again. This is an unwanted oscillation and the capacitor helps to mitigate it.

The motor and another SR1045 diode are placed in parallel and isolated from ground by the N-Channel MOSFET of a CPC1709 optoisolator. When the microcontroller turns on the optoisolator through its optical diode, the FET turns on and conducts. Then, the motor and diode essentially have a proper ground connection and the motor begins to run. Once the motor loses power, it discharges its excess current through the SR1045, which acts as a clamp diode for the inductive voltage spike.

The LMZ14201 evaluation module powers the microcontroller. The default UVLO configured on the module board is 8 V, meaning any input voltage below 8 V will result in a turned-off output. This is OK because the 42V panel rarely drops below that voltage, even when the motor is turned on. The MCU can also read the solar panel voltage as well. With a 500k to 30k resistor divider connected to an analog input, the MCU can safely read a scaled-down version of the solar panel voltage. Based on experimental results, every 1 V increase in the panel voltage causes the analog read value to increase by about 18 (17.7 to be more exact, but the pin is read as an integer.)

Tank-side Circuit

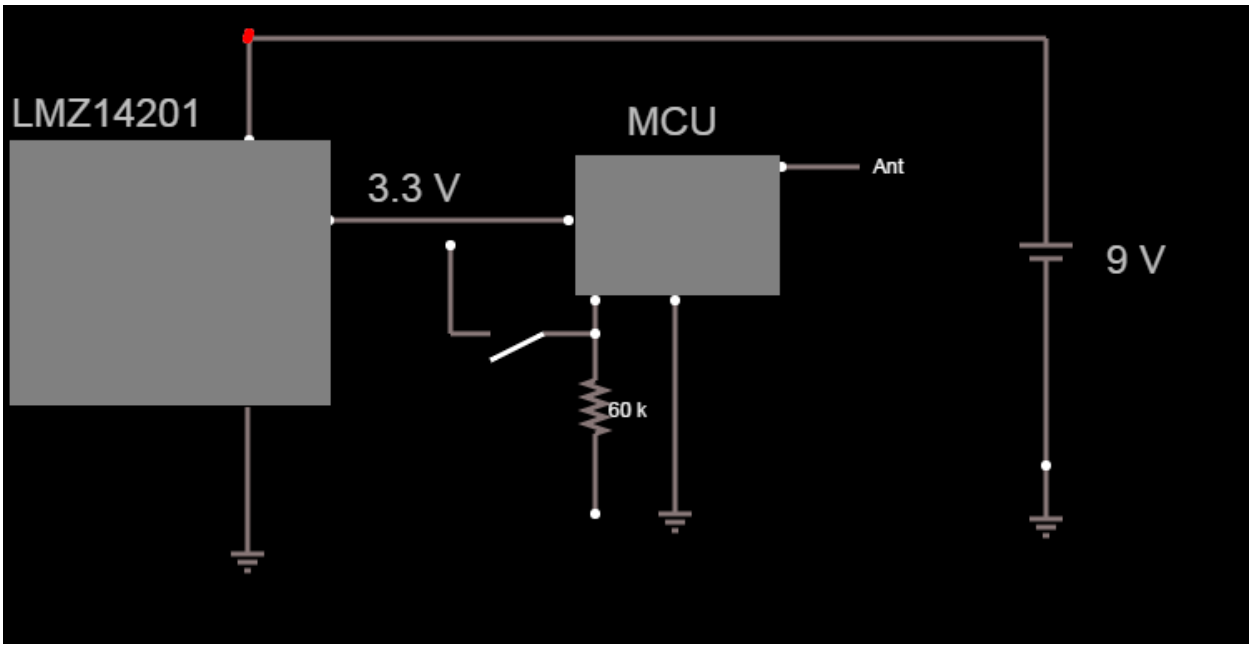

Figure 7: Tank-side Circuit Schematic (Drawn from falstad.com/circuit)

The tank-side circuit is fortunately much simpler than the pump side. Instead of running from a 42V panel, the tank side runs from a 9 V panel. However, this panel voltage can easily drop below the default UVLO of 8 V, so the resistor divider network on the LMZ14201EVM needed to be adjusted so that the MCU would operate with a panel voltage of as low as 6 V, the minimum input of the LMZ14201 integrated circuit itself.

The one other major design change in the tank-side circuit is the addition of a float switch. This switch will be placed in the tank. Once the water rises high enough, the float switch will float up and point upwards, creating a connection between a GPIO and 3.3 V. This will signal to the MCU that the tank is full and does not need to be pumped with water anymore. If the tank is not full, the float switch will be hanging down and will remain an open circuit, allowing the pin on the MCU to be pulled down to GND.

Software Design

The final code is composed of two modified versions of struct_send and struct_receive. I called them TX.ino and RX.ino. In each file, the LowPowerLab RFM69HW libraries are included. The importance of the –HW in the part number indicates a high power mode that can be activated via a register setting. This register setting is easy to change in software using a command called radio.setHighPower().

The libraries allow a lot of configurability. For example, the settings include node IDs, network IDS, gateway IDs, frequencies, and encryption keys, all of which allow a large number of Moteinos to communicate independently without interference. The data to be transmitted is contained in a struct called “theData,” which is a hold-over from the example code. The payload contains three integers: a counter which counts the index number of the transmission (used for determining whether transmissions have been lost), a temperature reading, and a float switch reading indicating whether the tank needs to be filled or not. The tank-side sends these three integers to the pump, and the pump uses two of them to determine whether it should pump at all. The third integer (the counter) is used for debug to make sure that the pump side is actually receiving the data as it is being transmitted.

Fortunately, the Moteino comes with an onboard orange LED that is attached to pin 9. In this project, the orange LED is used as an error indicator. See the pump-side code listing for details on which indicator scheme means which error. For example, if the pump solar panel has too low of a voltage measurement, the orange LED will blink three times. This error indicator scheme allows the user to understand what is going wrong without having to break open the waterproof boxes and probe across the circuit with a multimeter.

Each Moteino contains code for serial transmission. This is not essential to the operation of the system, but it is important for debug. It allows the user to walk around with a laptop attached to a Moteino receiver to figure out how far the signal can actually

reach and when the transmissions start to become flaky. To turn on the pump and read the float switch, each Moteino simply uses its built-in GPIO capabilities.

Assembly and Results

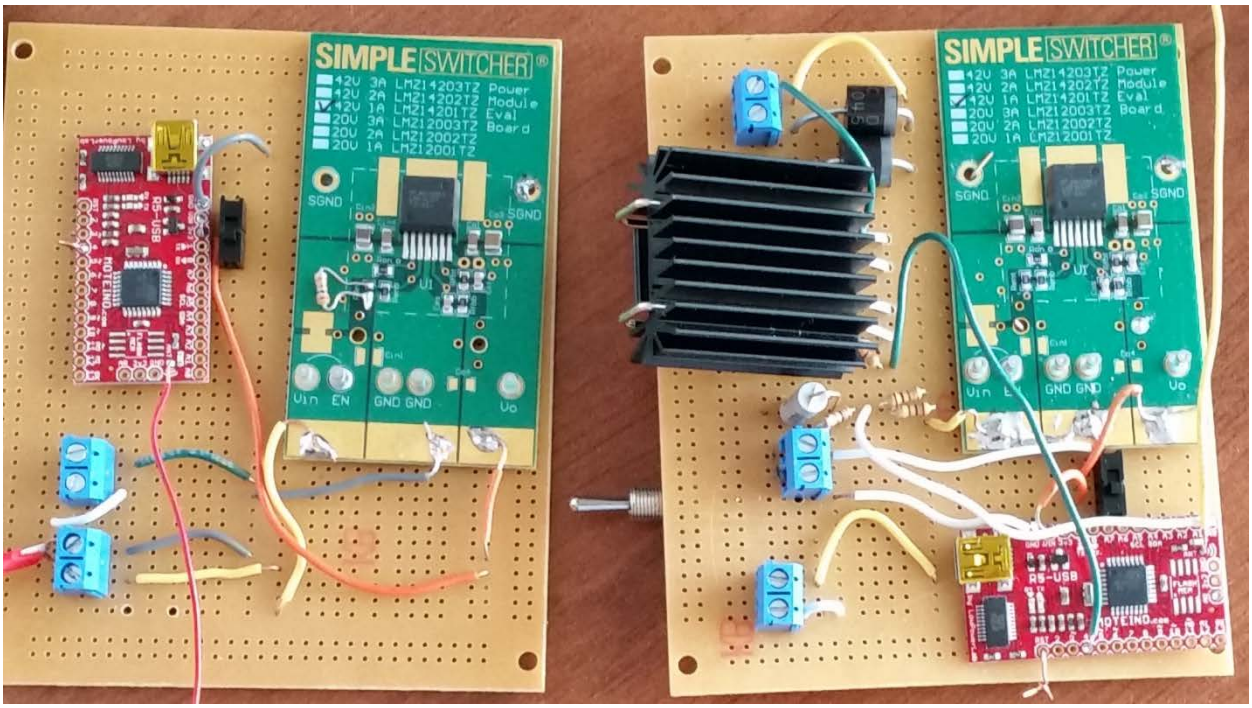

Figure 8: Tank-side (left) and pump-side (right) solder boards for project

The assembly of the project was done on a generic copper solder board rather than using a company to manufacture a PCB because the project was expected to go through many iterations before it was complete. This turned out to be true – soldering on these boards, changing components, rearranging positions of components, and adding interfaces via the soldering station became almost a daily task. The existing boards (LMZ14201 and Moteino) were fastened onto each board with simple copper wire and solder. They were interfaced to the rest of the circuit via 22AWG copper wire from the lab.

The solar panel supplies, float switch, and motor leads were attached to the boards via the blue screw terminal blocks seen above. This allowed the user to detach and re-attach components as necessary when attempting the final assembly of the project.

The most common problem during testing was the FET low-side switch burning up during motor switching. This was due to the fact that the gate of the FET switch was connected directly to the Moteino output, which has a 20mA limit. The gate capacitance of early FETs tested were less than 1 uF each, but the Moteino output ports were still too weak to drive the gate hard enough to do the switching of the motor. The motor began drawing current while the gate was still partially on, and the I^2R losses in the drain-to-source conducting region of the FET caused it to overheat and malfunction. As a result, the FET burned up after a single power cycle of the motor and turned into a resistor of about 100 ohms.

Eventually, the opto-isolated CPC1709 proved to be the best option. The Moteino only had to power an internal LED, and the gate of the CPC1709 was optically coupled to that LED. The optocoupler was chosen over a solid-state relay due to its price on Digikey being 5 to 10 times less. In addition, Digikey sells a compatible heat-sink which was attached to the thermal pad of the device and can be seen in the image above. With a heatsink, the CPC1709 is rated at 36 A, far higher than the current drawn by the motor. After testing the motor with the CPC1709, it was found to produce almost no heat as compared to the burning FETs that were tested before.

For debug purposes, power switches were added between the LMZ14201 and each Moteino. It is a best practice to use the switches to detach the Moteino from the power regulator before updating the code via USB. However, after accidentally plugging in the USB during powered-on mode, it was found that this is not too big of an issue. The voltage difference between the USB input and the regulator output is not that great, and therefore there was no issue with overheating or blowing of any power supply.

For final assembly, a Bud Industries polycarbonate electrical box was used to house each board for weather-proofing. The image below shows the Digikey photo of the box:

Figure 9: Bud Industries polycarbonate electronics box (digikey.com).

The box has screws and a rubber gasket to protect against moisture. The size of the box was small enough to be portable but large enough to hold the boards and give the cables some flexibility. Holes were drilled in the clear plastic part of the box and used to insert all of the external cables necessary (power, float switch, motor). Then, large cables were fastened to the box with cable clamps to prevent excessive torsion on the cables as the housing was moved. Silicone rubber fills the remaining holes to prevent water from leaking into the spots where cables are inserted in the box.

The final assembly of the project, including the polycarbonate housing, is shown in the images below.

Figure 10: Final assembly of tank-side panel and float switch

Figure 11: Final assembly of pump-side panel, circuit, and motor (opaque half of box removed for visual).

Final Testing and Results

The entire system was tested with a bucket of water and moved to Burdett for final functionality testing. It was tested on a sunny Friday afternoon that would have been the last sunny day before the project was due.

There were a few errors in the code that needed to be fixed before the project was declared as working. The most prominent error was the fact that, once the Moteino started running the pump, it would flash three blinks of the LED for every transmission it received from the tank side, even if the tank side was telling it to turn off the pump. This was because, in testing the transmission, the pump-side would check its own panel voltage before taking any action. When the pump is running, the panel voltage drops below the action threshold. The code had to be changed to check the panel voltage ONLY if the motor is not currently running. Once this change was made, the pump-side responded appropriately to the tank-side's request for water or no water.

After it was verified that the pump was responding appropriately to the orientation of the float switch, the next step was to test the transmission to see if the signal could reach from the pump to the pond. To test this, the transmitter was brought out to the pond while the pump-side sat near the tank (opposite of final configuration, but irrelevant because the transmission is equal in both directions). Using the USB port on the computer, the pump-side reported that it received most of the transmissions at -85 dBm, which is more than appropriate for the system to work as intended. All parts of the system were determined to work as expected on this final testing day.

Code Listings

Tank-side (Transmit) Adruino code (based on LowPowerLab.com example)

```
#include <RFM69.h>
#include <SPI.h>
#include <SPIFlash.h>

#define NODEID      99
#define NETWORKID   100
#define GATEWAYID   4

//Match this with the version of your Moteino! (others: RF69_433MHZ, RF69_868MHZ)
#define FREQUENCY   RF69_433MHZ

//has to be same 16 characters/bytes on all nodes, not more not less!
#define KEY         "sampleEncryptKey"
#define LED         9
#define SERIAL_BAUD 115200

// # of ms to wait for an ack
#define ACK_TIME    250

//transmit a packet to gateway so often (in ms)
```

```
int TRANSMITPERIOD = 5000;
RFM69 radio;


//Define the structure to be send over radio
typedef struct {
  int           numberCounter;
  int           needsWater;
  int           temperature;
} Payload;

Payload theData;

void setup() {
  radio.initialize(FREQUENCY,NODEID,NETWORKID);
  radio.setHighPower(); //RFM69HW
  radio.encrypt(KEY);
  theData.numberCounter=1;

}

long lastPeriod = -1;
void loop() {

    //fill in the struct with new values
    theData.numberCounter = theData.numberCounter + 1;
    theData.needsWater = 0; //Default, does not need water.
    theData.temperature = radio.readTemperature(0);
    if (!tankIsFull()){
      Blink(LED,100);
      theData.needsWater = 1;

    }else{
      theData.needsWater=0;
    }
       radio.sendWithRetry(GATEWAYID, (const void*)(&theData), sizeof(theData));

    Blink(LED,100);

  delay(TRANSMITPERIOD);
}

void Blink(byte PIN, int DELAY_MS)
{
  pinMode(PIN, OUTPUT);
  digitalWrite(PIN,HIGH);
  delay(DELAY_MS);
  digitalWrite(PIN,LOW);
  delay(DELAY_MS);
}

int tankIsFull(){
  pinMode(4,INPUT);
  if (digitalRead(4)){ //Float switch down, pin4 50k pullup to VCC, tank empty
    return 0;
  }else{ //Float switch up, pin4 pulled to GND, tank full
    return 1;
  }
}
```

Pump-side (Receive) Adruino code (based on LowPowerLab.com example)

```
#include <RFM69.h>
#include <SPI.h>
#include <SPIFlash.h>

#define NODEID      4
#define NETWORKID   100

//Match this with the version of your Moteino! (others: RF69_433MHZ, RF69_868MHZ)
#define FREQUENCY   RF69_433MHZ

//has to be same 16 characters/bytes on all nodes, not more not less!
#define KEY         "sampleEncryptKey"
#define LED         9
#define GATE        4
#define GATE_PERIOD 1000
#define ONE_SECOND  1000
#define SERIAL_BAUD 115200

// # of ms to wait for an ack
#define ACK_TIME    150

// Orange LED Blinking Signal Indicators  //
// Error correlated with number of blinks //

#define signal_received                         1
#define signal_missed_too_many_transmissions    2
#define voltage_too_low                         3
#define temperature_too_low                     4

// Parameters for Pump Operation //

#define minimumPumpTemp                         5
#define minimumPanelVoltage                     24
#define voltageToADCMultiplier                  17.7

RFM69 radio;

//Pump running indicator
int pumpIsRunning = 0;

//set to 'true' to sniff all packets on the same network
bool promiscuousMode = false;

typedef struct {
  int           numberCounter; //store this nodeId
  int           needsWater;
  int           temperature;
} Payload;
Payload theData;

void setup() {
  delay(5000);
  Serial.begin(SERIAL_BAUD);
  delay(10);
  radio.initialize(FREQUENCY,NODEID,NETWORKID);
  radio.setHighPower(); //RFM69HW
  radio.encrypt(KEY);
  radio.promiscuous(promiscuousMode);
  Serial.print("count,RX_RSSI,adc,temp"); //debug
  Serial.println();
}
```

```
byte ackCount=0;
void loop() {

  //Read ADC value, this is panel voltage * 17.7
  if (radio.receiveDone())
  {
    if (radio.DATALEN != sizeof(Payload))
      Serial.print("Invalid payload received, not matching Payload struct!");
    else //If data is good
    {
      //Indicate if there are any problems with the system.
      if (analogRead(0) < minimumPanelVoltage * voltageToADCMultiplier &&
pumpIsRunning == 0 ){
        statusIndicator(voltage_too_low);
      }else if (theData.temperature < minimumPumpTemp){
        statusIndicator(temperature_too_low);
      }else if (!(theData.needsWater)){
        TurnOffMotor();
        delay(15000); //Cooldown time.
      }else{
        statusIndicator(signal_received);
        TurnOnMotor();
      }
      theData = *(Payload*)radio.DATA;
      //Serial.print("count=");
      Serial.print(theData.numberCounter);
      Serial.print(",");
      Serial.print(radio.readRSSI());
      Serial.print(",");
      Serial.print(theData.needsWater);
      Serial.print(",");
      Serial.print(theData.temperature);
      Serial.println();
    }
  }

}

//Blinks the LED according to status of the system.
//Number of blinks corresponds to error value.

void statusIndicator(int indicator_value){
  while (indicator_value--){
    Blink(LED, 150);
  }
}

//Simple orange LED blink routine

void Blink(byte PIN, int DELAY_MS)
{
  pinMode(PIN, OUTPUT);
  digitalWrite(PIN,HIGH);
  delay(DELAY_MS);
  digitalWrite(PIN,LOW);
  delay(DELAY_MS);
}

void TurnOnMotor(){

  pumpIsRunning = 1;
```

```
  //Turn on light to indicate motor usage
  pinMode(LED,OUTPUT);
  digitalWrite(LED,HIGH);

  //Open up the motor for a set duration
  pinMode(4,OUTPUT);
  digitalWrite(4,HIGH);

}

void TurnOffMotor(){

  pumpIsRunning = 0;

  //Turn off the motor pin
  digitalWrite(4,LOW);

  //Turn off orange LED once motor is finished.
  digitalWrite(LED,LOW);
}
```