Single Axis Solar Tracking System

Final Project Report
December 12th, 2019
E155

Akshay Trikha and Kahiwa Hoe

Abstract:

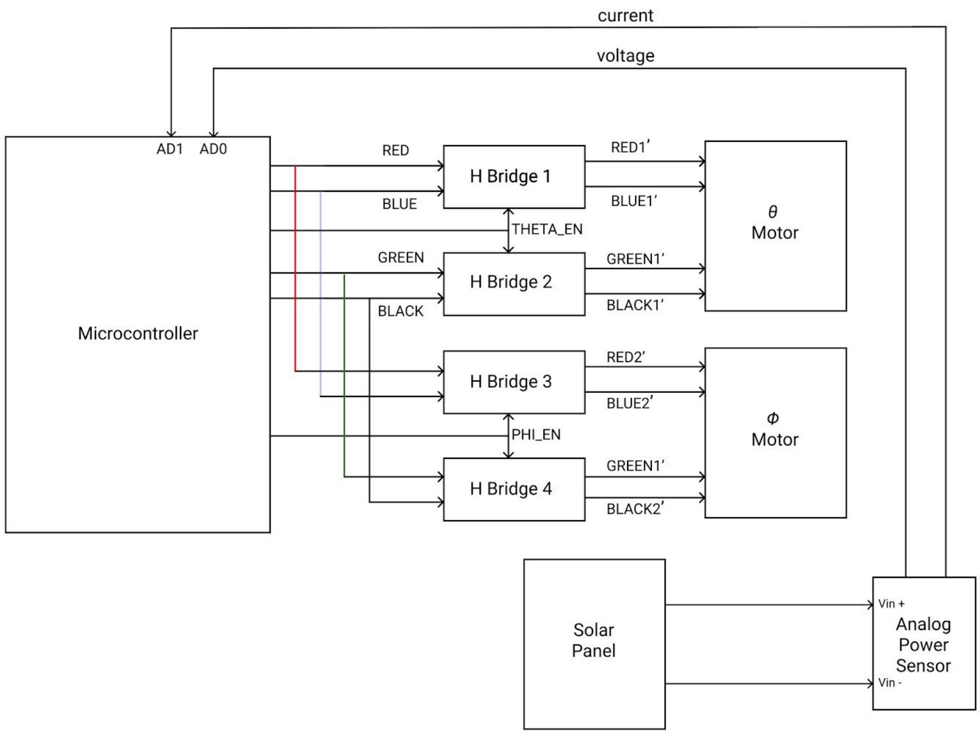
As our sun travels in its trajectory throughout the day, the optimal angle to harness as much of its sunlight also changes. However, most solar panels when set up are placed at a set angle and stay put for their lifetimes. The variation in the sun's position is significant enough to noticeably change the incident light intensity, and thus power output of solar panels. Our system tracks the sun's position with respect to a solar panel by monitoring the power output of the panel and adjusting its position accordingly through driving a stepper motor.

Introduction

It is our mission as engineers to create devices that will benefit our society, and in particular will combat climate change. Actualizing our love for renewable energy and our shared environment, we aimed to create a solar tracking system to harness as much of the sun's radiant energy as possible. The variation in light intensity noticed by a fixed solar panel throughout the day is significant enough to motivate us to create a system that keeps a panel perpendicular to the sun. On top of maximizing power output of a solar panel, we also wanted to minimize power consumption of the tracking system.

Our system consists of a microcontroller, solar panel, analog power sensor, and two motors and H bridges. The microcontroller is used to read the power sensor and implements a routine control algorithm to update the panel's orientation. It then goes into sleep mode for a programmable amount of time, until the next cycle.

Figure Data 1. Data Flow Diagram



As shown in Figure 1. the power sensor measures the current power output of the solar panel

The microcontroller sends out signals enabling the desired H bridges, which drive the respective motors. The power sensor reads the power output of the solar panel and sends that data to the microcontroller, which makes the decision of which motor to step and by how much.

New Hardware

Our key new pieces of hardware was a stepper motor, H-bridges to drive it, and operational amplifier for power measurement. Stepper motors function by alternating the direction of a magnetic field, to which an internal permanent magnet aligns. Two pairs of colored input wires: black with green and red with blue are used to control the magnetic field by passing current in either direction. For each wire pair, the current flows from the wire of a higher voltage towards the lower voltage. The 23HS22 stepper motor was rated for a current draw of 2.8 A and voltage of 12V with a holding torque of 1.26 N•m. Given our relatively light solar panel we found that it was satisfactory to drive it at 1.1A and 2.5V - which helped reduce our systems power consumption. Specifications on voltage levels for a turning cycle were given by the datasheet, and the number of cycles required to complete a full rotation was given as 200 steps, giving a precision of 1.8º per step.

In order to facilitate the large current draw, three L293DNE H-bridges were stacked and used to redirect power from a Model 6224A power supply, capable of generating 3 A of current, to the stepper motors. The H-bridge includes enable signals each corresponding to two input and output pairs. The integrated circuit (IC) is given two input voltage sources: V_{CC1} and V_{CC2}, corresponding to the logic level source (5 V) and driver source (12 V), respectively. The motor signals are given input voltages ranging from 0 to 3.3 V for on and off settings, respectively, and the output is a corresponding on or off signal with increased current. There are also two enable signals, each controlling a pair of inputs, that make the output follow the input only when the enable is set high. The datasheet only rates the H-bridges for an output of 600 mA, so three H-bridges were soldered onto each other to increase the overall capability to 1.8 A. The L93DNE H-bridges also have a diode system in place to protect the system against large increases in voltage from inductive kicks when the motor is abruptly turned off.

An MCP6002 operational amplifier was used to amplify the voltage across a small resistor. The op-amp is used in a non-inverting amplification circuit shown below:

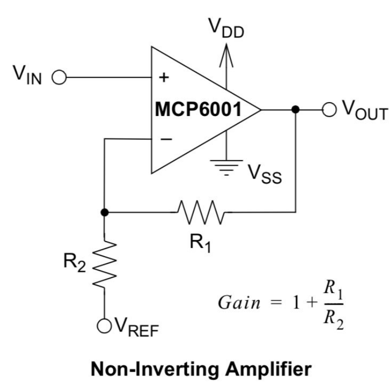


Figure 2. Non-inverting amplifier circuit taken from datasheet using MCP6001 op-amp

Note that the MCP6002 consists of two MCP6001 op-amps combined into a single IC. The input voltage V_{IN} is related to the output V_{OUT} by the following equation:

$$V_{OUT} = (1 + \frac{R_1}{R_2})V_{IN} \quad (Eq.\ 1)$$

The value of the resistors R_1 and R_2 are 10 kΩ and 1 kΩ, respectively, thus amplifying the input signal by a factor of 11. This gives the ADC greater precision in measuring the voltage across the small resistor. If the theoretical value of V_{OUT} exceeds V_{DD} or goes below V_{SS}, the voltage will rail out to the nearest bounding voltage. The op-amp receives a V_{DD} of 3.3 V and V_{SS} of 0 to limit the voltage range to that of the microcontroller's ADC.

The op-amp was a part of a larger power measurement circuit, which consists of a main resistive load, voltage divider, smaller resistor, and the non-inverting amplifier. The main resistive load is 30 Ω in order to match the internal resistance of the solar panel, maximizing its power output. The voltage divider's output is proportional to the voltage across the solar panel reduced by a factor of 11. The smaller resistor of 1 Ω acts as a lowside measurement of the panel's current, giving a voltage directly proportional to the current through the resistor according to Ohm's Law. The voltage on the highside of the small resistor was passed through the non-inverting amplifier to output a voltage 11 times that of the resistor.

Schematics

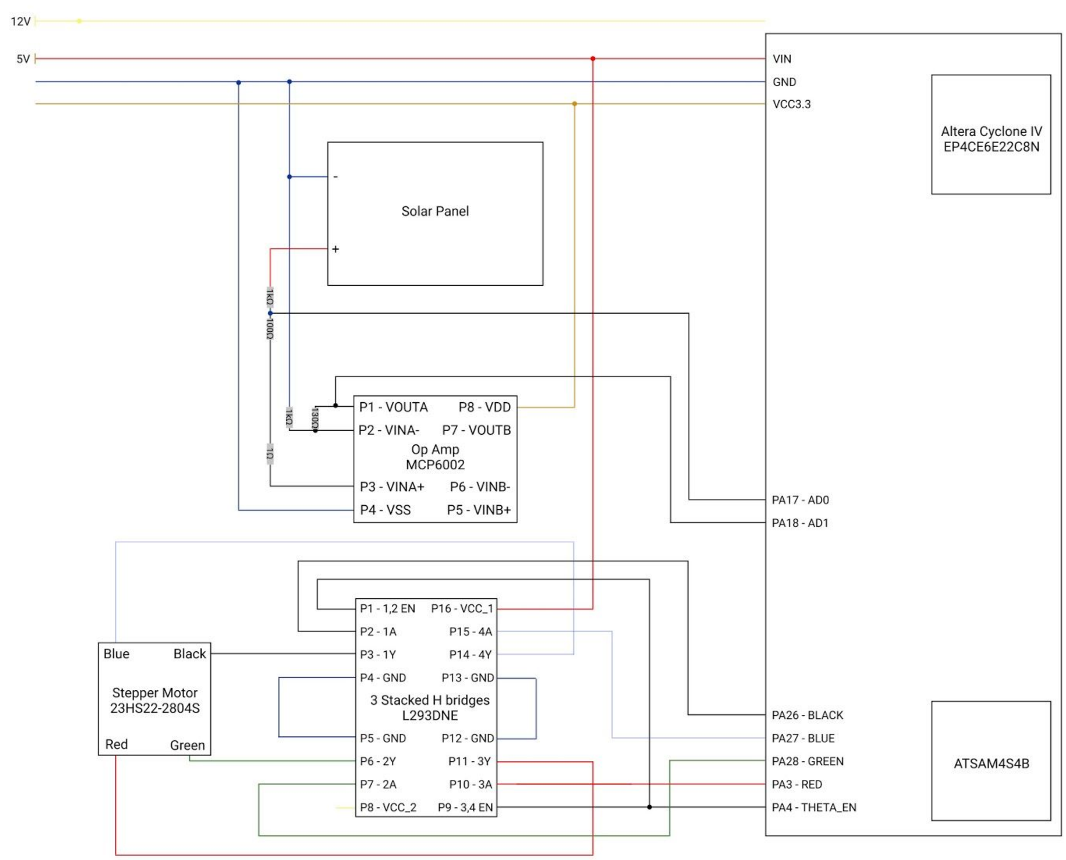


Figure 3. Overall system schematics including pin assignments

Microcontroller Design

The ATSAM4S4B microcontroller has been tasked with driving the motor input signals, measuring the solar panel's power output, and enacting a simple search algorithm to maximize power from the solar panel. A The stepAnglePositive function utilized the general peripheral functions to drive the black (PA26), blue (PA27), green (PA28), and red (PA3) wire logic pins in the clockwise sequence given by the datasheet (the stepAngleNegative function performs this sequence in reverse). The functions were made to receive an input angle and frequency for managing angular speed. The hold function is given a duration and pulses the motor enable every second to maintain the current angular position of the panel.

The getPower function utilizes the analog-to-digital converter (ADC) peripherals onboard to sample the voltage divider and non-inverting amplifier from the power measurement circuit. These values are then scaled by the inverse of their respective gains and multiplied to get the current power output of the solar panel. The getPowerAvg gives a 5-point average of the voltage reading sampled at a rate of 20 kHz.

The search algorithm stores the currentPower variable set to the value of getPowerAvg at a certain time step. The search then begins incrementing clockwise by a 2º step and measures the power. This movement continues until the average power begins to decrease. When the power decreases, the panel increments counterclockwise according to the same search and holds the final position until the next search begins. Each search will be separated by a transition into backup mode, decreasing the current consumption of the ATSAM to 1 µA in order to improve the energy efficiency of the algorithm.

FPGA Design

The FPGA was intended to be responsible for keeping the timing of the system, waking up the microcontroller at hourly intervals and sending over the time of day when the ATSAM requests the data over SPI communication.

Results

The project demonstrated a working single-axis tracking system, but the final product was less energy-efficient than intended. The solar panel was able to follow a UV light generator, however the algorithm resulted in an overshoot of the ideal angle because it detects a single decrease in power and then stops. Although a backup program was created and tested, the mechanical design was unable to maintain the angle upon backup initiation, so the design consumes much more power due to the constant stepper motor input to hold the motor in position.

The most difficult parts of the design were sampling a power measurement and constructing the mechanical apparatus. The power sampling was initially to be accomplished via an INA260 power sensor, which would send its data over via I^2C protocol, but we encountered multiple issues with this peripheral on the ATSAM. Despite following the ATSAM's peripheral access diagram and checking the associated registers via the Keil µVision debugger, we were unable to trace down the error in our initialization and communication process. We spent 3 weeks working through this bug and eventually settled on the analog power measurement circuit used in the final product in the interest of time and completing a working demonstration.

The overall mechanical apparatus and motor connection can be seen below:

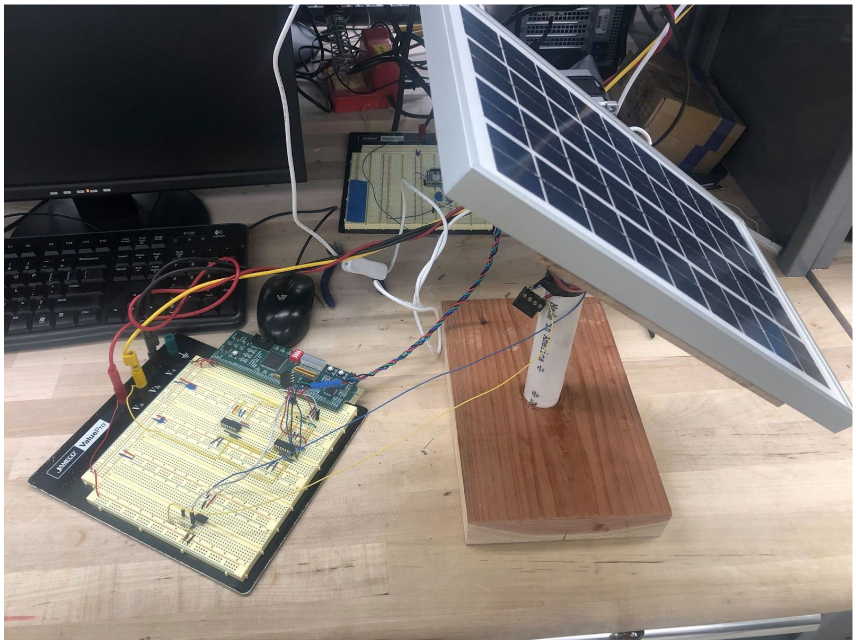


Figure 4. Solar panel with stand and motorized axis

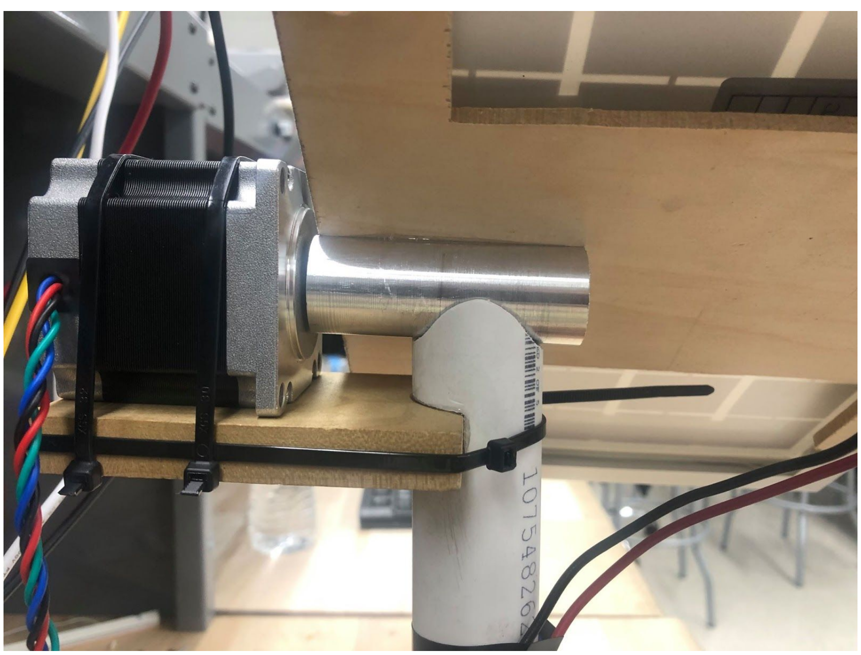


Figure 5. Stepper motor on supporting platform connected to solar panel mount

Attaching the stepper motor's axis to a mountable axis proved the most difficult in the mechanical design because of the metal lathe and CNC mill machining required to fit the pieces together. The mount design also proved difficult because of the need to attach the piece without altering the existing panel form.

The FPGA was intended to manage the timing of the backup mode initialization, but we prioritized erecting a working tracking system because of time constraints.

Overall, we'd still say that our project was a success and our proud of what we have accomplished given our time constraint.

References

[1] Rosetti, L. (2012).*The Designing, Building, and Testing of an Azimuthal-Altitude Dual-Axis Solar Tracker*. Undergraduate. Pomona College.

[2] Harris, Sarah L., and David Money Harris. "Chapter 9: I/O Systems." Digital Design and Computer Architecture, Elsevier/Morgan Kaufmann, 2016.

[3] 23HS22-2804S Full Datasheet Nema Stepper Motor https://www.oyostepper.com/images/upload/File/23HS22-2804S.pdf

[4] MCP6001/1R/1U/2/4 Datasheet http://ww1.microchip.com/downloads/en/DeviceDoc/20001733K.pdf

[5] L293x Quadruple Half-H Drivers Datasheet http://www.ti.com/lit/ds/symlink/l293.pdf

Parts List

Part	Source	Vendor Part #	Price
12V Solar Panel			Available in Physics dept.
2.8A 1.26Nm Stepper Motor	Nema	23HS22-2804S	$21.99
MCP6002 Operational Amplifier	Ximimark	MCP6002	$4.98
L293DNE H Bridge		L293DNE	Available in digital lab
Model 6224A Power Supply		6224A	Available in stockroom
		Grand Total	**$48.96**

Appendices

```c
/*
  Akshay Trikha & Kahiwa Hoe
  atrikha@hmc.edu & khoe@hmc.edu
  20th November, 2019

  main file for solar tracker SAM4S4B code
*/

// includes
#include <stdio.h>
#include <stdint.h>
#include "SAM4S4B/SAM4S4B.h"

// pin definitions for theta/phi motors
#define RED         PIO_PA15
#define BLUE        PIO_PA27
#define BLACK       PIO_PA26
#define GREEN       PIO_PA28
#define THETA_EN    PIO_PA16

#define GREEN_LED   PIO_PA10
#define RED_LED     PIO_PA8

#define DELAY 300 // ms

// steps motor in positive direction at frequency Hz for duration seconds

// steps motor in positive direction through angle in degrees, with delay in ms between steps
void stepAnglePositive(float angle, uint32_t delay) {
  uint8_t maxSteps = (uint8_t) (angle / 1.8);
  uint8_t steps = 0;

  while(steps < maxSteps) {
    // enable motor signals on H-Bridge
    pioDigitalWrite(THETA_EN, 1);

    // step 0
    pioDigitalWrite(BLACK, 1);
    pioDigitalWrite(RED, 1);
    pioDigitalWrite(GREEN, 0);
    pioDigitalWrite(BLUE, 0);
    tcDelayMillis(delay);
    pioDigitalWrite(BLACK, 0);
    pioDigitalWrite(RED, 0);

    // step 1
    pioDigitalWrite(BLACK, 0);
    pioDigitalWrite(RED, 1);
    pioDigitalWrite(GREEN, 1);
    pioDigitalWrite(BLUE, 0);
    tcDelayMillis(delay);
    pioDigitalWrite(RED, 0);
    pioDigitalWrite(GREEN, 0);

    // step 2
    pioDigitalWrite(BLACK, 0);
    pioDigitalWrite(RED, 0);
    pioDigitalWrite(GREEN, 1);
    pioDigitalWrite(BLUE, 1);
    tcDelayMillis(delay);
    pioDigitalWrite(GREEN, 0);
    pioDigitalWrite(BLUE, 0);

    // step 3
    pioDigitalWrite(BLACK, 1);
    pioDigitalWrite(RED, 0);
    pioDigitalWrite(GREEN, 0);
    pioDigitalWrite(BLUE, 1);
    tcDelayMillis(delay);
    pioDigitalWrite(BLACK, 0);
    pioDigitalWrite(BLUE, 0);

```

```c
    steps += 4;
  }
  // disable motor signals on H-Bridge
  pioDigitalWrite(THETA_EN, 0);
}

// steps motor in negative direction at frequency Hz for duration seconds
void stepAngleNegative(float angle, uint32_t delay) {
  uint8_t maxSteps = (uint8_t) (angle / 1.8);
  uint8_t steps = 0;

  while(steps < maxSteps) {
    // enable motor signals on H-Bridge
    pioDigitalWrite(THETA_EN, 1);

    // step 3
    pioDigitalWrite(BLACK, 1);
    pioDigitalWrite(RED, 0);
    pioDigitalWrite(GREEN, 0);
    pioDigitalWrite(BLUE, 1);
    tcDelayMillis(delay);
    pioDigitalWrite(BLACK, 0);
    pioDigitalWrite(BLUE, 0);

    // step 2
    pioDigitalWrite(BLACK, 0);
    pioDigitalWrite(RED, 0);
    pioDigitalWrite(GREEN, 1);
    pioDigitalWrite(BLUE, 1);
    tcDelayMillis(delay);
    pioDigitalWrite(GREEN, 0);
    pioDigitalWrite(BLUE, 0);

    // step 1
    pioDigitalWrite(BLACK, 0);
    pioDigitalWrite(RED, 1);
    pioDigitalWrite(GREEN, 1);
    pioDigitalWrite(BLUE, 0);
    tcDelayMillis(delay);
    pioDigitalWrite(RED, 0);
    pioDigitalWrite(GREEN, 0);

    // step 0
    pioDigitalWrite(BLACK, 1);
    pioDigitalWrite(RED, 1);
    pioDigitalWrite(GREEN, 0);
    pioDigitalWrite(BLUE, 0);
    tcDelayMillis(delay);
    pioDigitalWrite(BLACK, 0);
    pioDigitalWrite(RED, 0);

    steps += 4;
  }
  // disable motor signals on H-Bridge
  pioDigitalWrite(THETA_EN, 0);
}

// holds motor in position for given duration
void hold(uint32_t duration) {
  uint8_t steps = 0;

  // keep pulsing a step every second
  while(steps < (duration / 1000)) {
    pioDigitalWrite(THETA_EN, 1);
    // energize one motor terminal to keep it powered
    pioDigitalWrite(BLACK, 1);
    tcDelayMillis(1000);
    pioDigitalWrite(THETA_EN, 0);
    steps++;
  }
}
```

```c
// enters SAM4S4B backup (essentially sleep) mode
void enterBackup() {
  // pass SUPC system key to enable operation
  SUPC->SUPC_CR.KEY = 0xA5;
  SUPC->SUPC_CR.VROFF = 1;

  pioDigitalWrite(RED_LED, PIO_HIGH);
  pioDigitalWrite(GREEN_LED, PIO_LOW);
}


// delay function that works in backup mode by using RTC
void backupDelay() {
  // resets clock to 00:00 AM
  rtcUpdateTime(0b0000000, 0b0000000, 0b000000, 0b0);

  // delay 1 second
  while(RTC->RTC_TIMR.SEC < 0b0000001);

  pioDigitalWrite(RED_LED, PIO_LOW);
}



// exits SAM4S4B backup mode
void exitBackup() {

  // pass SYSC system key to enable operation
  SYSC_WPMR->WPKEY = 0x525443;
//  // disable SYSC write protection
  SYSC_WPMR->WPEN = 0;

  // enable WKUPT0 input
  SUPC->SUPC_WUIR.WKUPEN0 = 1;
  // wake up core power supply
  SUPC->SUPC_WUIR.WKUPT0 = 1;

  pioDigitalWrite(GREEN_LED, PIO_HIGH);


}



// returns instantaneous power of solar panel
// not actual power output of panel because ADC input had to be scaled down
float getPower() {
  float voltageData;
  float currentData;

  // read from ADC Channel 1
  voltageData = adcRead(ADC_CH0);
  // voltage divider maps down voltage for ADC
  voltageData *= 11;

  // read from ADC Channel 2
  currentData = adcRead(ADC_CH1);
  // current op-amp has gain of 10
  currentData /= 10;

  // multiply voltage x current to get power
  return voltageData * currentData;
}

// returns average of 5 power readings of solar panel
float getPowerAvg() {
  float currentPower0 = getPower();
  // short delay between readings
  tcDelayMicroseconds(50);
  float currentPower1 = getPower();
  tcDelayMicroseconds(50);
  float currentPower2 = getPower();
```

```c
  tcDelayMicroseconds(50);
  float currentPower3 = getPower();
  tcDelayMicroseconds(50);
  float currentPower4 = getPower();
  return (currentPower0 + currentPower1 + currentPower2 + currentPower3 + currentPower4) / 5;
}


// initialize SAM4S4B
void init() {
  // initialize microcontroller's PIO capabilities
  samInit();
  pioInit();
  tcDelayInit();

  // initialize ADC with 12 bit resolution
  adcInit(ADC_MR_LOWRES_BITS_12);
  adcChannelInit(ADC_CH0, ADC_CGR_GAIN_X1, ADC_COR_OFFSET_OFF);
  adcChannelInit(ADC_CH1, ADC_CGR_GAIN_X1, ADC_COR_OFFSET_OFF);

  // set motor pins as outputs
  pioPinMode(RED, PIO_OUTPUT);
  pioPinMode(BLUE, PIO_OUTPUT);
  pioPinMode(BLACK, PIO_OUTPUT);
  pioPinMode(GREEN, PIO_OUTPUT);
  pioPinMode(THETA_EN, PIO_OUTPUT);
}


int main(void) {
  // initialize SAM4S4B microcontroller
  init();

  // to keep track of bounds of frame
  uint8_t step = 0;

  // to keep track of power
  float currentPower = 0;
  float leftPower = 0;
  float rightPower = 0;

  // motor needs to align magnetic fields
  pioDigitalWrite(BLACK, 0);
  pioDigitalWrite(RED, 0);
  pioDigitalWrite(GREEN, 0);
  pioDigitalWrite(BLUE, 0);

  // keep looping
  while(1) {
    // get current power output of panel

    currentPower = getPowerAvg();
    hold(1000);


    while(1) {
      // step right and check power output
      if (step <= 40) {
        stepAnglePositive(2, 1000);
        step++;
        rightPower = getPowerAvg();

        if (rightPower < currentPower) {
          break;
        } else {
          currentPower = rightPower;
        }
      }
    }

    while(1) {
      // step left and check power output
```

```
        if (step >= 0) {
          stepAngleNegative(2,
    1000
     );
          step--;
          leftPower = getPowerAvg();

          if (leftPower < currentPower) {
            break;
          } else {
            currentPower = leftPower;
          }
        }
      }
    }

    return 0;
  }
```

```c
/* SAM4S4B_pmc.h
 *
 * atrikha@hmc.edu
 * 20th November, 2019
 *
 * Contains base address locations, register structs, definitions, and functions for the SUPC
 * peripheral (Supply Controller) of the SAM4S4B microcontroller. */

#ifndef SAM4S4B_SUPC_H
#define SAM4S4B_SUPC_H

#include <stdint.h>

////////////////////////////////////////////////////////////////////////////////////////////////////
// SUPC Base Address Definitions
////////////////////////////////////////////////////////////////////////////////////////////////////

#define SUPC_BASE   (0x400E1410) // SUPC Base Address

////////////////////////////////////////////////////////////////////////////////////////////////////
// SUPC Registers
////////////////////////////////////////////////////////////////////////////////////////////////////

// Bit field struct for the SUPC_CR register
typedef struct {
    volatile uint32_t           : 2;
    volatile uint32_t VROFF     : 1;
    volatile uint32_t XTALSEL   : 1;
    volatile uint32_t           : 20;
    volatile uint32_t KEY       : 8;
} SUPC_CR_bits;

// Bit field struct for the SUPC_SMMR register
typedef struct {
    volatile uint32_t SMTH     : 4;
    volatile uint32_t          : 4;
    volatile uint32_t SMSMPL   : 3;
    volatile uint32_t          : 1;
    volatile uint32_t SMRSTEN  : 1;
    volatile uint32_t SMIEN    : 1;
    volatile uint32_t          : 18;
} SUPC_SMMR_bits;

// Bit field struct for the SUPC_MR register
typedef struct {
    volatile uint32_t           : 12;
    volatile uint32_t BODRSTEN  : 1;
    volatile uint32_t BODDIS    : 1;
    volatile uint32_t ONREG     : 1;
    volatile uint32_t           : 5;
    volatile uint32_t OSCBYPASS : 1;
    volatile uint32_t           : 3;
    volatile uint32_t KEY       : 8;
} SUPC_MR_bits;

// Bit field struct for the SUPC_WUMR register
typedef struct {
    volatile uint32_t           : 1;
    volatile uint32_t SMEN      : 1;
    volatile uint32_t RTTEN     : 1;
    volatile uint32_t RTCEN     : 1;
    volatile uint32_t           : 1;
    volatile uint32_t LPDBCEN0  : 1;
    volatile uint32_t LPDBCEN1  : 1;
    volatile uint32_t LPDBCCLR  : 1;
    volatile uint32_t           : 4;
    volatile uint32_t WKUPDBC   : 3;
    volatile uint32_t           : 1;
    volatile uint32_t LPDBC     : 3;
    volatile uint32_t           : 13;
} SUPC_WUMR_bits;

```

```c
// Bit field struct for the SUPC_WUIR register
typedef struct {
    volatile uint32_t WKUPEN0   : 1;
    volatile uint32_t WKUPEN1   : 1;
    volatile uint32_t WKUPEN2   : 1;
    volatile uint32_t WKUPEN3   : 1;
    volatile uint32_t WKUPEN4   : 1;
    volatile uint32_t WKUPEN5   : 1;
    volatile uint32_t WKUPEN6   : 1;
    volatile uint32_t WKUPEN7   : 1;
    volatile uint32_t WKUPEN8   : 1;
    volatile uint32_t WKUPEN9   : 1;
    volatile uint32_t WKUPEN10  : 1;
    volatile uint32_t WKUPEN11  : 1;
    volatile uint32_t WKUPEN12  : 1;
    volatile uint32_t WKUPEN13  : 1;
    volatile uint32_t WKUPEN14  : 1;
    volatile uint32_t WKUPEN15  : 1;
    volatile uint32_t WKUPT0    : 1;
    volatile uint32_t WKUPT1    : 1;
    volatile uint32_t WKUPT2    : 1;
    volatile uint32_t WKUPT3    : 1;
    volatile uint32_t WKUPT4    : 1;
    volatile uint32_t WKUPT5    : 1;
    volatile uint32_t WKUPT6    : 1;
    volatile uint32_t WKUPT7    : 1;
    volatile uint32_t WKUPT8    : 1;
    volatile uint32_t WKUPT9    : 1;
    volatile uint32_t WKUPT10   : 1;
    volatile uint32_t WKUPT11   : 1;
    volatile uint32_t WKUPT12   : 1;
    volatile uint32_t WKUPT13   : 1;
    volatile uint32_t WKUPT14   : 1;
    volatile uint32_t WKUPT15   : 1;
} SUPC_WUIR_bits;

// Bit field struct for the SUPC_SR register
typedef struct {
    volatile uint32_t           : 1;
    volatile uint32_t WKUPS     : 1;
    volatile uint32_t SMWS      : 1;
    volatile uint32_t BODRSTS   : 1;
    volatile uint32_t SMRSTS    : 1;
    volatile uint32_t SMS       : 1;
    volatile uint32_t SMOS      : 1;
    volatile uint32_t OSCSEL    : 1;
    volatile uint32_t           : 5;
    volatile uint32_t LPDBCS0   : 1;
    volatile uint32_t LPDBCS1   : 1;
    volatile uint32_t           : 1;
    volatile uint32_t WKUPIS0   : 1;
    volatile uint32_t WKUPIS1   : 1;
    volatile uint32_t WKUPIS2   : 1;
    volatile uint32_t WKUPIS3   : 1;
    volatile uint32_t WKUPIS4   : 1;
    volatile uint32_t WKUPIS5   : 1;
    volatile uint32_t WKUPIS6   : 1;
    volatile uint32_t WKUPIS7   : 1;
    volatile uint32_t WKUPIS8   : 1;
    volatile uint32_t WKUPIS9   : 1;
    volatile uint32_t WKUPIS10  : 1;
    volatile uint32_t WKUPIS11  : 1;
    volatile uint32_t WKUPIS12  : 1;
    volatile uint32_t WKUPIS13  : 1;
    volatile uint32_t WKUPIS14  : 1;
    volatile uint32_t WKUPIS15  : 1;
} SUPC_SR_bits;

// Peripheral struct for a PMC peripheral
typedef struct {
    volatile SUPC_CR_bits   SUPC_CR;        // (Supc Offset: 0x0000) Supply Controller Control Register
    volatile SUPC_SMMR_bits SUPC_SMMR;      // (Supc Offset: 0x0004) Supply Controller Supply Monitor
```

```
Mode Register
    volatile SUPC_MR_bits   SUPC_MR;       // (Supc Offset: 0x0008) Supply Controller Mode Register
    volatile SUPC_WUMR_bits SUPC_WUMR;     // (Supc Offset: 0x000C) Supply Controller Wake-up Mode Register
    volatile SUPC_WUIR_bits SUPC_WUIR;     // (Supc Offset: 0x0010) Supply Controller Wake-up Inputs Register
    volatile SUPC_SR_bits   SUPC_SR;       // (Supc Offset: 0x0014) Supply Controller Status Register
    volatile uint32_t       Reserved1;
} Supc;

// Pointer to a Supc-sized chunk of memory at the SUPC peripheral
#define SUPC ((Supc *) SUPC_BASE)

#endif

```

```c
/* SAM4S4B_pmc.h
 *
 * atrikha@hmc.edu
 * 20th November, 2019
 *
 * Contains base address locations, register structs, definitions, and functions for the SYSC_WPMR
 * peripheral (System Controller Write Protection Mode Register) of the SAM4S4B microcontroller. */

#ifndef SAM4S4B_SYSC_WPMR_H
#define SAM4S4B_SYSC_WPMR_H

#include <stdint.h>

////////////////////////////////////////////////////////////////////////////////////////////////////
// SYSC Base Address Definition
////////////////////////////////////////////////////////////////////////////////////////////////////

#define SYSC_WPMR_BASE   (0x400E14E4) // SYSC Base Address

// Bit field struct for the SYSC_WPMR
 typedef struct {
     volatile uint32_t WPEN      : 1;
     volatile uint32_t           : 7;
     volatile uint32_t WPKEY     : 24;
 } Sysc_wpmr;

 // Pointer to a Sysc-sized chunk of memory at the SYSC peripheral
#define SYSC_WPMR ((Sysc_wpmr *) SYSC_WPMR_BASE)

#endif

```