

```

#include <EEPROM.h>

#define SoftVer 2 // Software Version 1 = 6 settings, 2 = 18 settings
#define SpeedToggle A4 // Defining Speed Toggle Switch Address as (A4)
#define TimeToggle A3 // Defining Time Toggle Switch Address as (A3)
#define ACenablePin 6 // Output AC Enable SCR Pin
#define ACpulseTime 500 // Amount of time to fire Trigger on SCR
#define AnalogPin0 A0 // Analog A0 for Frequency Potentiometer input
#define AnalogPin1 A1 // Analog A1 for Frequency Potentiometer output
#define CycleLED 13 // Define Digital Pin 5 for output Cycle LED
#define IntPin 2 // Interrupt pin being used
#define TrigPin 8 // Output Capacitor Charge Trigger Pin
#define PulseDelay 250 // Pulse delay after zero crossing
#define StartDelay 1500 // Startup Delay in milliseconds
#define ZeroCrossPin 4 // ***** Input Zero Crossing Pin *****

int BAUD=9600; // Serial Port Baud Rate
int eeAddress = 0; // EEPROM initial starting address
int FireSwitch = 0; // Holds Toggle Switch setting for Address Line (SpeedToggle)
int IntFlag=0; // Interrupt flag initialized to zero
int PgmNo; // Initialize Program Number
int PulseCnt=0; // Pulse Count
int PulseCntHold=0; // Pulse Count Hold field
int RandFlag; // Random Frequency Flag
int RandNumber; // Random Number Field
int RunPgm=0; // Run Program Switch 0 = Stop -1 = Run
int temp; // temp of general purpose work & testing
int temper1; // temp of general purpose work & testing
int temper2; // temp of general purpose work & testing
int Time; // use for Time Setting 1,2,5,10,20 or 30 minutes
int TimeSwitch; // Holds Togle Swithc setting for Address Line (A?)
unsigned long EndTime; // Used for timer control
unsigned long EndTime2; // Used for timer control
unsigned long RunTime; // Used for timer control
unsigned long PerSecCnt=1; // Time Delay to control the number of pulses per second
unsigned long Waste; // Waste time field for wasting time
struct EPromObject {
    int field1;

```

```

int field2;
int field3;
int field4;
int field5;
};

void setup()
{
  noInterrupts(); // Critical Timing disable Interrupt
  pinMode(ACenablePin, OUTPUT); // AC enable Pin defined as Output 6
  digitalWrite(ACenablePin, LOW); // Insure AC enable Pin not triggering SCR
  pinMode(TrigPin, OUTPUT); // Trigger Capacitor Charge Pin as Output 7
  digitalWrite(TrigPin, LOW); // Insure Trigger Pin not being fired
  pinMode(ZeroCrossPin, INPUT); // Zero Crossing Pin defined as Input 4
  digitalWrite(ZeroCrossPin, HIGH); // Enable pullup resistor 4
  pinMode(IntPin, INPUT); // Assign the Interrupt pin 2
  digitalWrite(IntPin, INPUT_PULLUP); // Enable pullup resistor 2
  attachInterrupt(digitalPinToInterrupt(IntPin), breakout, RISING);
  Serial.begin(BAUD); // Set the baud rate to 9600
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  pinMode(CycleLED, OUTPUT); // D05 --- NOT USING, using D13 instead
  digitalWrite(CycleLED, LOW); // Initialize LED to off state

  RunPgm=0; // Insure Program not running
  PulseCnt=0; // Insure Pulse Counter Intialized to zero
  temper1=analogRead(A1); // Read the Thermistor 1 to determing heat temperature
  temper2=analogRead(A0); // Read the Thermistor 2 to determing heat temperature
  Settings(); // Get all the settings and check temperature
  delay(StartDelay); // Before Firing anything wait awhile
  interrupts(); // Critical Timing completed enable interrupts
  IntFlag=0; // Interupt Flag
  randomSeed(analogRead(3)); // Randomize the random number generator
}

void loop() // Main Loop

```

```

{
  if(IntFlag==true){
    while(digitalRead(IntPin)==LOW){ // Look for a LOW 1st incase button is being held down
      delay(250); // give it some time to debounce
    }
    delay(250); // give it some time to debounce
    IntFlag=false; // Turn off Interupt flag
    if(RunPgm==false){ // Flip the RunPgrm Flag
      Settings(); // Get all the settings and check temperature
      RunPgm=true; // If here set it to Run the Program
      RunTime=EndTime+millis(); // Reset the RunTime clock time counter
    }
  }
  else{
    if(PulseCnt>5){ // Interupt happened
      DisplayParamatures(); // display only if fired more than 5 times
    }
    else{
      RunPgm=false; // If here set it to Stop the Program
      PulseCnt=0; // Reset Pulse count for next run
    }
  }
  interrupts(); // Critical Timing completed enable interrupts
}

if(RunPgm==true){ // On condition RUN Fire Routine
  if(RunTime>millis()){ // RUN only if not out of time
    Fire(PerSecCnt); // If here then everything is a go to RUN
  }
  else{
    DisplayParamatures(); // If Serial Port on these will display
    if(TimeSwitch > 1){ // Check if doing pause and repeat
      RunTime=EndTime2+millis(); // Reset the RunTime clock with rest time
      while(RunTime>millis()){ // pause for as long as selected run time
        if(IntFlag!=true){
          for(Waste=0; Waste<10000; Waste++) // I know - could of used 'Delay function'
            digitalWrite(CycleLED, HIGH); // toggle the LED - ON
          for(Waste=0; Waste<10000; Waste++)
            digitalWrite(CycleLED, LOW); // toggle the LED - ON
        }
      }
    }
  }
}

```

```

        for(Waste=0; Waste<10000; Waste++)
            digitalWrite(CycleLED, HIGH);    // toggle the LED - ON
        for(Waste=0; Waste<100000; Waste++)
            digitalWrite(CycleLED, LOW);     // toggle the LED - ON
    }
    else{
        TimeSwitch=1;
        RunTime=millis();
        RunPgm=false;
        IntFlag=0;
    }
}
}
--TimeSwitch;                // if here repeat for requested times
if((TimeSwitch>0) && (IntFlag!=true)){
    RunPgm=true;
    RunTime=EndTime+millis();    // Reset the RunTime clock with Dialed Time
}
}
}
}

void DisplayParamatures(){
    RunPgm=false;                // If here set it to Stop the Program
    EPromObject customVar;      // Vairables to store custom objects from EEPROM
    EEPROM.get(eeAddress,customVar); // Fetch data from EPROM

    Serial.println("          Speed Time Pulses SCR1 SCR2"); // display header
    Serial.print("Prev Run  "); // Some leading Spaces
    Serial.print(customVar.field1); // Print the Speed Setting
    Serial.print("          "); // Some leading Spaces
    Serial.print(customVar.field2); // Print the Time in Munutes
    Serial.print("          "); // Some leading Spaces
    Serial.print(customVar.field3); // Print the Pulse Count
    Serial.print("          "); // Some leading Spaces
    Serial.print(customVar.field4); // Print the Charging SCR1 thermister value
    Serial.print("          "); // Some leading Spaces
    Serial.println(customVar.field5); // Print the Coil SCR2 thermister value
}

```

```

customVar.field1=7-PgmNo;           // Speed
customVar.field2=EndTime/60000,    // Time
customVar.field3=PulseCnt,         // Pulses
customVar.field4=analogRead(A0),    // SCR1 Thermistor
customVar.field5=analogRead(A1),    // SCR2 Thermistor

Serial.print("Last Run   ");       // Some leading Spaces
Serial.print(customVar.field1);     // Print the Speed Setting
Serial.print("         ");         // Some leading Spaces
Serial.print(customVar.field2);     // Print the Time in Munutes
Serial.print("         ");         // Some leading Spaces
Serial.print(customVar.field3);     // Print the Pulse Count
Serial.print("         ");         // Some leading Spaces
Serial.print(customVar.field4);     // Print the Charging SCR1 thermister value
Serial.print("         ");         // Some leading Spaces
Serial.println(customVar.field5);    // Print the Coil SCR2 thermister value

EEPROM.put(eeAddress,customVar);    // Put latest run into EPROM
PulseCnt=0;
}

void Fire(int){
  PulseCnt++;                       // Increment Pulse Counter
  digitalWrite(CycleLED, HIGH);     // toggle the LED - ON
  while(digitalRead(ZeroCrossPin)==LOW); // Look for a LOW 1st to avoid starting in LOW
  while(digitalRead(ZeroCrossPin)==HIGH); // NOTE: when not connected it read high 4
  delayMicroseconds(5800);          // delay period after zero crossing about 5 millisecs
  digitalWrite(ACenablePin,HIGH);   // 1. Enable AC to charge Charge Capacitor 6
  delayMicroseconds(ACpulseTime);   // AC SCR Trigger Time 100 to 500 microsecs
  digitalWrite(ACenablePin,LOW);    // AC is enabled, turn off the trigger 6

if(FireSwitch < 2){                // Original Code without FireSwitch
  delayMicroseconds(15500);         // wait till next pulse
  digitalWrite(ACenablePin,HIGH);   // 2. Enable AC to charge Charge Capacitor 6
  delayMicroseconds(ACpulseTime);   // AC SCR Trigger Time 100 to 500 microsecs
  digitalWrite(ACenablePin,LOW);    // AC is enabled, turn off the trigger 6
}
}

```

```

delayMicroseconds(15500);           // wait till next pulse
digitalWrite(ACenablePin,HIGH);     // 3. Enable AC to charge Charge Capacitor 6
delayMicroseconds(ACpulseTime);    // AC SCR Trigger Time 100 to 500 microsecs
digitalWrite(ACenablePin,LOW);      // AC is enabled, turn off the trigger 6

delayMicroseconds(15500);           // wait till next pulse
digitalWrite(ACenablePin,HIGH);     // 4. Enable AC to charge Charge Capacitor 6
delayMicroseconds(ACpulseTime);    // AC SCR Trigger Time 100 to 500 microsecs
digitalWrite(ACenablePin,LOW);      // AC is enabled, turn off the trigger 6

delayMicroseconds(15500);           // wait till next pulse
digitalWrite(ACenablePin,HIGH);     // 5. Enable AC to charge Charge Capacitor 6
delayMicroseconds(ACpulseTime);    // AC SCR Trigger Time 100 to 500 microsecs
digitalWrite(ACenablePin,LOW);      // AC is enabled, turn off the trigger 6
}

if(FireSwitch == 2){                // Additional Code Options 2
  if(PgmNo > 4){
    delayMicroseconds(15500);       // wait till next pulse
    digitalWrite(ACenablePin,HIGH); // 5. Enable AC to charge Charge Capacitor 6
    delayMicroseconds(ACpulseTime); // AC SCR Trigger Time 100 to 500 microsecs
    digitalWrite(ACenablePin,LOW);  // AC is enabled, turn off the trigger 6
  }
  if(PgmNo > 3){
    delayMicroseconds(15500);       // wait till next pulse
    digitalWrite(ACenablePin,HIGH); // 5. Enable AC to charge Charge Capacitor 6
    delayMicroseconds(ACpulseTime); // AC SCR Trigger Time 100 to 500 microsecs
    digitalWrite(ACenablePin,LOW);  // AC is enabled, turn off the trigger 6
  }
  if(PgmNo > 2){
    delayMicroseconds(15500);       // wait till next pulse
    digitalWrite(ACenablePin,HIGH); // 2. Enable AC to charge Charge Capacitor 6
    delayMicroseconds(ACpulseTime); // AC SCR Trigger Time 100 to 500 microsecs
    digitalWrite(ACenablePin,LOW);  // AC is enabled, turn off the trigger 6
  }
  if(PgmNo > 1){
    delayMicroseconds(15500);       // wait till next pulse
    digitalWrite(ACenablePin,HIGH); // 3. Enable AC to charge Charge Capacitor 6
  }
}

```

```

delayMicroseconds(ACpulseTime); // AC SCR Trigger Time 100 to 500 microsecs
digitalWrite(ACenablePin,LOW); // AC is enabled, turn off the trigger 6
}
}

digitalWrite(CycleLED, LOW); // toggle the LED - OFF
delayMicroseconds(11200); // Give enough time to insure AC is off
delayMicroseconds(11200); // Give enough time to insure AC is off
delayMicroseconds(11200); // Give enough time to insure AC is off
digitalWrite(TrigPin, HIGH); // Trigger SCR to Discharge Cap trough the coil 7
delayMicroseconds(ACpulseTime); // defined a pulse width time
digitalWrite(TrigPin,LOW); // Turn off Discharge pulse 7
delay(16); // Wait a real long time

if (725 > analogRead(A1)){ // Read the Thermistor to determing heat temperature
  RunTime=millis(); // This terminates RUN by forcing TIME OUT in test above
  digitalWrite(CycleLED, HIGH); // toggle the LED - ON because of over heating
}

if (725 > analogRead(A0)){ // Read the Thermistor to determing heat temperature
  RunTime=millis(); // This terminates RUN by forcing TIME OUT in test above
  digitalWrite(CycleLED, HIGH); // toggle the LED - ON because of over heating
}

if (RandFlag==1){ // If Random Flag on change the pulse rate
  RandNumber=random(1,18); // crazy way of selecting different pulse rate
  PerSecCnt=360/RandNumber; // Per Second Cnt controls pulse rate
}

delay(PerSecCnt); // run delay to control pulses per second
}

void Settings()
{
  RandFlag=0; // Turn off Random Flag
  PgmNo = analogRead(A7)/178+1; // read the Potentiometer value to determing speed
  FireSwitch = 0; // Default to Original Random & 5 Mode Capacitor charges
  if(SoftVer == 2){ // Do only if this Software Version 2

```

```

    temp = analogRead(SpeedToggle);           // Address (A?) to determining which RUN TIME is set
    if(temp > 100)                             // > then 100 means switch is set to position 1 or 2
        FireSwitch = 1;                       // load for switch position 1 speed settings
    if(temp > 1000)                             // > then 1000 means switch is set to position 2
        FireSwitch = 2;                       // load for switch position 2 speed settings
}

Serial.println(FireSwitch);    // Display Toggle Switch

if(FireSwitch == 0){          // Frequency Options 0
    switch(PgmNo){
        case 6:
            RandFlag=1;      // if here Rrandom Flag is activated.
            break;
        case 5:
            PerSecCnt=865;   // Delay Count to generate a 1.0 hertz rate
            break;
        case 4:
            PerSecCnt=370;   // Delay Count to generate a 2.0 hertz rate
            break;
        case 3:
            PerSecCnt=205;   // Delay Count to generate a 3.0 hertz rate
            break;
        case 2:
            PerSecCnt=115;   // Delay Count to generate a 4.0 hertz rate
            break;
        case 1:
            PerSecCnt=75;    // Delay Count to generate a 5.0 hertz rate
            break;
    }
}

if(FireSwitch == 1){          // Frequency Options 1
    switch(PgmNo){
        case 6:
            PerSecCnt=525;   // Delay Count to generate a 1.53 hertz rate
            break;
        case 5:

```



```

    PerSecCnt=300;           // Delay Count to generate a 2.31 hertz rate
    break;
case 4:
    PerSecCnt=150;         // Delay Count to generate a 3.68 hertz rate
    break;
case 3:
    PerSecCnt=95;          // Delay Count to generate a 4.63 hertz rate
    break;
case 2:
    PerSecCnt=55;          // Delay Count to generate a 5.43 hertz rate
    break;
case 1:
    PerSecCnt=35;          // Delay Count to generate a 6.0 hertz rate
    break;
}
}

if(FireSwitch == 2){       // Frequency Options 2
switch(PgmNo){
case 6:
    PerSecCnt=20;         // Delay Count to generate a 6.67 hertz rate
    break;
case 5:
    PerSecCnt=1;          // Delay Count to generate a 7.57 hertz rate
    break;
case 4:
    PerSecCnt=1;          // Delay Count to generate a 8.62 hertz rate
    break;
case 3:
    PerSecCnt=1;          // Delay Count to generate a 10.00 hertz rate
    break;
case 2:
    PerSecCnt=1;          // Delay Count to generate a 12.0 hertz rate
    break;
case 1:
    PerSecCnt=1;          // Delay Count to generate a 14.7 hertz rate
    break;
}
}

```

```

}

Time = (analogRead(A5)/178+1);           // read the Potentiometer value to determining run time
TimeSwitch = 0;                          // Preload TimeSwitch for setting up Software Version 1
if(SoftVer == 2){                         // check if actually Software Version 2
    temp = analogRead(TimeToggle);        // if here Address (A?) determines which RUN TIME is set

    if(temp > 100)                        // > then 100 means switch is in position 1
        TimeSwitch = 4;                   // load to do RUN/REPEAT 4 times
    if(temp > 1000)                       // > then 1000 means switch is in Position 2
        TimeSwitch = 8;                   // load to do RUN/REPEAT 8 times
}
Serial.println(TimeSwitch); // Display Toggle Switch 2

switch(Time){
    case 6:
        EndTime=60000;                    // Set to Run for 1 minute
        EndTime2=600000;                  // Rest Time set for 10 minutes
        break;
    case 5:
        EndTime=120000;                   // Set to Run for 2 minutes
        EndTime2=600000;                  // Rest Time set for 10 minutes
        break;
    case 4:
        EndTime=300000;                   // Set to Run for 5 minutes
        EndTime2=900000;                  // Rest Time set for 15 minute
        break;
    case 3:
        EndTime=600000;                   // Set to Run for 10 minute
        EndTime2=1200000;                 // Rest Time Set to 20
        break;
    case 2:
        EndTime=1200000;                  // Set to Run for 20 minute
        EndTime2=4800000;                 // set to Run for 40 minutes
        break;
    case 1:
        EndTime=1800000;                  // Set to Run for 30 minute

```

```
        EndTime2=3600000;           // set to Run for 60 minutes
        break;
    }
}

void breakout()
{
    noInterrupts();                // Breakout interrupt routine
    IntFlag=true;                  // Critical Timing disable Interrupt
}
// Turn on Interupt flag
```