EJEMPLO: CONTROL BASICO DEL ADE7753.

Generalidades: El ADE7753 es un IC medidor de energía que puede ser utilizado para obtener información sobre valores RMS de las señales de corriente y voltaje y valores de potencia y energía.

Componentes necesarios:

* Un arduino.
* Un hardware controlador del ADE7753, (shield del medidor para el arduino).
* Pinza de corriente 30.
* Transformador 120/12 V.

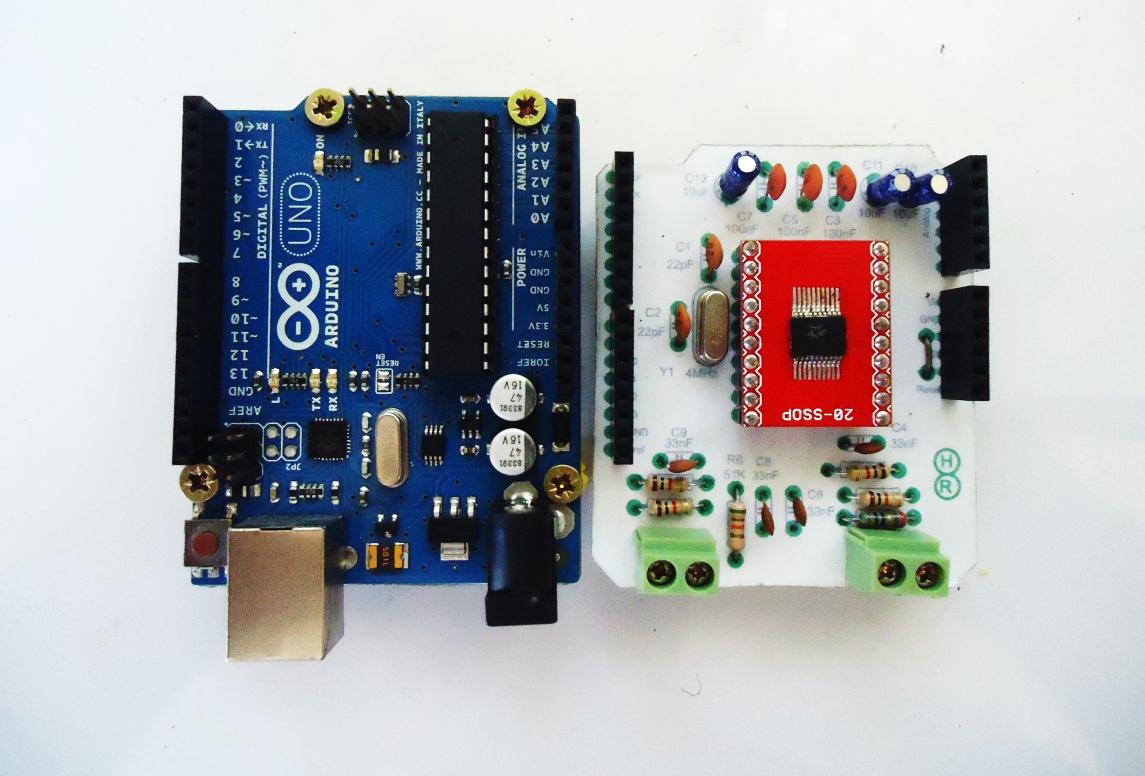


Figura 1. Arduino y la shield del medidor para arduino usando el ADE7753.

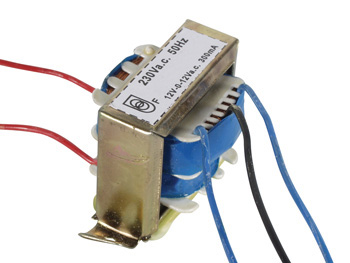


Figura 2. Pinza de corriente y transformador para censar voltaje y corriente.

## ADE7753 INTERFACE SERIAL.

Todas las funcionalidades del ADE7753 son accesibles a través de muchos registros que ya están dentro del chip, ver figura 13. Los contenidos de estos registros pueden ser actualizados o leídos usando la interface serial del chip. Después de encender, mandar a RESET un pulso bajo o en el borde de bajada de El ADE7753 es puesto en modo de comunicación; en este modo el IC espera que se escriba en su registro de comunicación. El dato escrito en el registro de comunicación determina si la siguiente operación es una lectura o una escritura y además indica la dirección del registro a acceder. Entonces todas las transferencias de datos no importando si es lectura o escritura deben iniciar con una escritura en el registro de comunicaciones.

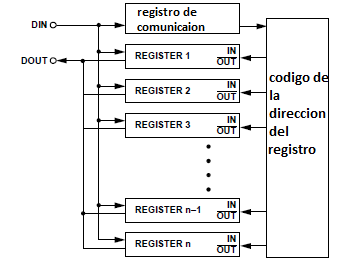


Figura 3. Direccionando los registros del ADE7753 vía registro de comunicación.

Para la conexión se debe tener en cuenta solamente que la pinza de corriente debe de llegar donde se encuentra la resistencia de 10 Ω dentro de la shield.

Configuración básica del ADE7753.

//declarando librerias a utilizar

#include <SPI.h>

//declarando alcunas constantes principales

const int cs=9; //debe ser este

const byte leer=0b00000000;

const byte escribir=0b10000000;

En la primera parte del código se inicia la librería necesaria para trabajar con el integrado ñuego se inicializan unas constantes como el selector de chip, y la constante principal que sirve para declarar el tipo de operación a realizar (di de leer o de escribir).

void setup(){

Serial.begin(9600);

//configuracion estandar de la comunicacion SPI del IC.

SPI.setDataMode(SPI\_MODE2);

SPI.setClockDivider(SPI\_CLOCK\_DIV32);

SPI.setBitOrder(MSBFIRST);

SPI.begin();

pinMode(9, OUTPUT);//pin de integrado

iniciarconstantes();

csalto();

delay(10);

}

En la última porción de código se inicia configurando el baud rate (velocidad de comunicación serial pines TX y RX) luego se debe trabajar en el MODO2 y con MSBFIRST, según especificaciones de la hoja de datos, mientras que la frecuencia de reloj con que trabajara la comunicación arduino integrado la podemos dejar con un DIV32.

Se declara coma salida el selector de chip y luego se llama a una función que sirve para asignar y declarar los registros, esto tiene como única finalidad hacer entendible el código y en lugar de poner números, poner el nombre al registro que se hace referencia.

<csalto> es una pequeña función junto con <csbajo>, útil cuando se trabaja con más dispositivos conectados a los mismos pines.

void csalto(void){

digitalWrite(cs,HIGH);

}

void cssdbajo(void){

digitalWrite(cssd,LOW);

}

En la siguiente porción de código vemos las rutinas de lectura y escritura básicas para registros de tamaño de 8 bits, en un pequeño esbozo de estas se puede decir que lo primero que se hace es capturar en el mismo byte la dirección del registro que debe ser un parámetro a enviar dentro de la función y además asignarle los primeros dos bits de identificación del tipo de operación (lectura o escritura) es por eso que se hace la suma lógica con reg OR leer, reg OR escribir. Además cuando ya se está listo con la información a transferir se selecciona al integrado con un csbajo (activo con un bajo) se esperan 5 microsegundos y se utiliza el comando <SPI.transfer> este es el que manda haciendo uso del protocolo SPI el byte correspondiente indicado dentro de sus parámetros. Después viene una espera de 5 microsegundos (se escoge un sol tiempo para no confundirse de los tiempos de la hoja de datos, el mayor tiempo con algo de sobra), la siguiente línea es para capturar y retornar el valor contenido en el registro del integrado. Luego se desactiva el chip.

unsigned char leer8(char reg){

reg = reg | leer;

csbajo();

delayMicroseconds(5);

SPI.transfer(reg);

delayMicroseconds(5);

return (unsigned char)SPI.transfer(0x00);//cambiar por unsigned byte

csalto();

}

Al igual que la porción de código anterior, la mayoría de comandos se repiten excepto porque ahora es una escritura y justo después de realizar la primera transferencia obligatoria, debemos seguir enviando los bytes que queremos separados por tiempos de 5 microsegundos.

void escribir8(char reg, char data){

reg = reg | escribir;

csbajo();

delayMicroseconds(10);

SPI.transfer((unsigned char)reg); //register selection

delayMicroseconds(5);

SPI.transfer((unsigned char)data);

delayMicroseconds(5);

csalto();

}

Con todo esto listo podemos pasar a hacer algo útil con el integrado, esto es obtener valores RMS de corriente y voltaje censados.

Se explica solamente el código para obtener la corriente, y se inicia describiendo las funciones que invoca esta función.

int getStatus(void){

return leer16(STATUS);

}

La función anterior es por decirlo así de segundo nivel pues ya utiliza las funciones leer16 que invocan toda la operación de lectura descrita anteriormente.

Además la forma en que se definen variables en las constantes es la siguiente:

#define ZX 0x0010 // bit 4 -

Las variables <lastupdate> <contando> y <salirse> solo sirven para evitar atascos por largos tiempos pero para propósitos de funcionamiento ideal debemos analizar el código sin tomarlos en cuenta.

La línea while (! (getStatus() & ZX )) tiene como objetivo sincronizar la operación de lectura con la frecuencia de la señal, ZX es uno en los ciclos positivos de la señal de voltaje, por lo tanto, getStatus() tiene como objetivo verificar en que ciclo esta la señal y dependiendo de eso permite salir de <while> cuya única restricción es esta. Haciendo esto logramos que los cálculos de RMS sean de acuerdo a la frecuencia de la señal, sino obtuviéramos cantidades innecesarias de lecturas repetidas pues el cálculo que el integrado realiza ni siquiera se ha llevado acabo porque la señal es lenta.

//obtener valor IRMS

long getIRMS(void){

long lastupdate=0;

resetStatus();

lastupdate=millis();

while (! (getStatus() & ZX ))

long contando=millis();

long salirse=contando-lastupdate;

if ( salirse > 200) break;

}

return leer24(IRMS);

}

//obtener valor VRMS

long getVRMS(void){

long lastupdate=0;

resetStatus();

lastupdate=millis();

while (! (getStatus() & ZX ))

{//Serial.println("encerrado en vrms");

long contando=millis();

long salirse=contando-lastupdate;

if ( salirse > 200) break;

}

return leer24(VRMS);

}

//LA SIGUIENTE FUNCION

//IGNORA LAS PRIMERAS GETRMS Y SACA EL PROMEDIODE 50 LECTURAS

long vrms(){

char i=0;

long v=0;

getVRMS();//ignora la primera lectura para evitar basura

getVRMS();

getVRMS();

for(i=0;i<50;++i){

v+=getVRMS();

}

return v/50;

}

//ahora para la corriente

long irms(){

char n=0;

long i=0;

getIRMS();//ignora la primera lectura para evitar basura

getIRMS();

getIRMS();

for(n=0;n<50;++n){

i+=getIRMS();

}

return i/50;

}

La anterior porción de código es la definitiva que sirve, para empezar ignora las primeras lecturas y luego toma 50 lectura todo esto tiene lugar en ciclo cuando la señal está en la parte negativa, por lo tanto al descartar y sacar un promedio de unas cuantas lecturas lo que se quiere lograr es la fiabilidad de los datos.

Las únicas líneas que deben ir en loop principal son:

long voltaje\_rms=vrms();

float voltaje\_rms\_corregido= voltaje\_rms\****0.000264585***;

//Serial.println(voltaje\_rms);// solo para calibrar, intercambiar según se explica

//Serial.println(voltaje\_rms\_corregido);//intercambiar según se necesite

Las tercera línea de la porción de código anterior es la que debemos des-comentar la primera vez que lo pongamos en funcionamiento, esto nos arrojara un entero que corresponde al valor binario guardado en el registro.

Para este caso no hay que tener un especial cuidado con los signos, puesto que los registros RMS son siempre positivos.

Si se desea mostrar en consola un valor real de la medición, se aplica una simple regla de tres, con un valor conocido de voltaje o corriente vemos cuanto es el valor entero del registro y ahora aplicamos la regla de tres para encontrar la constante por la cual debemos multiplicar el registro y obtener el valor de acuerdo a lo que estamos midiendo.