6.1 Software Main

```
/*
 * main.c
* Author: Ignacio Carlucho
*/

#include <msp430.h>
#include "global.h"
#include "functions.h"
#include <stdio.h>
/*
 * main.c
*/

                // flags
volatile int configuration=0x00;                    // configuration = 0x00 me dice que no debo realizar la comprobacion de lectura,
//porque los integrados no estan programados
// configuracion = 0x01 me dice que los integrados estan configurados y puedo hacer la comprobacion de escritura.
volatile char charging_state=0x00; // if 0x00 not charging.. if battery pack 0x01 is charging
volatile char cell_balance=0x00;     // activa y desactiva el balanceo de las celdas
                                                // el balanceo esta activo cuando cell_balance=0xaa;
                                                // Configuraciones de cantidad de celdas x integrado.
volatile char cant_celdas=0x06;
 volatile char cant_integrado=0x01;   // es una constante global que me determina la cantidad de integrados
 volatile char matriz_celda [1][6];  // matriz de celdas, [cantidad de integrados][número de celdas x integrado]
                                        //cant_integrado = 1 significa 1 integrado .. la primer fila de la matriz no se usa
                                        //numero de celdas
   volatile char last_voltage[2]; // el numero entre corchetes tiene que ser dos veces la cantidad de integrados cant_integrados..
   volatile char celdas_xintegrado[4]={0x00,0x06,0x04,0x06};           // aca se ingresa el numero de celdas por integrado.
                                                // para hacer la configuracion. porque despues voy a poner celdas_xintegrado(n)
                                                //y voy a tener un algoritmo que me configure el registro 30
                                                // el primer valor se deja en cero
   volatile long count_up;
               volatile long ciclos_ref;
               volatile char ecualization_form; // 0x01 , solo ecualiza durante la carga, 0x11, ecualiza en reposo y en la carga.
   // variables
   volatile char error_count=0x00;
   volatile char crc_error=0x00;
   volatile char error_lectura_crc=0x00;
   volatile char read_state=0x00;
////////////////////////////////////////////////// MAIN ////////////////////////////////////////////////////////
int main(void)
{
               WDTCTL = WDTPW | WDTHOLD;                        // Stop watchdog timer
               //P1OUT = 0x00;
               P1DIR |= 0x01;                                                   // Set P1.0 to output direction
               P2DIR |= 0x01;                                                   // set P2.0 to output direction
                        // variables para la transmision de datos
               volatile char MLSB1,MLSB2,MLSB3,MLSB4,MLSB5,MLSB6;
               volatile char MSB1,MSB2,MSB3,MSB4,MSB5,MSB6;
               volatile char LSB1,LSB2,LSB3,LSB4,LSB5,LSB6;
               volatile char integrado;
               volatile char balancing;
               volatile char stock;
               volatile char gross=0x00;
               volatile long long A=0x00;
               volatile char i,m;
               volatile char fucking[6];
               volatile float voltage_prueba1,voltage_prueba2,voltage_prueba3,voltage_prueba4,voltage_prueba5,voltage_prueba6;
               volatile char test1,test,test2,test3,test4,test5;
               volatile long pruebita,pruebita2,pruebita3,pruebita4,pruebita5;
               volatile char  MSB,LSB, isum;
               volatile char debugger=0x01;
               char string[11];
   volatile unsigned char lectura_estadoa, lectura_nombre;
   volatile char lectura_estadob,lectura_estadod ;
   volatile char lectura_estadoe,lectura_estadof ;
   volatile char lectura_estadoc ;
   volatile char power_on_status;  // variable que almacena el estatus del power on, = 0 fallido, 1 correcto.
   volatile char n, desborde;
   volatile char para_pruebas, debug;   // variables para pruebas dy debugeo
   volatile char para_pruebas_1, debug_1;
   volatile int cuenta_int=0x00;
   volatile char temperature1_h, temperature1_l,temperature2_h, temperature2_l;
   lectura_estadob=0x00;
   lectura_estadoa=0x00;
```

```
   lectura_estadoc=0x00;
   P2OUT = 0x00;



////////////////////////////  Comunicación SPI//////////////////////////
            UCB0CTL1 |=  UCSWRST;  //reseteo antes de configurar
            UCB0CTL1 |=  UCSSEL_2 ; // Control register 1 . SMCLK
            //*********************************************//
            // Modificacion!!!                            //
            UCB0CTL0 |= UCMSB+UCMST+UCSYNC+UCCKPL ;          // Control register 0 // now clock is active high
            //*********************************************//
            UCB0BR0  |= 0x04;
            UCB0BR1 = 0;
   //UCB0STAT |= UCLISTEN ;
            P1SEL = BIT1 + BIT2 + BIT5 + BIT6 + BIT7;
            P1SEL2 = BIT1 + BIT2 +BIT5 + BIT6 + BIT7;
   ////////////////////////////  Para comunicarme con la terminal
    //P1SEL = BIT1 + BIT2 ;                  // P1.1 = RXD, P1.2=TXD
    //P1SEL2 = BIT1 + BIT2 ;                 // P1.1 = RXD, P1.2=TXD
    UCA0CTL1 |= UCSSEL_2;                    // SMCLK
    UCA0BR0 = 104;                          // 1MHz 9600
    UCA0BR1 = 0;                           // 1MHz 9600
    UCA0MCTL = UCBRS0;                       // Modulation UCBRSx = 1
    UCA0CTL1 &= ~UCSWRST;                    // **Initialize USCI state machine**
    //IE2 |= UCA0RXIE;                      // Enable USCI_A0 RX interrupt
    //TACCR0  |=;
    //IE2 |= UCB0RXIE + UCB0TXIE;                    // Enable USCI0 RX y TX interrupt
    // UCB0TXBUF Para escribir
    // UCB0RXBUF Para leer.
             UCB0CTL1 &= ~UCSWRST;    // saca el reset
             //escribir(0x01,0x34,0x01);
             //_delay_cycles(1000);
             debugger=0x01;
escribir(0x01,0x33,0x0a);  // cuando se activa lo hace por 10 segundos
            /*****************************************************************/
            /*                                    Interrupciones
            */
            // Cuando tenga el prototipo bien armado voy a realizar interrupciones con las patas de
            //                          ** fault       P2.4
            //                          ** Alert       P2.3
            //
            //                                    Ahora (2553) las estoy implementando en estas patas porque son las que tengo disponibles
            //                                    Luego (247) pueden (o deben) variar
            //
            // Seguramente estos códigos, o por lo menos la habilitación de las instrucciones, deberá ir más abajo
            // porque no debería habilitar las interrupciones hasta lo que los integrados estén propiamente configurados
            // Esto está claro, entonces la habilitación de la interrupción y el limpiado del flag , irán luego de la primer rutina de
            // Lectura de estado de los integrados
            //
            // En que consistirá la interrupción?
            // Fácil : si es alert : se hace lo mismo que en la lectura de estado, se buscara cual es la causa.
            //             se anotara en la tabla, o se tomaran medidas de acuerdo a lo que corresponda
            //                          si es fault: idem

/*
            P2IE |= BIT4 | BIT3;                          // habilita la interrupción del P2.4 Y P2.3
            P1IES |= BIT5;                                // porque la transición del flag es cuando pasa de high a low.
            P1REN |= BIT5;                                                  // habilita la resistencia de pullup/pulldown
            P1IFG &= ~BIT5;                                                 // limpio el flag
//                  _BIS_SR(GIE);                                           // habilita las interrupciones no enmascarables

            */

            P1IE |= BIT3;
            P1IES |= BIT3;
            P1IFG &= ~BIT3;
            CCTL0 = CCIE;                  // CCR0 interrupt enabled
   CCR0 = 50000;
            TACTL = TASSEL_2 + MC_1 + ID_3 ;
                  debug_1  = 0x01;
            para_pruebas_1 = 0x01;
            para_pruebas_1 = 0x00;
                  /*****************************************************************/
                  /*                                    Rutina Power On
                  */
                  // Rutina Power ON! .
                  // Esto es medio un desastre, porque si el power on no funciona,
```

```
                        // Mediante unos leds doy aviso de esto, y re intento infinitas veces!!!!!!!!!!!!
                        // El power on debe funcionar si o si!!
                        // _BIS_SR( GIE);
            do{
                          power_on_status = power_on();    // Con esta subrutina asigno adrress a los integrados, regresa 1 si la asignacion fue
correcta, 0 si incorrecta.
                P1OUT ^=0x01;
              }while(power_on_status==0x00);

                        /*****************************************************************/
                        /*                                        Reset de los latchs
            */
            //Reset de los latchs de alert y fault
            // posiblemente esto tenga que ponerlo en la subrutina power on.
            // utilizo la variable intermedia n para contar
            desborde=cant_integrado+0x01;     // me indica la cantidad de vueltas q debo realizar, osea una mas q la cant, de integrados
            n=0x01;                     // inicio la cuenta, osea arranco con el integrado n=1, y lo hago hasta cant de integrados
            do{
              /* Reset latch de Alert*/
                escribir(n,0x20,0x80);    // Primera parte de la Rutina necesaria para borrar el latch de alerta, del registro alert.
                escribir(n,0x20,0x00);    // Segunda parte de la rutina. COmo veras tengo q escribir dos veces, primero un 1 en el bit 7,  y
despues un cero.
               /* Reset Latchs Fault. */
               /* no estoy seguro de la necesidad de hacer un doble write para limpiar el flag de fault
               pero mas vale prevenir que curar*/
                escribir(n,0x21,0x08);
                escribir(n,0x21,0x00);
                n++;      // aumento la cuenta
             }while(n<desborde);              // fin del ciclo cuando N=desborde
  /*Configuracion de los registros*/
     conf_int();
     configuration=0x01;
//luego de la rutina de configuracion yo deberia checkear el bit 3, del registro 00,
//que me indica si hay algun error de un bit en la eeprom del device,
//q yo acabo de escribir.->
//en realidad lo deberia hacer en los tres integrados.
     /*Lecura de estado */
////////////////////////////// Lectura del Estado ////////////////////////
                    // variable que sirve para recorrer el arreglo de errores
                   // error es un  arreglo de caracteres char, que me sirve para guardar los errores codificados.
                    // la codificacion de errores es de la siguiente manera.
                    // 2 bits mas significativos corresponden al número de integrado
                    // 3 bits siguiente corresponden al número de celda
                    // y los ultimos tres corresponden al código del error
                    // codigos ---> overvolta = 0b000
                    // undervotlaje = 0b001;
                    // overtemperature= 0b010
                    // undertemperature=
                    // ultra-low voltaje 0b100

lectura_estado(); // lee el estado y almacena en la matriz celda. realizando correciones en el caso de ser necesario.
ciclos_ref=0x12C;
ecualization_form=0x01; // 0x01 , solo ecualiza durante la carga, 0x11, ecualiza en reposo y en la carga.
debug=0x00;
// Esta funcion tiene que estar si o si ejecutada para cada integrado que haya, antes de ejecutar el query_chargin
query_tension(0x01); // leo el primer estado. para compara luego si esta cargando o no.
count_up=0xF0;            // pongo a la cuenta en un valor alto, para que la siguiente lectura sea a los 2 minutos..
////////////////////////// Ejecucion continua//////////////////////
while(1)
{
   lectura_estado(); // lee el estado y almacena en la matriz celda. realizando correciones en el caso de ser necesario.
   if(count_up==ciclos_ref)   // cuando llega al  valor de ciclos_ref  (10 minutos), vuelve a comparar.
   {
      query_chargin(0x01); // se pregunta si esta esta cargando, si es asi, pone el estado en carga
   }                        // y reinicia la cuenta.
   count_up++;
   if( charging_state==0x01)
   {
      balance_capture(0x01);  // si esta cargando hace las mediciones de voltaje, a la par que balancea. Y al final transmite
   }
   if( charging_state=0x00)
     {
        capture_data();      // Si no esta cargando. solo mide, y transmite.
     }
   _delay_cycles(2020000); // delay, 2 segundos.
}
}
```

6.2 Funciones de Software

```
#include <msp430.h>
#include "global.h"
#include "functions.h"



// esta funcion tiene que estar puesta si o si antes de usar el query_chargin

void query_tension(char integrado)
{
                last_voltage[0]=leer(integrado,0x03);
                last_voltage[1]=leer(integrado,0x04);

return;
}

// esta funcion se encarga de checkear si las baterias se estan cargando.
// cambia el flag charging_state y lo pone en uno
// solo lee la primer celda de cada integrado, que es descriptiva del funcionamiento del resto.
void query_chargin(char integrado)
{
volatile char MSB;
volatile char LSB;

   MSB=leer(integrado,0x03);
   LSB=leer(integrado,0x04);  // lee los voltajes de la primera celda.


if  (ecualization_form==0x01) // solo ecualiza durante la carga
{
  if (MSB >last_voltage[0]){  // si el valor actual de tension es mayor que el anterior. está cargando
       charging_state=0x01;  // activa la ecualizacion.
       last_voltage[0]=MSB;    // y almacena el nuevo valor.
   }

   if (last_voltage[0]>=MSB){                    // si el valor actual de tension es menor o igual que el anterior esta descargandose
       charging_state=0x00;                      // no ecualiza
       last_voltage[0]=MSB;    // y almacena el nuevo valor.
   }

   count_up=0x000;  // reinicia la cuenta.
}

if  (ecualization_form==0x11) // ecualiza durante la carga y en reposo
{
  if (MSB >= last_voltage[0]){  // si el valor actual de tensión es mayor o igual que el anterior. está cargando
       charging_state=0x01;  // activa la ecualización.
       last_voltage[0]=MSB;    // y almacena el nuevo valor.
   }

   if (last_voltage[0]>MSB){                     // si el valor actual de tensión es menor que el anterior esta descargandose
       charging_state=0x00;                      // no ecualiza
       last_voltage[0]=MSB;    // y almacena el nuevo valor.
   }

   count_up=0x000;  // reinicia la cuenta.
}



  return;

}



void balance_capture (char integrado)
{
volatile char MLSB1,MLSB2,MLSB3,MLSB4,MLSB5,MLSB6;
volatile char MSB1,MSB2,MSB3,MSB4,MSB5,MSB6;
volatile char LSB1,LSB2,LSB3,LSB4,LSB5,LSB6;
volatile char balancing;
```

```
volatile char stock;
volatile char gross=0x00;
volatile float voltage_prueba1;
 volatile char test1,test,test2,test3,test4,test5;
 volatile long pruebita,pruebita2,pruebita3,pruebita4,pruebita5;



                    P1OUT ^= 0x01;
                    escribir(0x01,0x34,0x01);
                    integrado=0x01;
                    balancing=0x00;

                    MSB1=leer(integrado,0x03);
                    MSB2=leer(integrado,0x05);
                    MSB3=leer(integrado,0x07);
                    MSB4=leer(integrado,0x09);
                    MSB5=leer(integrado,0x0b);
                    MSB6=leer(integrado,0x0d);
                    LSB1=leer(integrado,0x04);
                    LSB2=leer(integrado,0x06);
                    LSB3=leer(integrado,0x08);
                    LSB4=leer(integrado,0x0a);
                    LSB5=leer(integrado,0x0c);
                    LSB6=leer(integrado,0x0e);

                    stock=MSB1;

                    // recorro los valores para ver cual es el mas bajo.
                    if (MSB2 < stock)
                    {   stock=MSB2;
                    }
                    if (MSB3 < stock)
                    {   stock=MSB3;
                    }
                    if (MSB4 < stock)
                    {   stock=MSB4;
                    }
                    if (MSB5 < stock)
                    {   stock=MSB5;
                    }
                    if (MSB6 < stock)
                    {   stock=MSB6;
                    }
                    // Ahora en stock esta almacenado el valor mas chico de voltage.
                    // recorro nuevamente, cualquier valor de tension mayor q stock, balanceo.

                    if (MSB1 > stock)
                    {   balancing ^= 0x01;
                    }
                    if (MSB2 > stock)
                    {   balancing ^= 0x02;
                    }
                    if (MSB3 > stock)
                    {   balancing ^= 0x04;
                    }
                    if (MSB4 > stock)
                    {   balancing ^= 0x08;
                    }
                    if (MSB5 > stock)
                    {   balancing ^= 0x10;
                    }
                    if (MSB6 > stock)
                    {   balancing ^= 0x20;
                    }

                    if(balancing==0x00)
                    {

                     gross=0x41;
                       MLSB1= LSB1 & 0xf0;
                       MLSB2= LSB2 & 0xf0;
                       MLSB3= LSB3 & 0xf0;
                       MLSB4= LSB4 & 0xf0;
                       MLSB5= LSB5 & 0xf0;
                       MLSB6= LSB6 & 0xf0;
                       // recorro para ver cual es el menor voltage
                       stock=MLSB1;
                       if (MLSB2 < stock)
```

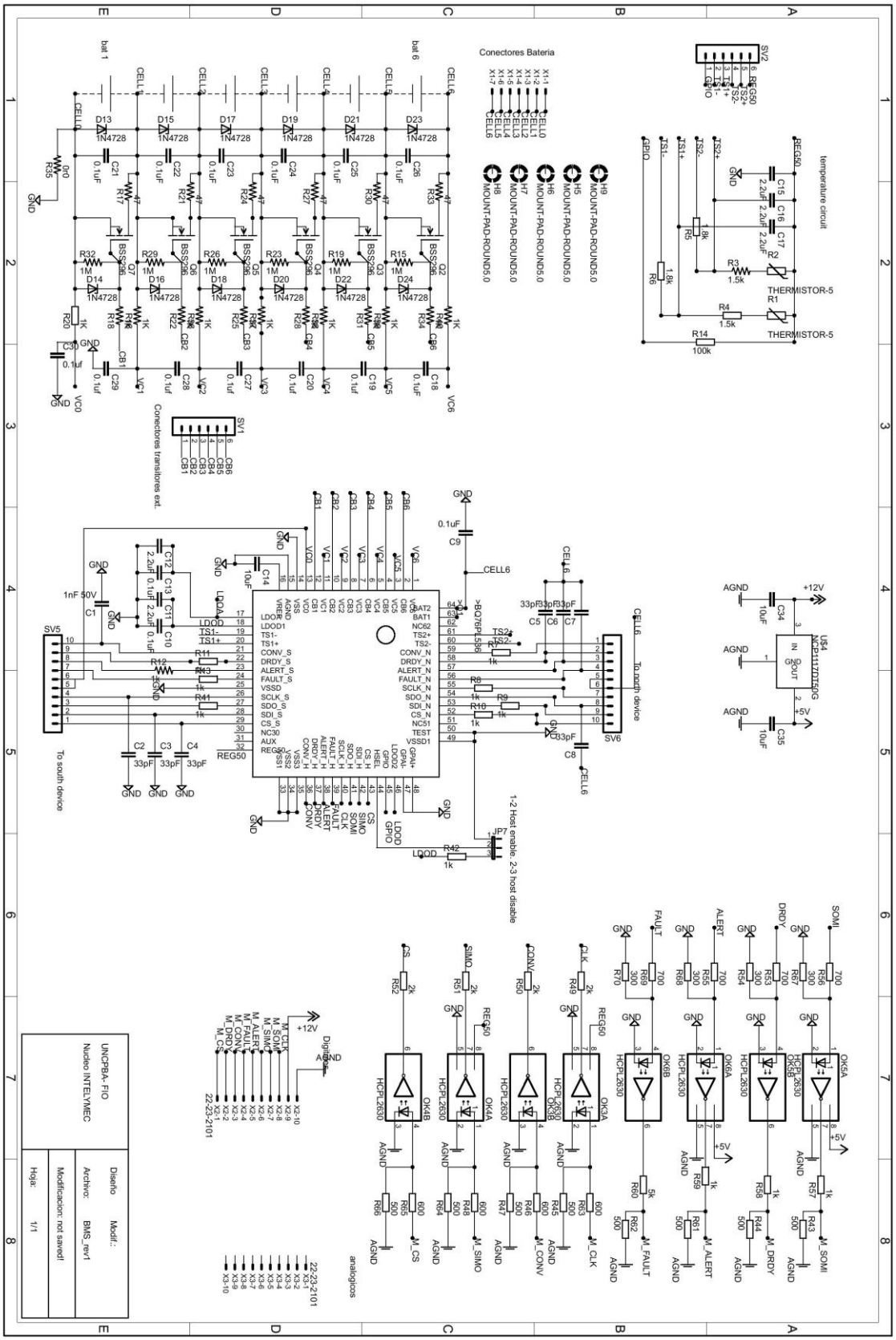
```
   {   stock=MLSB2;
   }
   if (MLSB3 < stock)
   {   stock=MLSB3;
   }
   if (MLSB4 < stock)
   {   stock=MLSB4;
   }
   if (MLSB5 < stock)
   {   stock=MLSB5;
   }
   if (MLSB6 < stock)
   {   stock=MLSB6;
   }

   if (MLSB1 > stock)
   {   balancing ^= 0x01;
   }
   if (MLSB2 > stock)
   {   balancing ^= 0x02;
   }
   if (MLSB3 > stock)
   {   balancing ^= 0x04;
   }
   if (MLSB4 > stock)
   {   balancing ^= 0x08;
   }
   if (MLSB5 > stock)
   {   balancing ^= 0x10;
   }
   if (MLSB6 > stock)
   {   balancing ^= 0x20;
   }
   }


if(balancing != 0x00)
 {
   escribir(0x01,0x32,balancing); // Lo desactivo por las dudas. Pero es este el registro que tengo que modificar cuando
quiero balancear
 }


voltage_prueba1=voltage_celda(MSB1,LSB1);
//voltage_prueba=3515.84;

pruebita=voltage_prueba1;
pruebita2=(pruebita % 10)+0x30;   // unidad

pruebita=pruebita/10;
pruebita3=(pruebita % 10)+0x30;   // decena

pruebita=pruebita/10;
pruebita4=(pruebita % 10)+0x30;   // centena

pruebita=pruebita/10;
pruebita5=(pruebita % 10)+0x30;   // millar

IFG2 &= ~UCA0RXIFG ;  // clear flag

 UCA0TXBUF=pruebita5;                //escribo el millar
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita4;                // centena
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita3;                  //decena
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita2;                //unidad
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=0x20;                // espacio

 voltage_prueba1=voltage_celda(MSB2,LSB2);
//voltage_prueba=3515.84;

pruebita=voltage_prueba1;
pruebita2=(pruebita % 10)+0x30;   // unidad

pruebita=pruebita/10;
pruebita3=(pruebita % 10)+0x30;   // decena

pruebita=pruebita/10;
```

```
pruebita4=(pruebita % 10)+0x30;   // centena

pruebita=pruebita/10;
pruebita5=(pruebita % 10)+0x30;   // millar

IFG2 &= ~UCA0RXIFG ;  // clear flag

 UCA0TXBUF=pruebita5;              //escribo el millar
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita4;              // centena
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita3;                 //decena
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita2;              //unidad
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=0x20;                // espacio

 voltage_prueba1=voltage_celda(MSB3,LSB3);
//voltage_prueba=3515.84;

pruebita=voltage_prueba1;
pruebita2=(pruebita % 10)+0x30;   // unidad

pruebita=pruebita/10;
pruebita3=(pruebita % 10)+0x30;   // decena

pruebita=pruebita/10;
pruebita4=(pruebita % 10)+0x30;   // centena

pruebita=pruebita/10;
pruebita5=(pruebita % 10)+0x30;   // millar

IFG2 &= ~UCA0RXIFG ;  // clear flag

 UCA0TXBUF=pruebita5;              //escribo el millar
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita4;              // centena
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita3;                 //decena
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita2;              //unidad
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=0x20;                // espacio

 voltage_prueba1=voltage_celda(MSB4,LSB4);
                  //voltage_prueba=3515.84;

pruebita=voltage_prueba1;
pruebita2=(pruebita % 10)+0x30;   // unidad

pruebita=pruebita/10;
pruebita3=(pruebita % 10)+0x30;   // decena

pruebita=pruebita/10;
pruebita4=(pruebita % 10)+0x30;   // centena

pruebita=pruebita/10;
pruebita5=(pruebita % 10)+0x30;   // millar

IFG2 &= ~UCA0RXIFG ;  // clear flag

 UCA0TXBUF=pruebita5;              //escribo el millar
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita4;              // centena
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita3;                 //decena
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=pruebita2;              //unidad
 while (!(IFG2&UCA0TXIFG));
 UCA0TXBUF=0x20;                // espacio

 voltage_prueba1=voltage_celda(MSB5,LSB5);
                                  //voltage_prueba=3515.84;

 pruebita=voltage_prueba1;
 pruebita2=(pruebita % 10)+0x30;   // unidad

 pruebita=pruebita/10;
 pruebita3=(pruebita % 10)+0x30;   // decena
```

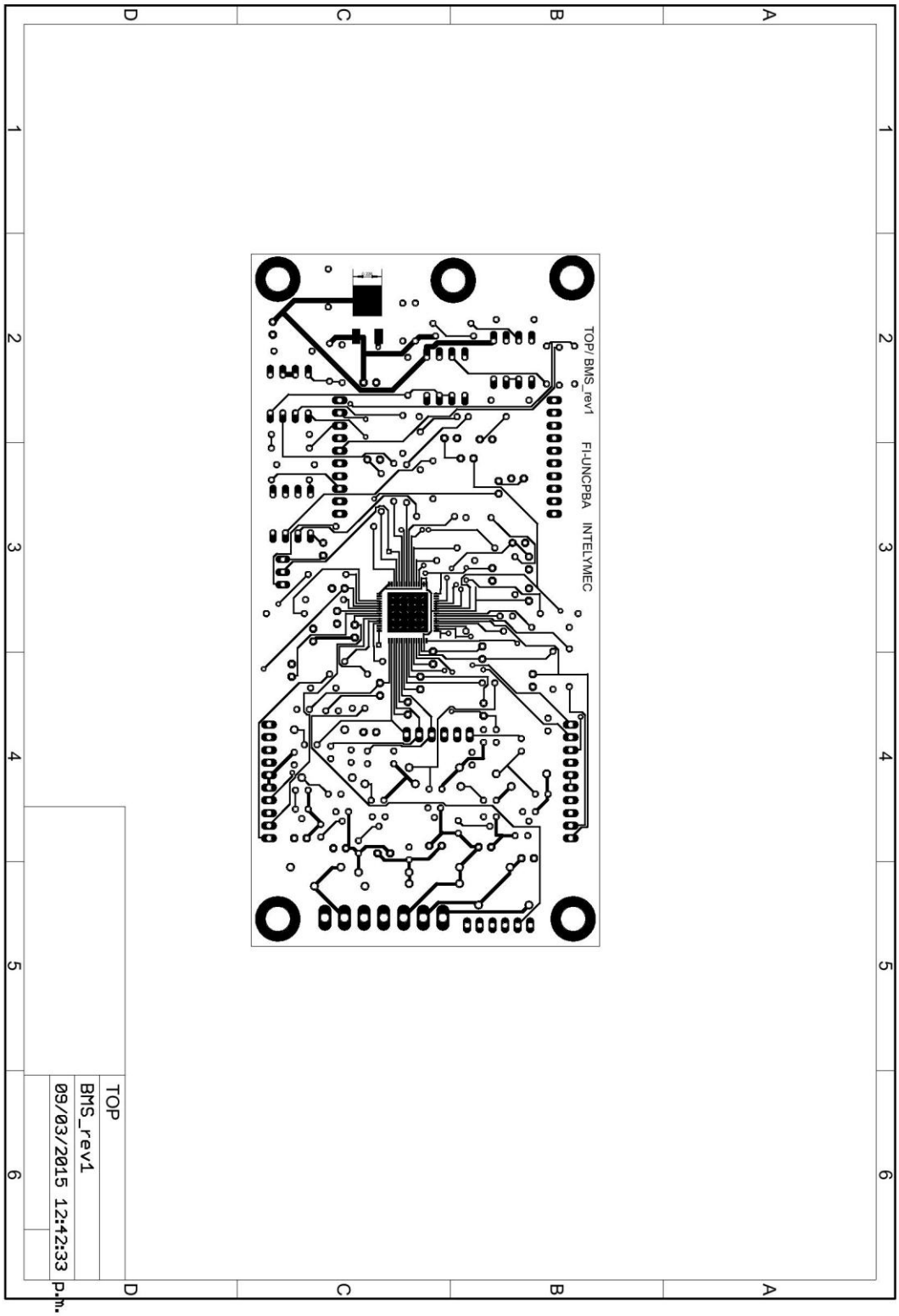
```
                        pruebita=pruebita/10;
                        pruebita4=(pruebita % 10)+0x30;   // centena

                        pruebita=pruebita/10;
                        pruebita5=(pruebita % 10)+0x30;   // millar

                        IFG2 &= ~UCA0RXIFG ;  // clear flag

                         UCA0TXBUF=pruebita5;               //escribo el millar
                         while (!(IFG2&UCA0TXIFG));
                         UCA0TXBUF=pruebita4;               // centena
                         while (!(IFG2&UCA0TXIFG));
                         UCA0TXBUF=pruebita3;                 //decena
                         while (!(IFG2&UCA0TXIFG));
                         UCA0TXBUF=pruebita2;               //unidad
                         while (!(IFG2&UCA0TXIFG));
                         UCA0TXBUF=0x20;                 // espacio


                         voltage_prueba1=voltage_celda(MSB6,LSB6);
                                                 //voltage_prueba=3515.84;

                         pruebita=voltage_prueba1;
                         pruebita2=(pruebita % 10)+0x30;   // unidad

                         pruebita=pruebita/10;
                         pruebita3=(pruebita % 10)+0x30;   // decena

                         pruebita=pruebita/10;
                         pruebita4=(pruebita % 10)+0x30;   // centena

                         pruebita=pruebita/10;
                         pruebita5=(pruebita % 10)+0x30;   // millar

                         IFG2 &= ~UCA0RXIFG ;  // clear flag

                          UCA0TXBUF=pruebita5;               //escribo el millar
                          while (!(IFG2&UCA0TXIFG));
                          UCA0TXBUF=pruebita4;               // centena
                          while (!(IFG2&UCA0TXIFG));
                          UCA0TXBUF=pruebita3;                 //decena
                          while (!(IFG2&UCA0TXIFG));
                          UCA0TXBUF=pruebita2;               //unidad
                          while (!(IFG2&UCA0TXIFG));
                          UCA0TXBUF=0x20;                 // espacio


                          while (!(IFG2&UCA0TXIFG));
                          UCA0TXBUF=balancing;              //
                          while (!(IFG2&UCA0TXIFG));
                          UCA0TXBUF=0x20;                 // espacio



                             while (!(IFG2&UCA0TXIFG));
                             UCA0TXBUF=gross;


                          while (!(IFG2&UCA0TXIFG));
                         UCA0TXBUF=0x0a;              //enter
                         while (!(IFG2&UCA0TXIFG));
                         UCA0TXBUF=0x0d;              // retorno de carro
                         IFG2=0x00;

                         gross=0x00;
                        // _delay_cycles(2020000);


}

void capture_data ()
{


    volatile char MLSB1,MLSB2,MLSB3,MLSB4,MLSB5,MLSB6;
    volatile char MSB1,MSB2,MSB3,MSB4,MSB5,MSB6;
    volatile char LSB1,LSB2,LSB3,LSB4,LSB5,LSB6;
    volatile char balancing,i,isum,MSB,LSB;
    volatile char stock,debugger;
```

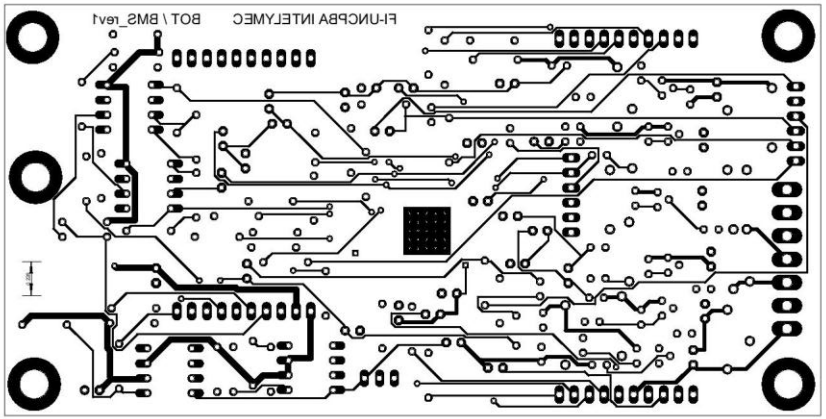
```
    volatile char gross=0x00;
    volatile float voltage_prueba1;
     volatile char test1,test,test2,test3,test4,test5;
     volatile long pruebita,pruebita2,pruebita3,pruebita4,pruebita5;
    P1OUT ^= 0x01;
            for(i=0x00;i<0x06;i++)
             {
             isum=i<<1;
             MSB=leer(0x01,0x03+isum);
             LSB=leer(0x01,0x04+isum);
             //MSB=0x20;
             //LSB=0x00;
             voltage_prueba1=voltage_celda(MSB,LSB);
             //voltage_prueba=3515.84;
             pruebita=voltage_prueba1;
             pruebita2=(pruebita % 10)+0x30;   // unidad
             pruebita=pruebita/10;
             pruebita3=(pruebita % 10)+0x30;   // decena
             pruebita=pruebita/10;
             pruebita4=(pruebita % 10)+0x30;   // centena
             pruebita=pruebita/10;
             pruebita5=(pruebita % 10)+0x30;   // millar
             IFG2 &= ~UCA0RXIFG ;  // clear flag
              UCA0TXBUF=pruebita5;                //escribo el millar
              while (!(IFG2&UCA0TXIFG));
              UCA0TXBUF=pruebita4;                // centena
              while (!(IFG2&UCA0TXIFG));
              UCA0TXBUF=pruebita3;                  //decena
              while (!(IFG2&UCA0TXIFG));
              UCA0TXBUF=pruebita2;                //unidad
              while (!(IFG2&UCA0TXIFG));
              UCA0TXBUF=0x20;                 // espacio
              if(i==0x05)
              {
                  while (!(IFG2&UCA0TXIFG));
                  UCA0TXBUF=0x0a;             //enter
                  while (!(IFG2&UCA0TXIFG));
                  UCA0TXBUF=0x0d;             // retorno de carro
                  IFG2=0x00;
                  _delay_cycles(1000);
              }
               }
             escribir(0x01,0x34,0x01);
             if(debugger==0x02){
             _delay_cycles(5500000);
             }
             if(debugger==0x01){
             _delay_cycles(2020000);
              }
             if(debugger==0x03){
              _delay_cycles(5000000);
              }
}
```

6.3 Esquemático Placa Base

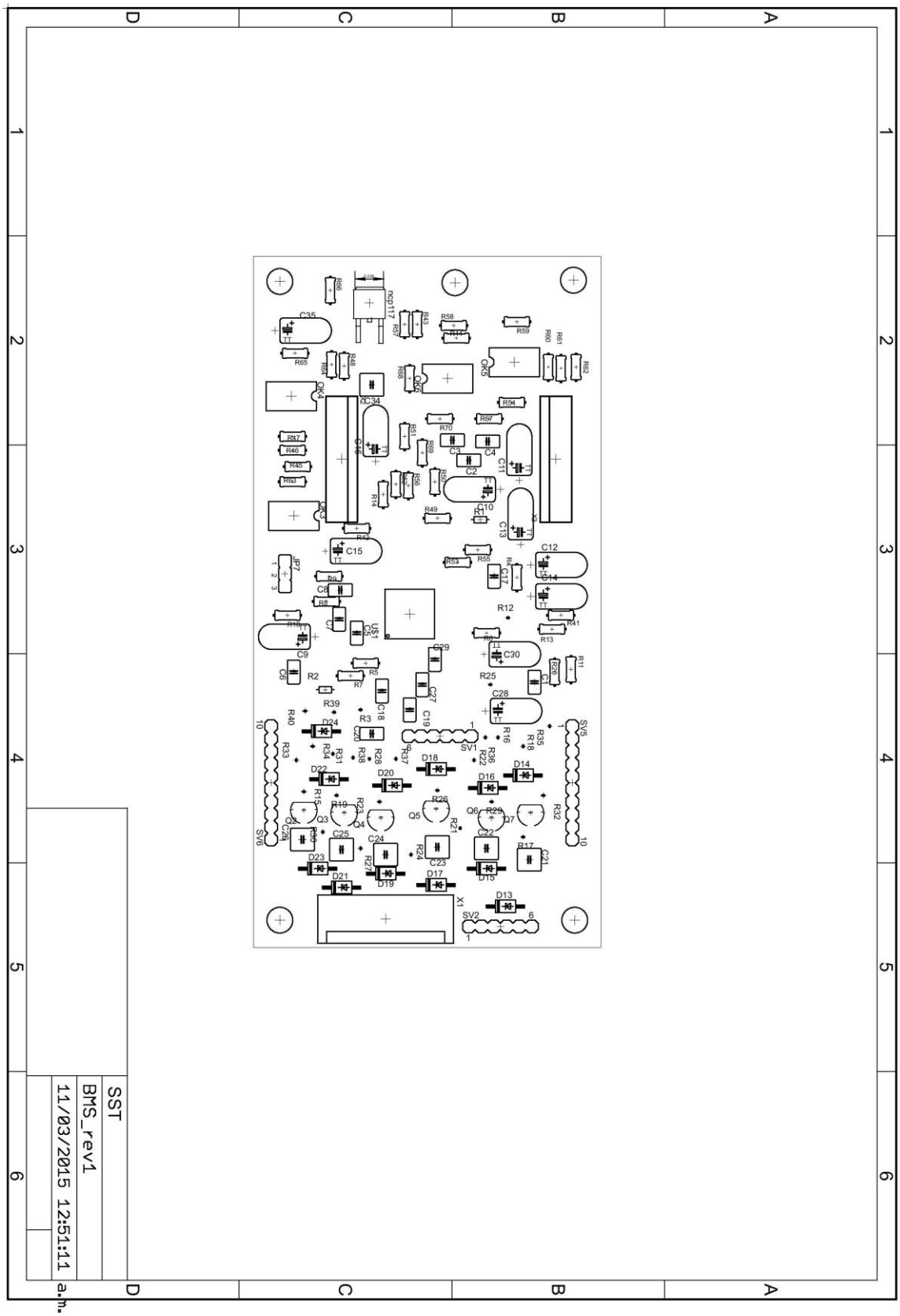


6.4 Board Layout Placa Base

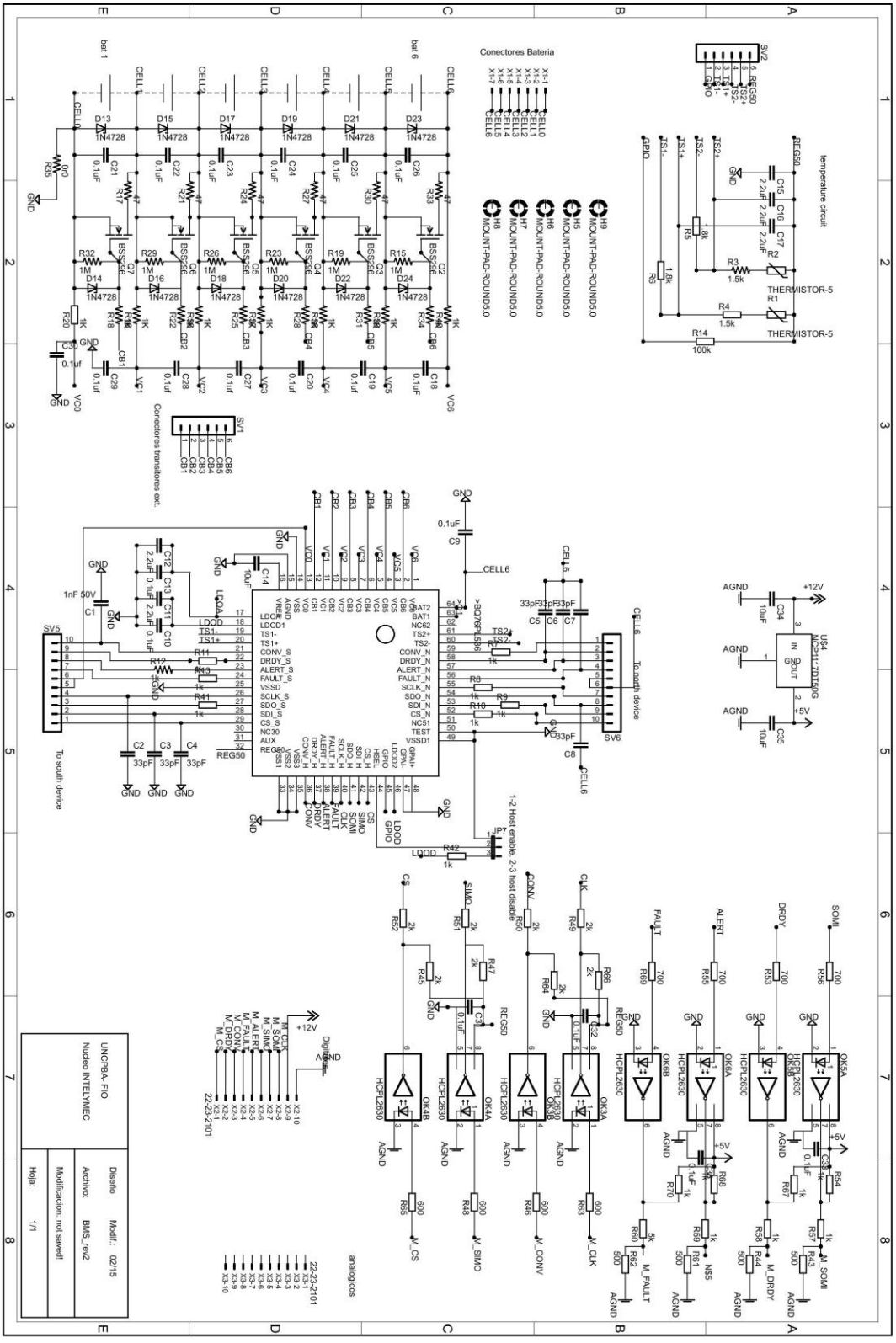




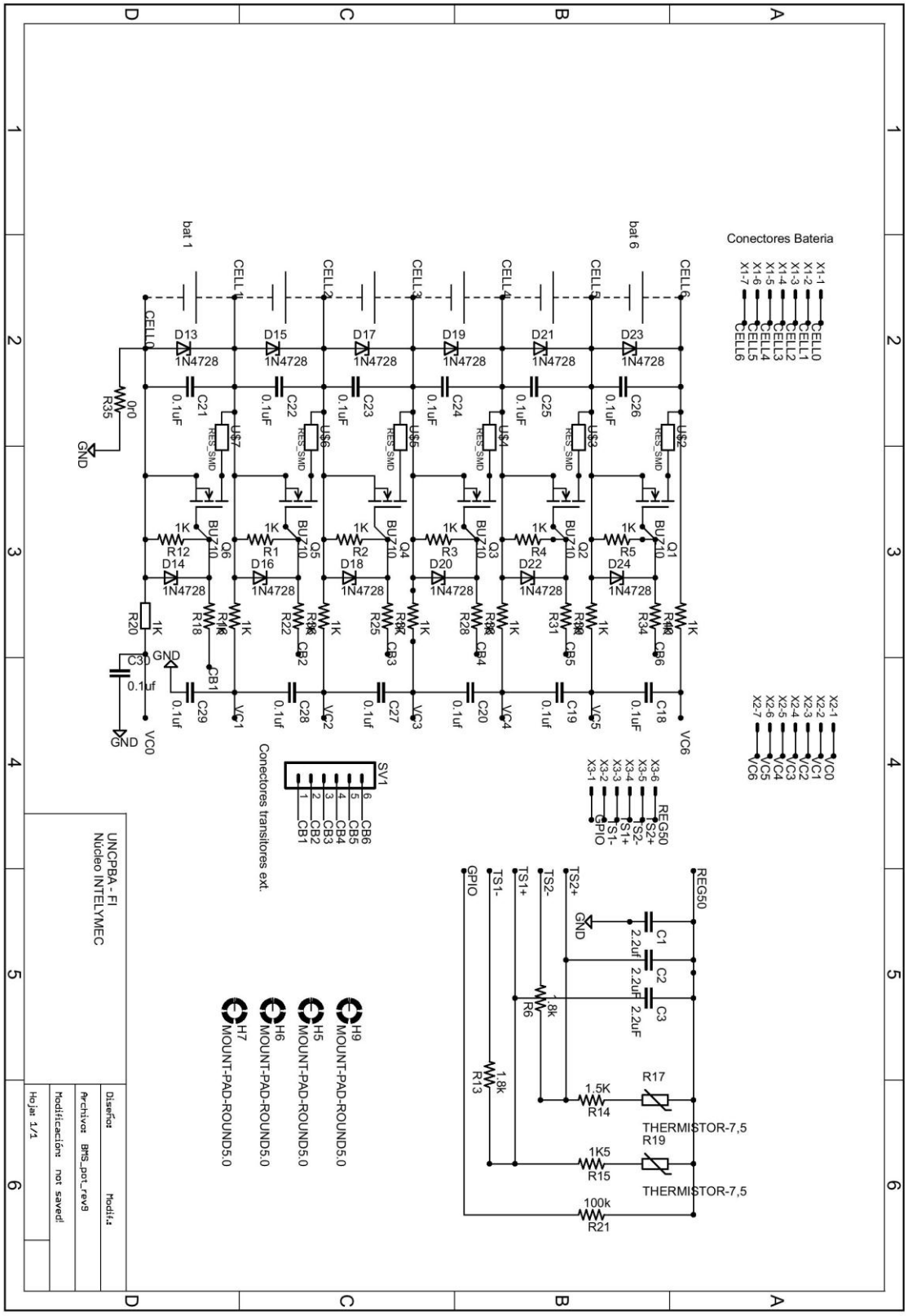
	BOT
	BMS_rev1
	09/03/2015 12:45:49 p.m.


SST
BMS_rev1
11/03/2015 12:51:11 a.m.

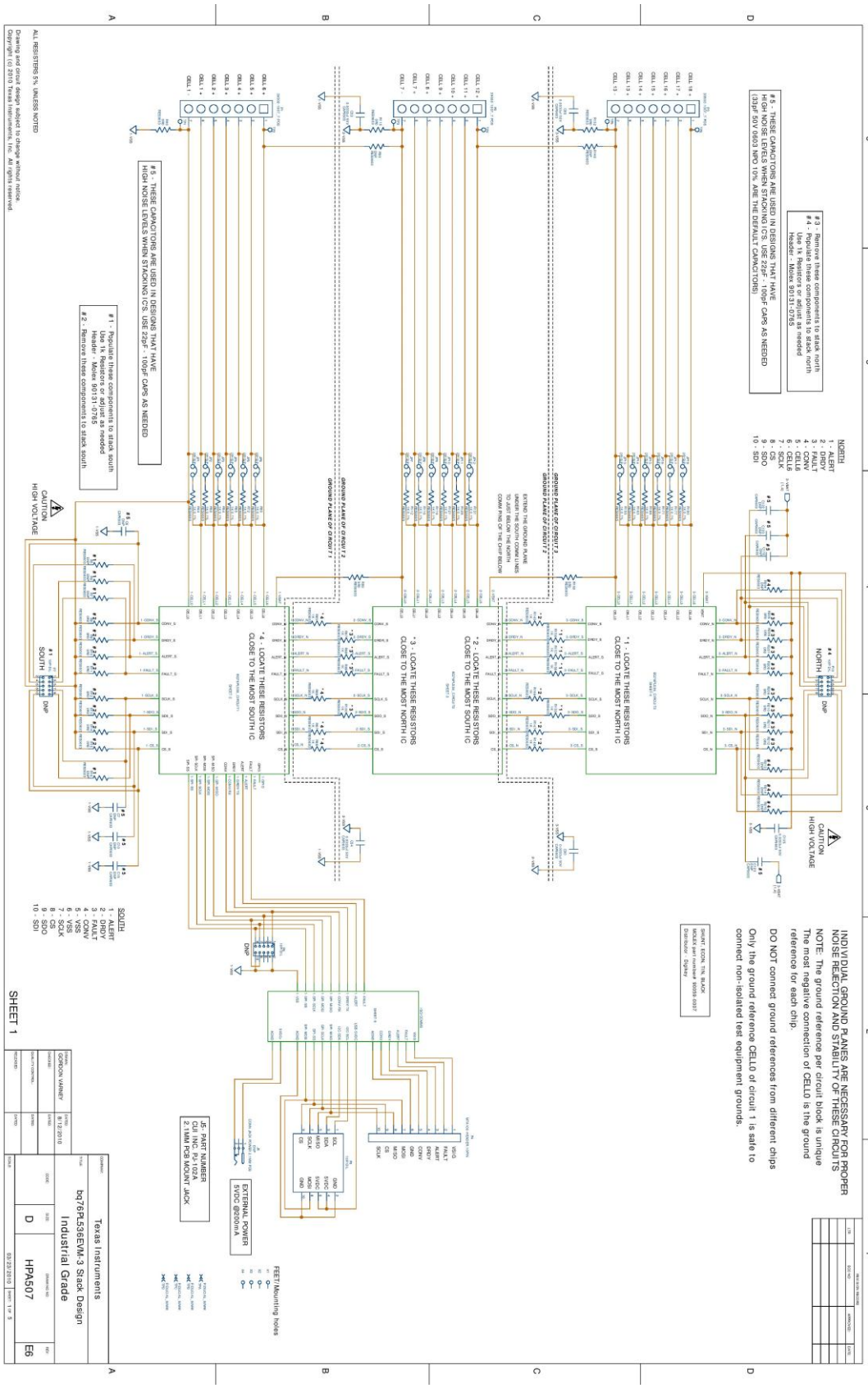
6.5 Esquemático de la PB con correcciones prupuestas

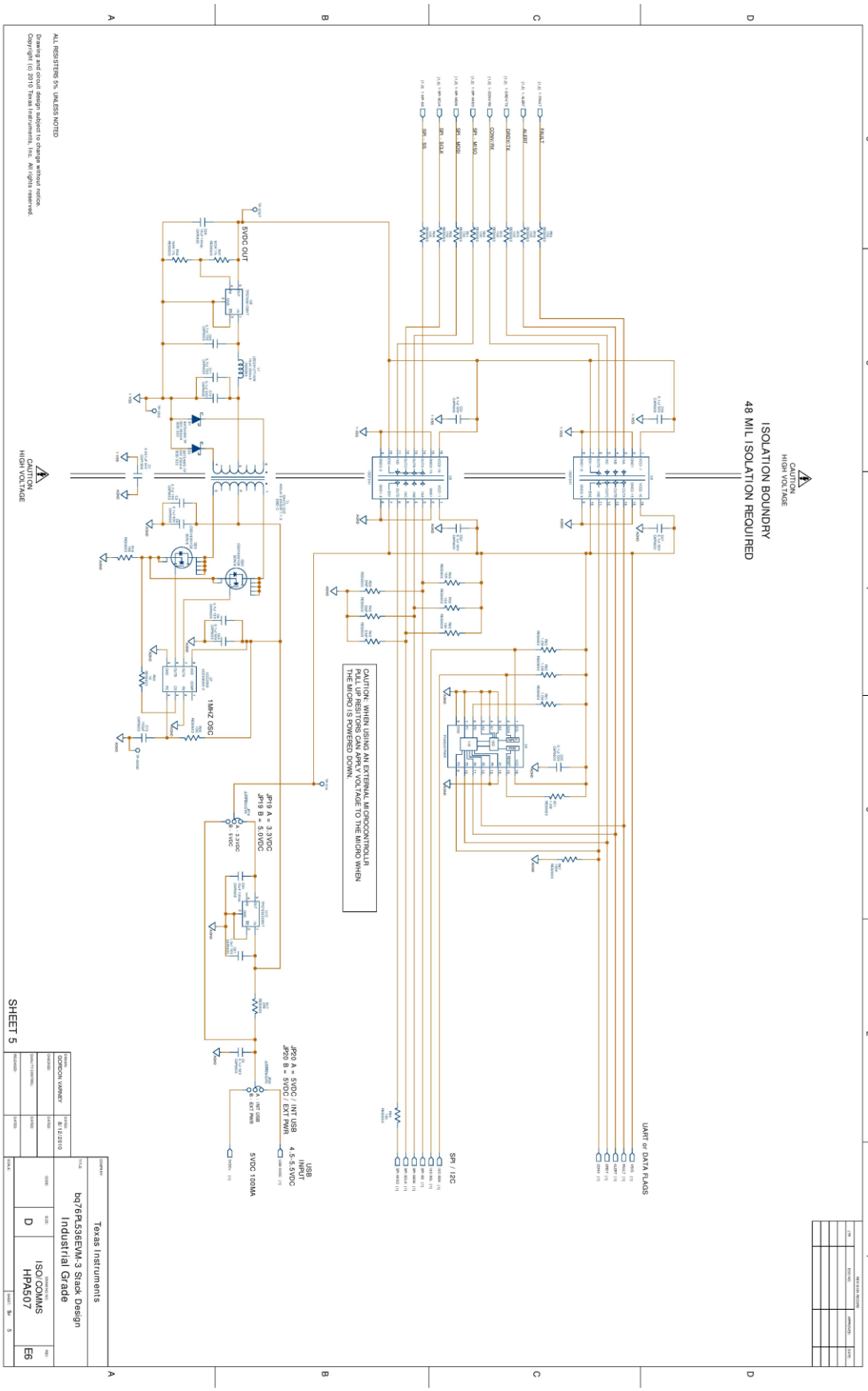


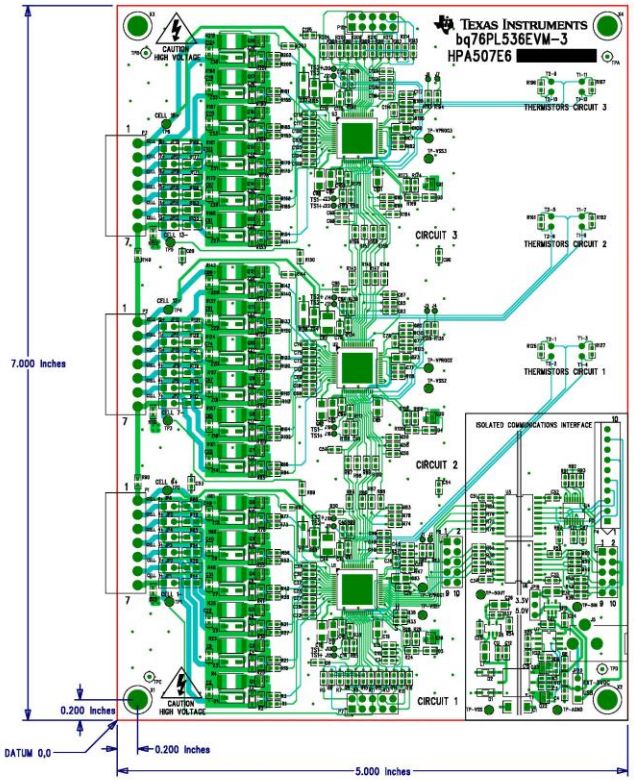
6.6 Esquemático de la placa de potencia propuesta



6.7 Esquemáticos y Board layout placa de Texas Instruments EVM-BQ76PL536




ISOLATION BOUNDRY
4B MIL ISOLATION REQUIRED
CAUTION
HIGH VOLTAGE
CAUTION: WHEN USING AN EXTERNAL MICRO/CONTROLLER PULL UP RESISTORS CAN APPLY VOLTAGE TO THE MICRO WHEN THE MICRO IS POWERED DOWN
5VDC OUT
1MHZ OSC
JP19 A = 3.3VDC
JP19 B = 5.0VDC
JP20 A = 5VDC / INT USB
JP20 B = 5VDC / EXT PWR
USB
INPUT
4.5-5.5VDC
5VDC 100MA
SPI / I2C
UART or DATA FLAGS
ALL RESISTERS 5% UNLESS NOTED
SHEET 5
Texas Instruments
bq76PL536EVM-3 Stack Design
Industrial Grade
ISO/COMMS
HPA507
E6



COMPONENT SIDE VIEW

*HOLE CHART IS IN MILS

FABRICATION CHART

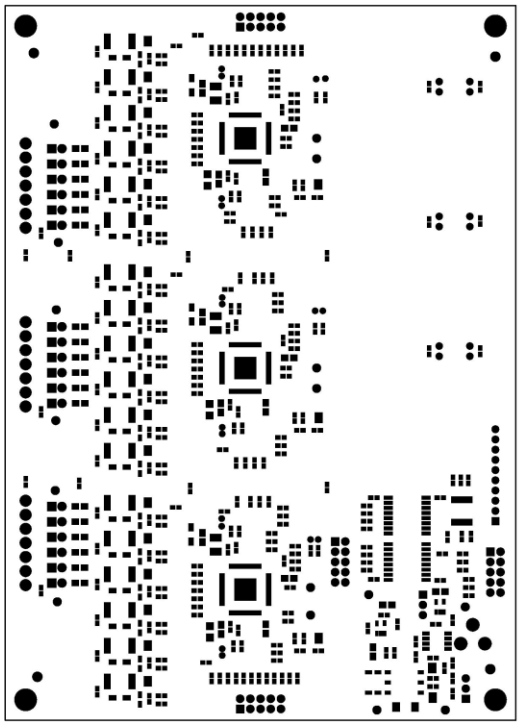
FINISHED THICKNESS	SILKSCREEN	SOLDERMASK	FINISHED COPPER WEIGHT
☐ 0.031 ☒ 0.062 ☐ 0.093 ☐ 0.125	☒ LAYER 1 ☐ LAYER 2 ☐ NONE	☒ LAYER 1 ☒ LAYER 2 ☐ NONE	☐ 1 OZ. ☒ 2 OZ. ☐ OTHER _____

DESIGN	TRACE/GAP SPACING	LAYER COUNT
☐ SMD ☐ THRU-HOLE ☒ MIX	☐ 0.010/0.010 ☒ 0.008/0.007 ☐ 0.006/0.006 ☐ OTHER _____	☐ SINGLE SIDED ☐ 2 LAYER ☒ 4 LAYER ☐ OTHER _____

NOTES: UNLESS OTHERWISE SPECIFIED

1. MATERIAL:	ALL MATERIALS, INCLUDING BUT NOT LIMITED TO BASE LAMINATE, BONDING MATERIALS AND SOLDERMASK COATINGS FORMING THE FINISHED PRINTED CIRCUIT BOARD SHALL MEET UL-796 REQUIREMENTS AND BE RoHS COMPLIANT AND HAVE A FLAMMABILITY OF UL94V-0
2. BASE LAMINATE:	PLASTIC SHEET, LAMINATED METAL CLAD, ONE OR TWO SIDES, BASE MATERIAL NE A TYPE FR-4 OR EQUIVALENT, W/Tg = 180 Deg C OR HIGHER MINIMUM COMPOSITION TEMP (Td) OF 340 Deg C. GLASS EPOXY RESIN, COPPER-CLAD IN ACCORDANCE WITH 2 LAYER STACK-UP, COMPLIANT WITH LEAD FREE PROCESS
3. SOLDERMASK:	SOLDERMASK OVER BARE COPPER (SMOBC) USING LIQUID PHOTO-IMAGEABLE SOLDERMASK IN ACCORDANCE WITH IPC-SM-840. COLOR GREEN. MINOR SOLDERMASK ADJUSTMENTS TO FACILITATE PCB FAB OR ASSEMBLY ARE ALLOWED PROVIDED NO DEFECTS ARE CREATED AS A RESULT.
4. PLATING:	HOLES REQUIRING PLATING SHALL HAVE .001 MIL MIN THICK COPPER PLATE
5. FINISH:	PLATE WITH RoHS COMPLIANT, IMMERSION SILVER PREFERRED, OR Sn/Ag/Cu, WITH RMA FLUX, 0.0005" +/ – 0.0003" THICK MIN ALL EXPOSED AREAS WITH RMA FLUX, 0.0005" +/ – 0.0003" THICK MIN ALL EXPOSED AREAS
6. LEGEND:	IF REQUIRED, SILKSCREEN LEGEND(S) WITH WHITE NON-CONDUCTIVE EPOXY INK.
7. MARKINGS:	BOARD MUST BEAR VENDOR'S IDENTIFICATION CODE (ETCH OR WHITE NON-CONDUCTIVE INK) LOCATION OPTIONAL.
8. WORKMANSHIP:	BOARD IS TO BE MANUFACTURED PER IPC-A-600 CLASS 2 REQUIREMENTS OR BETTER
9. DOCUMENTATION:	PCB VENDOR IS REQUIRED TO RETURN ANY AND ALL DOCUMENTS SUPPLIED OR ULTIMATELY PURCHASED BY TEXAS INSTRUMENTS UPON COMPLETION OF PURCHASE ORDER
10. DRILL SIZES:	HOLE DIAMETERS SHOWN ARE FINISHED SIZES AFTER PLATING UNLESS OTHERWISE NOTED.
11. TOLERANCES:	UNLESS OTHERWISE SPECIFIED PCB TOLERANCES SHALL BE: PCB THICKNESS TOLERANCE = +/ – 10% RELATIVE TO FINISHED THICKNESS, MEASURED FROM COPPER TO COPPER PCB DIMENSIONAL TOLERANCE = +/–.005 INCHES RELATIVE TO BOARD PROFILE AND DRILL TO EDGE DIMENSIONS. FINISHED HOLE DIAMETERS SHALL BE +/ –.003 INCHES RELATIVE THE DRILL TABLE DIMENSIONS.
12. PANEL BORDER:	ANY METAL IN BORDER AREA INCLUDING PART NUMBER, DATECODE AND/OR REVISION LETTERS MUST BE COVERED WITH SOLDERMASK.
13. PROCESS CHANGES:	NO DIMENSIONAL, MATERIAL, OR PROCESS CHANGES ARE ALLOWED WITHOUT PRIOR EXPLICIT WRITTEN PERMISSION FROM TEXAS INSTRUMENTS.

TEXAS INSTRUMENTS	COPPER LAYER NAME				SILKSCREEN		SOLDER MASK		PASTE MASK		ASSEMBLY		FAB DRAWING
	TOP 1	INNER 2	INNER 3	BOT 4	TOP	BOT	TOP	BOT	TOP	BOT	TOP	BOT	
BOARD NO: HPA507 REV: E6	LY1			LY4	SST								FB
DATE: 06/15/2010	FILENAME: HPA507E6 bq76PL536EVM-3 Stack Design.PCB		ENGINEER: Gordon Varney		PCB DESIGNER: Gordon Varney		MODIFIED DATE: 06/15/2010		TIME STAMP:				



FABRICATION CHART

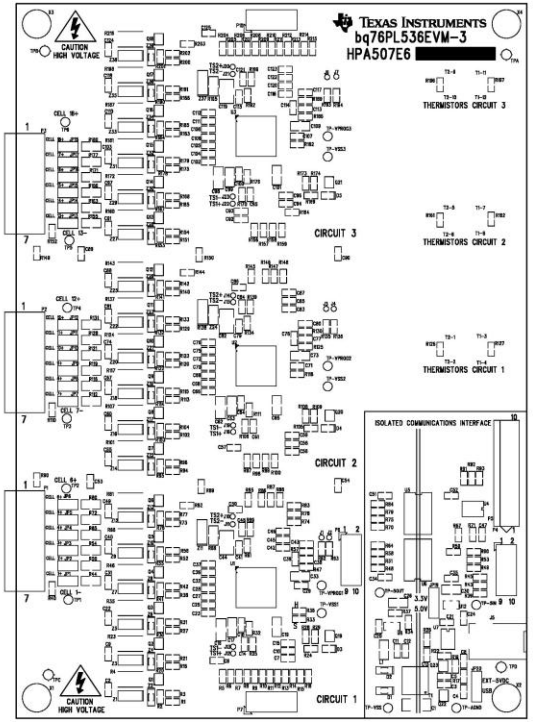
FINISHED THICKNESS	SILKSCREEN	SOLDERMASK	FINISHED COPPER WEIGHT
☐ 0.031 ☒ 0.062 ☐ 0.093 ☐ 0.125	☒ LAYER 1 ☐ LAYER 2 ☐ NONE	☒ LAYER 1 ☒ LAYER 2 ☐ NONE	☐ 1 OZ. ☒ 2 OZ. ☐ OTHER _____

DESIGN	TRACE/GAP SPACING	LAYER COUNT
☐ SMD ☐ THRU-HOLE ☒ MIX	☐ 0.010/0.010 ☒ 0.008/0.007 ☐ 0.006/0.006 ☐ OTHER _____	☐ SINGLE SIDED ☐ 2 LAYER ☒ 4 LAYER ☐ OTHER _____

NOTES: UNLESS OTHERWISE SPECIFIED

1. MATERIAL: ALL MATERIALS, INCLUDING BUT NOT LIMITED TO BASE LAMINATE, BONDING MATERIALS AND SOLDERMASK COATINGS FORMING THE FINISHED PRINTED CIRCUIT BOARD SHALL MEET UL-796 REQUIREMENTS AND BE RoHS COMPLIANT AND HAVE A FLAMMABILITY OF UL94V-0
2. BASE LAMINATE: PLASTIC SHEET, LAMINATED METAL CLAD, ONE OR TWO SIDES, BASE MATERIAL NE A TYPE FR-4 OR EQUIVALENT, W/Tg = 180 Deg C OR HIGHER MINIMUM COMPOSITION TEMP (Td) OF 340 Deg C. GLASS EPOXY RESIN, COPPER-CLAD IN ACCORDANCE WITH 2 LAYER STACK-UP, COMPLIANT WITH LEAD FREE PROCESS
3. SOLDERMASK: SOLDERMASK OVER BARE COPPER (SMOBC) USING LIQUID PHOTO-IMAGEABLE SOLDERMASK IN ACCORDANCE WITH IPC-SM-840. COLOR GREEN. MINOR SOLDERMASK ADJUSTMENTS TO FACILITATE PCB FAB OR ASSEMBLY ARE ALLOWED PROVIDED NO DEFECTS ARE CREATED AS A RESULT.
4. PLATING: HOLES REQUIRING PLATING SHALL HAVE .001 MIL MIN THICK COPPER PLATE
5. FINISH: PLATE WITH RoHS COMPLIANT, IMMERSION SILVER PREFERRED, OR $Sn/Ag/Cu_x$
   WITH RMA FLUX, 0.0005" +/ – 0.0003" THICK MIN ALL EXPOSED AREAS
   WITH RMA FLUX, 0.0005" +/ – 0.0003" THICK MIN ALL EXPOSED AREAS
6. LEGEND: IF REQUIRED, SILKSCREEN LEGEND(S) WITH WHITE NON-CONDUCTIVE EPOXY INK.
7. MARKINGS: BOARD MUST BEAR VENDOR'S IDENTIFICATION CODE (ETCH OR WHITE NON-CONDUCTIVE INK) LOCATION OPTIONAL.
8. WORKMANSHIP: BOARD IS TO BE MANUFACTURED PER IPC-A-600 CLASS 2 REQUIREMENTS OR BETTER
9. DOCUMENTATION: PCB VENDOR IS REQUIRED TO RETURN ANY AND ALL DOCUMENTS SUPPLIED OR ULTIMATELY PURCHASED BY TEXAS INSTRUMENTS UPON COMPLETION OF PURCHASE ORDER
10. DRILL SIZES: HOLE DIAMETERS SHOWN ARE FINISHED SIZES AFTER PLATING UNLESS OTHERWISE NOTED.
11. TOLERANCES: UNLESS OTHERWISE SPECIFIED PCB TOLERANCES SHALL BE:
    PCB THICKNESS TOLERANCE = +/ – 10% RELATIVE TO FINISHED THICKNESS, MEASURED FROM COPPER TO COPPER
    PCB DIMENSIONAL TOLERANCE = +/–.005 INCHES RELATIVE TO BOARD PROFILE AND DRILL TO EDGE DIMENSIONS.
    FINISHED HOLE DIAMETERS SHALL BE +/ –.003 INCHES RELATIVE THE DRILL TABLE DIMENSIONS.
12. PANEL BORDER: ANY METAL IN BORDER AREA INCLUDING PART NUMBER, DATECODE AND/OR REVISION LETTERS MUST BE COVERED WITH SOLDERMASK.
13. PROCESS CHANGES: NO DIMENSIONAL, MATERIAL, OR PROCESS CHANGES ARE ALLOWED WITHOUT PRIOR EXPLICIT WRITTEN PERMISSION FROM TEXAS INSTRUMENTS.

TEXAS INSTRUMENTS	COPPER LAYER NAME				SILKSCREEN		SOLDER MASK		PASTE MASK		ASSEMBLY		FAB DRAWING
	TOP 1	INNER 2	INNER 3	BOT 4	TOP	BOT	TOP	BOT	TOP	BOT	TOP	BOT	
BOARD NO: HPA507 REV: E6							SMT						FB
DATE: 08/15/2010	FILENAME: HPA507E6 bq78PL536EVM-3 Stack Design.PCB		ENGINEER: Gordon Varney		PCB DESIGNER: Gordon Varney		MODIFIED DATE: 08/10/2010		TIME STAMP:				



FABRICATION CHART

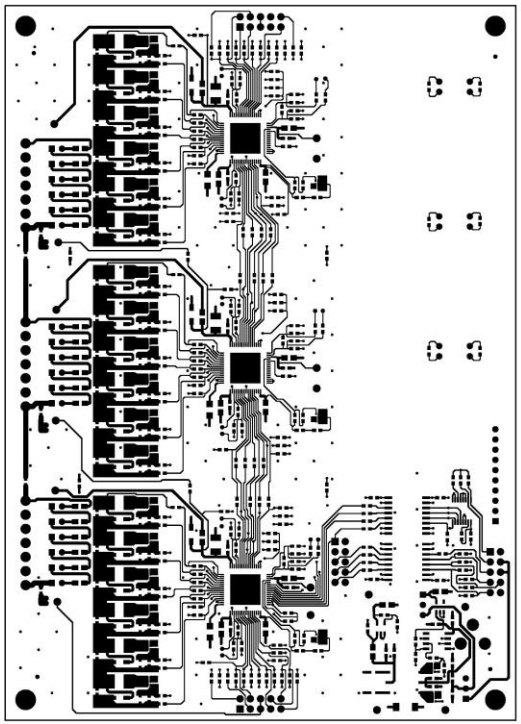
FINISHED THICKNESS	SILKSCREEN	SOLDERMASK	FINISHED COPPER WEIGHT
☐ 0.031 ☒ 0.062 ☐ 0.093 ☐ 0.125	☒ LAYER 1 ☐ LAYER 2 ☐ NONE	☒ LAYER 1 ☒ LAYER 2 ☐ NONE	☐ 1 OZ. ☒ 2 OZ. ☐ OTHER _____

DESIGN	TRACE/GAP SPACING	LAYER COUNT
☐ SMD ☐ THRU-HOLE ☒ MIX	☐ 0.010/0.010 ☒ 0.008/0.007 ☐ 0.006/0.006 ☐ OTHER _____	☐ SINGLE SIDED ☐ 2 LAYER ☒ 4 LAYER ☐ OTHER _____

NOTES: UNLESS OTHERWISE SPECIFIED

1. MATERIAL:	ALL MATERIALS, INCLUDING BUT NOT LIMITED TO BASE LAMINATE, BONDING MATERIALS AND SOLDERMASK COATINGS FORMING THE FINISHED PRINTED CIRCUIT BOARD SHALL MEET UL-796 REQUIREMENTS AND BE RoHS COMPLIANT AND HAVE A FLAMMABILITY OF UL94V-0
2. BASE LAMINATE:	PLASTIC SHEET, LAMINATED METAL CLAD, ONE OR TWO SIDES, BASE MATERIAL NE A TYPE FR-4 OR EQUIVALENT, W/Tg = 180 Deg C OR HIGHER MINIMUM COMPOSITION TEMP (Td) OF 340 Deg C. GLASS EPOXY RESIN, COPPER-CLAD IN ACCORDANCE WITH 2 LAYER STACK-UP, COMPLIANT WITH LEAD FREE PROCESS
3. SOLDERMASK:	SOLDERMASK OVER BARE COPPER (SMOBC) USING LIQUID PHOTO-IMAGEABLE SOLDERMASK IN ACCORDANCE WITH IPC-SM-840. COLOR GREEN. MINOR SOLDERMASK ADJUSTMENTS TO FACILITATE PCB FAB OR ASSEMBLY ARE ALLOWED PROVIDED NO DEFECTS ARE CREATED AS A RESULT.
4. PLATING:	HOLES REQUIRING PLATING SHALL HAVE .001 MIL MIN THICK COPPER PLATE
5. FINISH:	PLATE WITH RoHS COMPLIANT, IMMERSION SILVER PREFERRED, OR Sn/Ag/Cu. WITH RMA FLUX, 0.0005" +/ – 0.0003" THICK MIN ALL EXPOSED AREAS WITH RMA FLUX, 0.0005" +/ – 0.0003" THICK MIN ALL EXPOSED AREAS
6. LEGEND:	IF REQUIRED, SILKSCREEN LEGEND(S) WITH WHITE NON-CONDUCTIVE EPOXY INK.
7. MARKINGS:	BOARD MUST BEAR VENDOR'S IDENTIFICATION CODE (ETCH OR WHITE NON-CONDUCTIVE INK) LOCATION OPTIONAL.
8. WORKMANSHIP:	BOARD IS TO BE MANUFACTURED PER IPC-A-600 CLASS 2 REQUIREMENTS OR BETTER
9. DOCUMENTATION:	PCB VENDOR IS REQUIRED TO RETURN ANY AND ALL DOCUMENTS SUPPLIED OR ULTIMATELY PURCHASED BY TEXAS INSTRUMENTS UPON COMPLETION OF PURCHASE ORDER
10. DRILL SIZES:	HOLE DIAMETERS SHOWN ARE FINISHED SIZES AFTER PLATING UNLESS OTHERWISE NOTED.
11. TOLERANCES:	UNLESS OTHERWISE SPECIFIED PCB TOLERANCES SHALL BE: PCB THICKNESS TOLERANCE = +/ – 10x RELATIVE TO FINISHED THICKNESS, MEASURED FROM COPPER TO COPPER PCB DIMENSIONAL TOLERANCE = +/–.005 INCHES RELATIVE TO BOARD PROFILE AND DRILL TO EDGE DIMENSIONS. FINISHED HOLE DIAMETERS SHALL BE +/ –.003 INCHES RELATIVE THE DRILL TABLE DIMENSIONS.
12. PANEL BORDER:	ANY METAL IN BORDER AREA INCLUDING PART NUMBER, DATECODE AND/OR REVISION LETTERS MUST BE COVERED WITH SOLDERMASK.
13. PROCESS CHANGES:	NO DIMENSIONAL, MATERIAL, OR PROCESS CHANGES ARE ALLOWED WITHOUT PRIOR EXPLICIT WRITTEN PERMISSION FROM TEXAS INSTRUMENTS.

TEXAS INSTRUMENTS	COPPER LAYER NAME				SILKSCREEN		SOLDER MASK		PASTE MASK		ASSEMBLY		FAB DRAWING
	TOP 1	INNER 2	INNER 3	BOT 4	TOP	BOT	TOP	BOT	TOP	BOT	TOP	BOT	
BOARD NO: HPA507 REV: E6	LY1				SST								FB
DATE: 06/15/2010	FILENAME: HPA507E6 bq76PL536EVM-3 Stack Design.PCB	ENGINEER: Gordon Varney	PCB DESIGNER: Gordon Varney	MODIFIED DATE: 06/15/2010	TIME STAMP:								



FABRICATION CHART			
FINISHED THICKNESS	SILKSCREEN	SOLDERMASK	FINISHED COPPER WEIGHT
☐ 0.031 ☒ 0.062 ☐ 0.093 ☐ 0.125	☒ LAYER 1 ☐ LAYER 2 ☐ NONE	☒ LAYER 1 ☒ LAYER 2 ☐ NONE	☐ 1 OZ. ☒ 2 OZ. ☐ OTHER _____
DESIGN	TRACE/GAP SPACING		LAYER COUNT
☐ SMD ☐ THRU-HOLE ☒ MIX	☐ 0.010/0.010 ☒ 0.008/0.007 ☐ 0.006/0.006 ☐ OTHER _____		☐ SINGLE SIDED ☐ 2 LAYER ☒ 4 LAYER ☐ OTHER _____

TEXAS INSTRUMENTS		COPPER LAYER NAME				SILKSCREEN		SOLDER MASK		PASTE MASK		ASSEMBLY		FAB DRAWING
		TOP 1	INNER 2	INNER 3	BOT 4	TOP	BOT	TOP	BOT	TOP	BOT	TOP	BOT	
BOARD NO: HPA507	REV: E6	LY1												FB
DATE: 06/15/2010	FILENAME: HPA507E6 bq78PL538EVM-3 Stack Design.PCB	ENGINEER: Gordon Varney			PCB DESIGNER: Gordon Varney			MODIFIED DATE: 06/15/2010				TIME STAMP:		

NOTES: UNLESS OTHERWISE SPECIFIED

1. MATERIAL:	ALL MATERIALS, INCLUDING BUT NOT LIMITED TO BASE LAMINATE, BONDING MATERIALS AND SOLDERMASK COATINGS FORMING THE FINISHED PRINTED CIRCUIT BOARD SHALL MEET UL-796 REQUIREMENTS AND BE RoHS COMPLIANT AND HAVE A FLAMMABILITY OF UL94V-0
2. BASE LAMINATE:	PLASTIC SHEET, LAMINATED METAL CLAD, ONE OR TWO SIDES, BASE MATERIAL NE A TYPE FR-4 OR EQUIVALENT, W/Tg = 180 Deg C OR HIGHER MINIMUM COMPOSITION TEMP (Td) OF 340 Deg C. GLASS EPOXY RESIN, COPPER-CLAD IN ACCORDANCE WITH 2 LAYER STACK-UP, COMPLIANT WITH LEAD FREE PROCESS
3. SOLDERMASK:	SOLDERMASK OVER BARE COPPER (SMOBC) USING LIQUID PHOTO-IMAGEABLE SOLDERMASK IN ACCORDANCE WITH IPC-SM-840. COLOR GREEN. MINOR SOLDERMASK ADJUSTMENTS TO FACILITATE PCB FAB OR ASSEMBLY ARE ALLOWED PROVIDED NO DEFECTS ARE CREATED AS A RESULT.
4. PLATING:	HOLES REQUIRING PLATING SHALL HAVE .001 ML MIN THICK COPPER PLATE
5. FINISH:	PLATE WITH RoHS COMPLIANT, IMMERSION SILVER PREFERRED, OR Sn/Ag/Cu. WITH RMA FLUX, 0.0005" +/ – 0.0003" THICK MIN ALL EXPOSED AREAS WITH RMA FLUX, 0.0005" +/ – 0.0003" THICK MIN ALL EXPOSED AREAS
6. LEGEND:	IF REQUIRED, SILKSCREEN LEGEND(S) WITH WHITE NON-CONDUCTIVE EPOXY INK.
7. MARKINGS:	BOARD MUST BEAR VENDOR'S IDENTIFICATION CODE (ETCH OR WHITE NON-CONDUCTIVE INK) LOCATION OPTIONAL.
8. WORKMANSHIP:	BOARD IS TO BE MANUFACTURED PER IPC-A-600 CLASS 2 REQUIREMENTS OR BETTER
9. DOCUMENTATION:	PCB VENDOR IS REQUIRED TO RETURN ANY AND ALL DOCUMENTS SUPPLIED OR ULTIMATELY PURCHASED BY TEXAS INSTRUMENTS UPON COMPLETION OF PURCHASE ORDER
10. DRILL SIZES:	HOLE DIAMETERS SHOWN ARE FINISHED SIZES AFTER PLATING UNLESS OTHERWISE NOTED.
11. TOLERANCES:	UNLESS OTHERWISE SPECIFIED PCB TOLERANCES SHALL BE: PCB THICKNESS TOLERANCE = +/ – 10% RELATIVE TO FINISHED THICKNESS, MEASURED FROM COPPER TO COPPER PCB DIMENSIONAL TOLERANCE = +/–.005 INCHES RELATIVE TO BOARD PROFILE AND DRILL TO EDGE DIMENSIONS. FINISHED HOLE DIAMETERS SHALL BE +/ –.003 INCHES RELATIVE THE DRILL TABLE DIMENSIONS.
12. PANEL BORDER:	ANY METAL IN BORDER AREA INCLUDING PART NUMBER, DATECODE AND/OR REVISION LETTERS MUST BE COVERED WITH SOLDERMASK.
13. PROCESS CHANGES:	NO DIMENSIONAL, MATERIAL, OR PROCESS CHANGES ARE ALLOWED WITHOUT PRIOR EXPLICIT WRITTEN PERMISSION FROM TEXAS INSTRUMENTS.