APÉNDICES

APÉNDICE A

```
/*****************************************************************************/
/*****************************************************************************/
/** Nombre del Archivo: Config.c          ****************************************/
/** Autor: Álvaro Payán Rodríguez         ****************************************/
/** Fecha: 13-09-2004                     ****************************************/
/*****************************************************************************/
/*****************************************************************************/
/** Descripción: función que se comunica con el sensor de fuerza a traves del  puerto serie
 *          y le pide un flujo continuo de datos
 *
 *  1 salida:  bytes enviados por el puerto serie(4)
 *
 *
 *  0 entradas:
 *
 *  3 parámetros:Tiempo de muestreo (1)
 *              bias (1)
 *              tipo de datos solicitados(1)
 *
 *
/*****************************************************************************/
/*****************************************************************************/


#define S_FUNCTION_NAME config
#define S_FUNCTION_LEVEL 2

#include <math.h>
#include <stdlib.h>

#include "simstruc.h"


/* Variables del espacio de trabajo */
#define    sec          rwork[ 0]                       /* secuencia de envío */

#define    T(i)         (*uPtrs[i])                     /* referencia a las entradas*/

/*Parametros*/
#define    Tm(S)       ssGetSFcnParam(S,0)          /*Tiempo de muestreo*/
#define    BIAS(S)      ssGetSFcnParam(S,1)          /*Parámetro de Bias*/
#define    TIPO(S)      ssGetSFcnParam(S,2)          /*Parámetro tipo de datos*/

static void mdlInitializeSizes(SimStruct *S)
{
        ssSetNumSFcnParams(S, 3);                   /* Número de parámetros esperados: 1 */
        if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
     return;                                        /*Si faltan parametros se da mensaje */
   }


        ssSetNumContStates(    S, 0);               /* Número de estados continuos */
   ssSetNumDiscStates(    S, 0);                   /* Número de estados discretos */

   if (!ssSetNumInputPorts(S, 0)) return;           /* 0 puerto de entrada */

   if (!ssSetNumOutputPorts(S,1)) return;           /* 1 puerto de salida*/
   ssSetOutputPortWidth(S, 0, 3);                   /* dimensión 3*/
        ssSetOutputPortDataType(S, 0, SS_UINT8);
```

```
        ssSetNumSampleTimes(   S,  1);              /* Numero de muestreos */
    ssSetNumRWork(         S,  1);                  /* Vector de numeros reales */
    ssSetNumIWork(         S,  0);                  /* Vector de numeros enteros */
    ssSetNumPWork(         S,  0);                  /* Vector de punteros */
    ssSetNumModes(         S,  0);                  /* Vector de modos de trabajo */
    ssSetNumNonsampledZCs(S,   0);                  /* Número de paso por cero sin muestreo*/
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);

}


 /*
 * mdlInitializeSampleTimes - Inicializa muestreos
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
        // Periodo de muestreo
        real_T  tm     = mxGetPr(Tm(S))[0];


        if(tm!=-1){                                 //Tiempo discreto
                ssSetSampleTime (S, 0, tm);
                ssSetOffsetTime (S, 0, 0.0);
        }
        else{                                       //Tiempo continuo
                ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME
                ssSetOffsetTime(S, 0, 0.0);
        }
}


#define MDL_INITIALIZE_CONDITIONS
/*
 * mdlInitializeConditions - Inicializa estados y parametros del robot
 *
 */

static void mdlInitializeConditions(SimStruct *S)
{
        real_T  *rwork      = ssGetRWork(S);

        sec=0;    //Desablilita recepcion de datos

}

/*
 * mdlOutputs - Calcula las salidas
 *
 */

static void mdlOutputs(SimStruct *S, int_T tid)
{
        int i;                                                          //Para recorrer la cadena de bytes
        real_T  *rwork      = ssGetRWork(S);
        int8_T  rcarro;
        real_T  bias        = mxGetPr(BIAS(S))[0];
        real_T  tipo        = mxGetPr(TIPO(S))[0];
        uint8_T *y = (uint8_T *)ssGetOutputPortSignal(S,0);          /* Accede a la salida*/

        rcarro=13;
```

```
        if(sec==0)  //Se mandan retornos de carro
        {
                y[0]=rcarro;
                y[1]=rcarro;
                y[2]=rcarro;
        }
        else if (sec==1)                //Se configura bias
        {
                if (bias==1)            //Si el parámetro bias vale 1 se solicita bias al controlador
                {
                        y[0]='s';
                        y[1]='b';
                        y[2]=rcarro;
                }
                else if(bias==2)        //Si el parámetro bias vale 2 se desabilita el uso de bias
                {
                        y[0]='s';
                        y[1]='z';
                        y[2]=rcarro;
                }
                //Si bias vale cero, no se hace nada
        }
        else if (sec==2)                 //primera trama para configuración del tipo de dato
        {
                y[0]='c';
                y[1]='d';
                y[2]=' ';
        }
        else if (sec==3)                //segunda trama para configuracion del tipo de dato
        {
                if (tipo==1)
                        y[0]='r';       //Communicate Data Resolved
                else if (tipo==2)
                        y[0]='d';       //Data Decimal Gage

                y[1]=rcarro;            //Bytes sobrantes se rellenan con retornos de carro
                y[2]=rcarro;
        }

        //se prepara la secuencia siguiente
        if (sec>=3)
                sec=0;
        else
                sec=sec+1;

}



static void mdlTerminate(SimStruct *S)

{
}


#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
```

```
#endif



/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/



/*****************************************************************************/
/*****************************************************************************/
/** Nombre del Archivo: Sfuerz2.c          ******************************************/
/** Autor: Álvaro Payán Rodríguez          ******************************************/
/** Fecha: 20-07-2004                      ******************************************/
/*****************************************************************************/
/*****************************************************************************/
/** Descripción: función que se comunica con el sensor de fuerza a traves del
 *          puerto serie y le pide un flujo continuo de datos
 *
 *  4 salida:  bytes enviados por el puerto serie(4)
 *              error (1)
 *              fuerza(3)
 *              torsion(3)
 *
 *  2 entradas: bytes recibidos (54)
 *
 *
/*****************************************************************************/
/*****************************************************************************/


#define S_FUNCTION_NAME sfuerz2
#define S_FUNCTION_LEVEL 2

#include <math.h>
#include <stdlib.h>

#include "simstruc.h"


/* Variables del espacio de trabajo */
#define    sec          rwork[ 0]                    /* secuencia de envío */

#define    T(i)         (*uPtrs[i])                  /*Referencia a las entradas*/

/*Parametros*/
#define    Tm(S)       ssGetSFcnParam(S,0)          /*Tiempo de muestreo*/

static void mdlInitializeSizes(SimStruct *S)
{
          ssSetNumSFcnParams(S, 1);                 /* Número de parámetros esperados: 1 */
          if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
     return;                                        /*Si faltan parametros se da mensaje */
   }
```

```
        ssSetNumContStates(    S, 0);                  /* Número de estados continuos */
    ssSetNumDiscStates(    S, 0);                      /* Número de estados discretos */

    if (!ssSetNumInputPorts(S, 1)) return;             /* 1 puerto de entrada */
        ssSetInputPortWidth(S, 0, 54);                 /* con 54 coordenadas */
        ssSetInputPortDataType(S, 0,SS_UINT8);
        ssSetInputPortDirectFeedThrough(S, 0, 1); /*realimentacion directa desde la entrada */



    if (!ssSetNumOutputPorts(S,4)) return;             /* 1 puerto de salida*/
    ssSetOutputPortWidth(S, 0, 3);                     /* dimensión 3*/
        ssSetOutputPortWidth(S, 1, 1);                 /* error (1)*/
        ssSetOutputPortWidth(S, 2, 3);                 /* fuerza (3)*/
        ssSetOutputPortWidth(S, 3, 3);                 /* torsión (3)*/
        ssSetOutputPortDataType(S, 0, SS_UINT8);

        ssSetNumSampleTimes(   S,  1);                 /* Numero de muestreos */
    ssSetNumRWork(         S,  1);                     /* Vector de numeros reales */
    ssSetNumIWork(         S,  0);                     /* Vector de numeros enteros */
    ssSetNumPWork(         S,  0);                     /* Vector de punteros */
    ssSetNumModes(         S,  0);                     /* Vector de modos de trabajo */
    ssSetNumNonsampledZCs(S,   0);                     /* Número de paso por cero sin muestreo */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);

}


 /*
 * mdlInitializeSampleTimes - Inicializa muestreos
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
        // Periodo de muestreo
        real_T  tm     = mxGetPr(Tm(S))[0];


        if(tm!=-1){                                    //Tiempo discreto
                ssSetSampleTime (S, 0, tm);
                ssSetOffsetTime (S, 0, 0.0);
        }
        else{                                          //Tiempo continuo
                ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
                ssSetOffsetTime(S, 0, 0.0);
        }
}


#define MDL_INITIALIZE_CONDITIONS
/*
 * mdlInitializeConditions - Inicializa estados y parametros del robot
 *
 */

static void mdlInitializeConditions(SimStruct *S)
{
        real_T  *rwork      = ssGetRWork(S);

        sec=0;                                         //Desablilita recepcion de datos
```

```
}

/*
 * mdlOutputs - Calcula las salidas
 *
 */

static void mdlOutputs(SimStruct *S, int_T tid)
{
        int i;                                  //Para recorrer la cadena de bytes
        real_T  *rwork      = ssGetRWork(S);
        int8_T  dato;
        int j;                                  //Para la cadena auxiliar
        int s;                                  //Para saber la coordenada de salida que corresponde
        int k;
        int signo;
        char aux[8];                            //cadena auxiliar para conversión
        real_T *y2;
   uint8_T *y = (uint8_T *)ssGetOutputPortSignal(S,0);          /* Accede a la salida*/

        //InputPtrsType uPtrs;
        InputPtrsType uauxPtrs;
        InputInt8PtrsType uPtrs;

        uauxPtrs = ssGetInputPortSignalPtrs(S,0);               /* Puntero al vector de entrada*/
        uPtrs = (InputInt8PtrsType)uauxPtrs;


        if(sec==1)  //Se lee una linea
        {
                i=5;                                    //Nos posicionamos en el primer caracter válido
                /* Identifico el error*/
                for (k=0;k<7;k++)
                {
                        if (k==0)                       //Identifico y paso a la salida el error
                        {
                                s=0;
                                y2=ssGetOutputPortRealSignal(S,1);          /* Accede a la salida error
*/
                        }
                        else if (k==1)
                        {
                                s=0;
                                y2=ssGetOutputPortRealSignal(S,2);          /* Accede a la salida
fuerza */
                        }
                        else if (k==4)
                        {
                                s=0;
                                y2=ssGetOutputPortRealSignal(S,3);          /* Accede a la salida
torsión */
                        }

                        while (T(i)==32)                // mientras haya espacios se sigue leyendo
                                i=i+1;

                        signo=1;
                        if (T(i)==45)                   //número negativo
                        {
                                signo=-1;
```

```
                        i=i+1;                  //Se salta el caracter de signo negativo
                    }

                    j=0;
                    while (T(i)>47 && T(i)<58)    //Se leen caracteres mientras sean números
                    {
                        aux[j]=T(i);
                        j=j+1;
                        i=i+1;
                    }

                    aux[j]='\0';            //Caracter de final de cadena

                    if (T(i)==44)       //Lo siguiente es una coma y la saltamos para continuar
                        i=i+1;

                    if (j>0)                    //Se ha leido al menos un caracter numérico
                        y2[s]=signo*atoi(aux);
                    else
                        y2[s]=0;            //se pone la correspondiente salida a cero

                    s=s+1;
                }
            }

            //Se pide un dato
            dato=13;
            y[0]='q';
            y[1]='r';
            y[2]=dato;

            sec=1;              //Se habilita recepcion de datos

}

static void mdlTerminate(SimStruct *S)

{
}


#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

```
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
```

```
/*****************************************************************************/
/*****************************************************************************/
/** Nombre del Archivo: Sensor.c          ****************************************/
/** Autor: Álvaro Payán Rodríguez         ****************************************/
/** Fecha: 13-04-2005                     ****************************************/
/*****************************************************************************/
/*****************************************************************************/
/** Descripción: función que se encarga de la comunicación con el sensor de
 *    fuerza. Gestiona la multiplexión de las señales de las galgas y a su vez
 *    lee las correspondientes señales que el sensor facilita.
 *
 *  1 entrada: Señales de salida proporcionadas por el sensor (2) : +Sig, -Sig
 *
 *  3 salidas: Bias (2-->+Bias, -Bias), MUX (3-->A0,A1,A2), Gauges
 *          (6-->señales de galgas leidas)
 *
 *  Parámetros 4: Tiempo muestreo, Valores iniciales F/T (6), Bias + (6), Bias - (6)
 *
 *
/*****************************************************************************/
/*****************************************************************************/



#define S_FUNCTION_NAME sensor
#define S_FUNCTION_LEVEL 2

#include <math.h>

#include "simstruc.h"


#define    gact(i)      rwork[0+i]              /* Señales de galgas actuales*/
#define    bpos(i)      rwork[6+i]              /* Señales de bias +*/
#define    bneg(i)      rwork[12+i]             /* Señales de bias -*/
#define    gant(i)      rwork[18+i]             /* Señales de galgas antiguas*/



#define    FLAG_INIT    iwork[ 0]               /* Bandera para inicialización*/
#define    sec          iwork[ 1]               /* Variable para marcar la secuencia de envío*/

/* Entradas y estados */

#define    T(i)         (*uPtrs[i])


#define       Tm(S) ssGetSFcnParam(S,0)        /* Tiempo de muestreo*/
#define       V_INI(S) ssGetSFcnParam(S,1)     /* Valor inicial F/T */
#define       BIASpos(S) ssGetSFcnParam(S,2)   /* Valores de tensiones BIAS */
#define       BIASneg(S) ssGetSFcnParam(S,3)   /* Valores de tensiones BIAS */



static void mdlInitializeSizes(SimStruct *S)
{
        ssSetNumSFcnParams(S, 4);               /* Número de parámetros esperados: 4 */
        if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
```

```
    return;                                          /*Si faltan parametros se da mensaje */
  }


        ssSetNumContStates(    S, 0);                /* Número de estados continuos */
  ssSetNumDiscStates(    S, 0);                     /* Número de estados discretos */

  if (!ssSetNumInputPorts(S, 1)) return;           /* 1 puerto de entrada */
        ssSetInputPortWidth(S, 0, 2);              /* de dimensión 2*/
        ssSetInputPortDirectFeedThrough(S, 0, 1); /* todas las salidas con realimentacion directa desde
la entrada */


  if (!ssSetNumOutputPorts(S,3)) return;           /* 3 puertos de salida*/
  ssSetOutputPortWidth(S, 0, 2);                   /* bias(2) */
  ssSetOutputPortWidth(S, 1, 3);                   /* mux (3) */
        ssSetOutputPortWidth(S, 2, 6);             /* gauges(6) */

        ssSetNumSampleTimes(   S,  1);             /* Numero de muestreos */
  ssSetNumRWork(         S, 24);                   /* Vector de numeros reales */
  ssSetNumIWork(         S,  2);                   /* Vector de numeros enteros */
  ssSetNumPWork(         S,  0);                   /* Vector de punteros */
  ssSetNumModes(         S,  0);                   /* Vector de modos de trabajo */
  ssSetNumNonsampledZCs(S,   0);                   /* Número de paso por cero sin muestreo */
  ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}


/*
* mdlInitializeSampleTimes - Inicializa muestreos
*/

static void mdlInitializeSampleTimes(SimStruct *S)
{
        // Periodo de muestreo: parámetro 2:
        real_T  tm     = mxGetPr(Tm(S))[0];

        if(tm!=-1){                                //Tiempo discreto
                ssSetSampleTime (S, 0, tm);
                ssSetOffsetTime (S, 0, 0.0);
        }
        else{                                      //Tiempo continuo
                ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
                ssSetOffsetTime(S, 0, 0.0);
        }
}


#define MDL_INITIALIZE_CONDITIONS
/*
* mdlInitializeConditions - Inicializa estados y parametros del robot
*
*/

static void mdlInitializeConditions(SimStruct *S)
{

        int i;
        real_T  *rwork    = ssGetRWork(S);
        int_T   *iwork    = ssGetIWork(S);
```

```
        real_T  *biaspos = mxGetPr(BIASpos(S));          /* Puntero a Vector BIASpos */
        real_T  *biasneg = mxGetPr(BIASneg(S));          /* Puntero a Vector BIASneg */



        FLAG_INIT  = 0;                  /* Flag de inicialización a cero */
        sec=0;

        for(i=0;i<6;i++)                 //Valores de bias necesarios para la salida
        {
                bpos(i)=biaspos[i];
                bneg(i)=biasneg[i];
        }

}


/*****************************************************************/
/*
 * mdlOutputs - Calcula las salidas
 *
 */

static void mdlOutputs(SimStruct *S, int_T tid)
{
        int i;
        real_T *b;                               //Bias
        real_T *g;                               //Galgas
        real_T *m;                               //Medidas finales
        real_T *rwork = ssGetRWork(S);
        int_T   *iwork    = ssGetIWork(S);

        InputRealPtrsType uPtrs;
        real_T  *v_ini = mxGetPr(V_INI(S));      /* Puntero a Vector V_INI */


        b=ssGetOutputPortRealSignal(S,0);        /* Accede a la salida de BIAS vía un puntero */
        m=ssGetOutputPortRealSignal(S,1);        /* Accede a la salida de mux vía un puntero */
        g=ssGetOutputPortRealSignal(S,2);        /* Accede a la salida de galgas vía un puntero */

        if (FLAG_INIT == 0) {
                /* Se escriben en la salida los valores iniciales establecidos*/

                for(i=0;i<6;i++)
                {
                        g[i] = v_ini[i];         /* Se asigna la posición inicial */
                        gant(i)=v_ini[i];
                        gact(i)=v_ini[i];
                }


                /* Se dejan inicialmente las señales de multiplexión para galga0*/
                for(i=0;i<3;i++)
                        m[i]=1;

                /*Señales de BIAS correspondientes a la galga 0;*/
                b[0]=bpos(5);
                b[1]=bneg(5);

                FLAG_INIT = 1;                   /* Inicializacion realizada */
```

```
}

else{
        /* Procedimiento normal, se recorren todas las galgas en 6 pasos*/

        uPtrs = ssGetInputPortRealSignalPtrs(S,0);  /*puntero al vector de entradas*/

        switch(sec){
                case 0:
                        /* Galga 0*/
                        m[0]=0;                 //multiplexión galga 0
                        m[1]=0;
                        m[2]=0;
                        b[0]=bpos(0);           //bias
                        b[1]=bneg(0);


                        gact(5)=T(0)-T(1);      // Valor de galga= (+Sig) - (-Sig)
                        break;
                case 1:
                        /* Galga 1*/
                        m[0]=1;                 //multiplexión galga 1
                        m[1]=0;
                        m[2]=0;
                        b[0]=bpos(1);           //bias
                        b[1]=bneg(1);


                        gact(0)=T(0)-T(1);
                        break;
                case 2:
                        /* Galga 2*/
                        m[0]=1;                 //multiplexión galga 2
                        m[1]=1;
                        m[2]=0;
                        b[0]=bpos(2);           //bias
                        b[1]=bneg(2);


                        gact(1)=T(0)-T(1);
                        break;
                case 3:
                        /* Galga 3*/
                        m[0]=0;                 //multiplexión galga 3
                        m[1]=1;
                        m[2]=0;
                        b[0]=bpos(3);           //bias
                        b[1]=bneg(3);


                        gact(2)=T(0)-T(1);
                        break;
                case 4:
                        /* Galga 4*/
                        m[0]=0;                 //multiplexión galga 4
                        m[1]=1;
                        m[2]=1;
                        b[0]=bpos(4);           //bias
                        b[1]=bneg(4);
```

```
                gact(3)=T(0)-T(1);

                break;
            case 5:
                /* Galga 5*/
                m[0]=1;                 //multiplexión galga 5
                m[1]=1;
                m[2]=1;
                b[0]=bpos(5);           //bias
                b[1]=bneg(5);

                gact(4)=T(0)-T(1);

                //Actualización de las variables de salida
                gant(0)=gact(0);
                gant(1)=gact(1);
                gant(2)=gact(2);
                gant(3)=gact(3);
                gant(4)=gact(4);
                gant(5)=gact(5);
                break;
            default:
                break;
        }

        //Salidas
        g[0]=gant(0);
        g[1]=gant(1);
        g[2]=gant(2);
        g[3]=gant(3);
        g[4]=gant(4);
        g[5]=gant(5);

        sec=sec+1;
        if(sec>5)
            sec=0;

    }

}


static void mdlTerminate(SimStruct *S)
{
}


#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif




/*********************************************************************************/
/*********************************************************************************/
/*********************************************************************************/
```

```
/*****************************************************************************/
/*****************************************************************************/
/** Nombre del Archivo: Bias.c              ******************************************/
/** Autor: Álvaro Payán Rodríguez           ******************************************/
/** Fecha: 01-09-2005                       ******************************************/
/*****************************************************************************/
/*****************************************************************************/
/** Descripción: función que ofrece la posibilidad de realizar la función de bias mediante software.
 *                  Cuando en su entrada de seleccion aparezca un flanco de subida se almacenarán
 *                  datos de bias para ser restados en medidas posteriores. Cuando en su entrada de
 *                  selección aparezca un flanco de bajada se eliminarán datos de bias almacenados
 *                  y la función de bias dejará de actuar como tal.
 *
 *  2 entradas: Señal de selección de aplicación (1),
 *                Señales de fuerza y torsión leidas (6)
 *
 *  1 salida:  Señales de fuerza y torsión a las que se ha aplicado la función
 *          de bias (6)
 *
 *  Parámetros 2: Tiempo muestreo, Valores iniciales F/T (6)
 *
 *
/*****************************************************************************/
/*****************************************************************************/



#define S_FUNCTION_NAME bias
#define S_FUNCTION_LEVEL 2

#include <math.h>

#include "simstruc.h"


#define    ftbias(i)      rwork[0+i]          /* Valores de bias que se restarán*/
#define    selant         rwork[6]            /* Valor de la señal de seleccion en el periodo anterior*/



#define             FLAG_INIT     iwork[ 0]


/* Entradas y estados */

#define    T(i)         (*uPtrs[i])              /*Referencia a puertos de entrada*/


#define                     Tm(S) ssGetSFcnParam(S,0)        /* Tiempo de muestreo*/
#define                     V_INI(S) ssGetSFcnParam(S,1)     /* Valor inicial F/T */


static void mdlInitializeSizes(SimStruct *S)
{
        ssSetNumSFcnParams(S, 2);                           /* Número de parámetros esperados: 2 */
        if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
      return;                                               /*Si faltan parametros se da mensaje */
```

```
    }


        ssSetNumContStates(    S, 0);                   /* Número de estados continuos */
    ssSetNumDiscStates(    S, 0);                       /* Número de estados discretos */

    if (!ssSetNumInputPorts(S, 2)) return;              /* 1 puerto de entrada */
        ssSetInputPortWidth(S, 0, 1);                   /* de dimensión 1*/
        ssSetInputPortWidth(S, 1, 6);                   /* de dimensión 6*/
        ssSetInputPortDirectFeedThrough(S, 0, 1);       /* todas las salidas con realimentacion
directa desde la entrada */
        ssSetInputPortDirectFeedThrough(S, 1, 1);


    if (!ssSetNumOutputPorts(S,1)) return;              /* 1 puerto de salida*/
    ssSetOutputPortWidth(S, 0, 6);                      /* señales de fuerza y torsión(2) */


    ssSetNumSampleTimes(   S,  1);                      /* Numero de muestreos */
    ssSetNumRWork(         S,  7);                      /* Vector de numeros reales */
    ssSetNumIWork(         S,  1);                      /* Vector de numeros enteros */
    ssSetNumPWork(         S,  0);                      /* Vector de punteros */
    ssSetNumModes(         S,  0);                      /* Vector de modos de trabajo */
    ssSetNumNonsampledZCs(S,   0);                      /* Número de paso por cero sin muestreo */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}


/*
* mdlInitializeSampleTimes - Inicializa muestreos
*/

static void mdlInitializeSampleTimes(SimStruct *S)
{
        // Periodo de muestreo: parámetro 2:
        real_T  tm     = mxGetPr(Tm(S))[0];

        if(tm!=-1){                                     //Tiempo discreto
                ssSetSampleTime (S, 0, tm);
                ssSetOffsetTime (S, 0, 0.0);
        }
        else{                                           //Tiempo continuo
                ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
                ssSetOffsetTime(S, 0, 0.0);
        }
}


#define MDL_INITIALIZE_CONDITIONS
/*
* mdlInitializeConditions - Inicializa estados y parametros
*
*/

static void mdlInitializeConditions(SimStruct *S)
{

        int i;
        real_T  *rwork      = ssGetRWork(S);
        int_T   *iwork    = ssGetIWork(S);
```

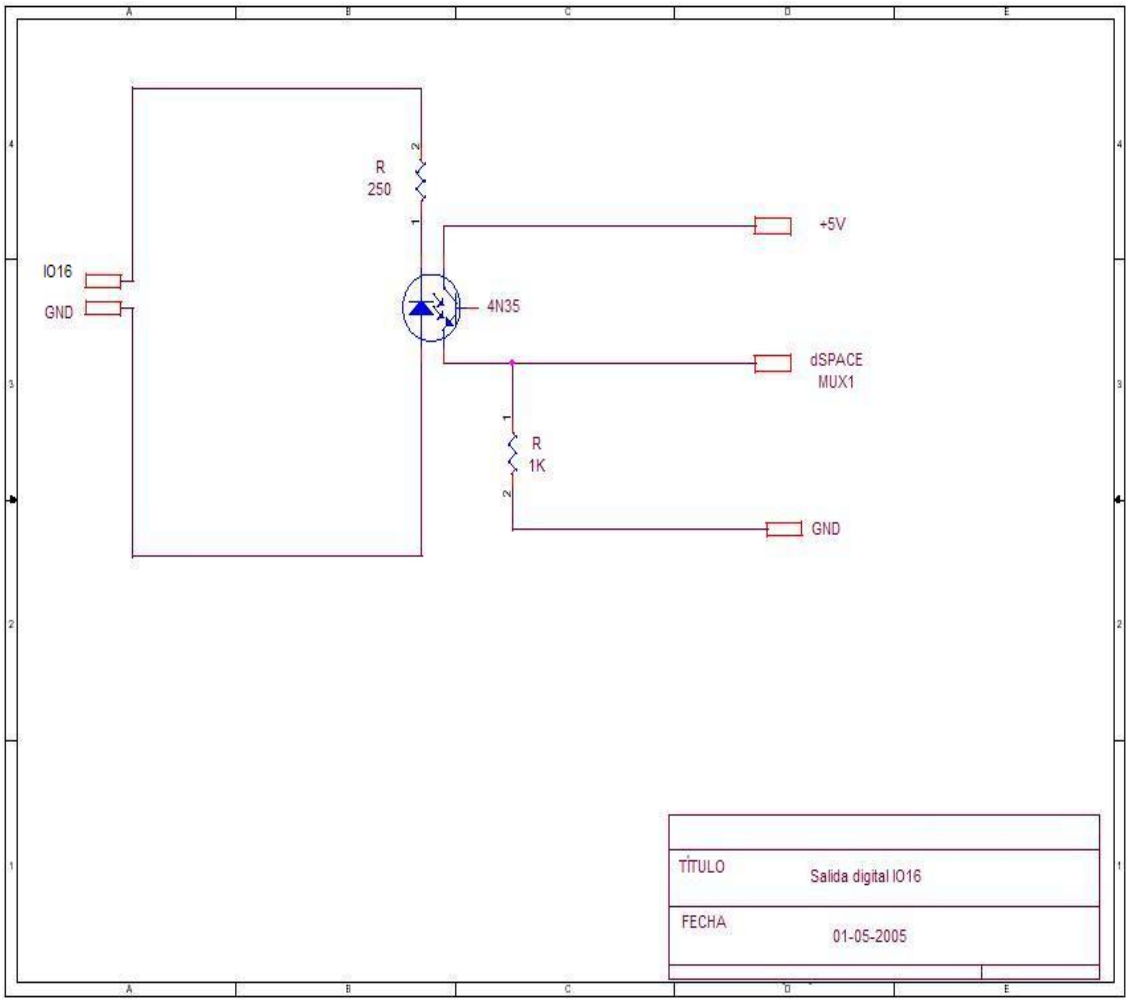
```
        FLAG_INIT  = 0;                         /* Flag de inicialización a cero */


        for(i=0;i<6;i++)                        //Valores de bias necesarios para la salida
                ftbias(i)=0;

        selant=0;


}


/*********************************************************/
/*
 * mdlOutputs - Calcula las salidas
 *
 */

static void mdlOutputs(SimStruct *S, int_T tid)
{
        int i;
        real_T *y;
        real_T *rwork = ssGetRWork(S);
        int_T   *iwork    = ssGetIWork(S);

        InputRealPtrsType uPtrs;
        real_T  *v_ini = mxGetPr(V_INI(S));             /* Puntero a Vector V_INI */



        y=ssGetOutputPortRealSignal(S,0);               /* Accede a la salida vía un puntero */


        if (FLAG_INIT == 0) {
                /* Se escriben en la salida los valores iniciales establecidos*/

                for(i=0;i<6;i++)
                        y[i] = v_ini[i];                /* Se asigna la posición inicial */

                FLAG_INIT = 1;                          /* Inicializacion realizada */
        }

        else{
                /* Procedimiento normal*/

                uPtrs = ssGetInputPortRealSignalPtrs(S,0);  /*puntero al vector de entradas 1*/

                if(T(0)==0)
                {
                        if(selant==1)                   //flanco de bajada
                        {
                                for(i=0;i<6;i++)
                                        ftbias(i)=0;
                                selant=0;
                        }
                }
                else
```

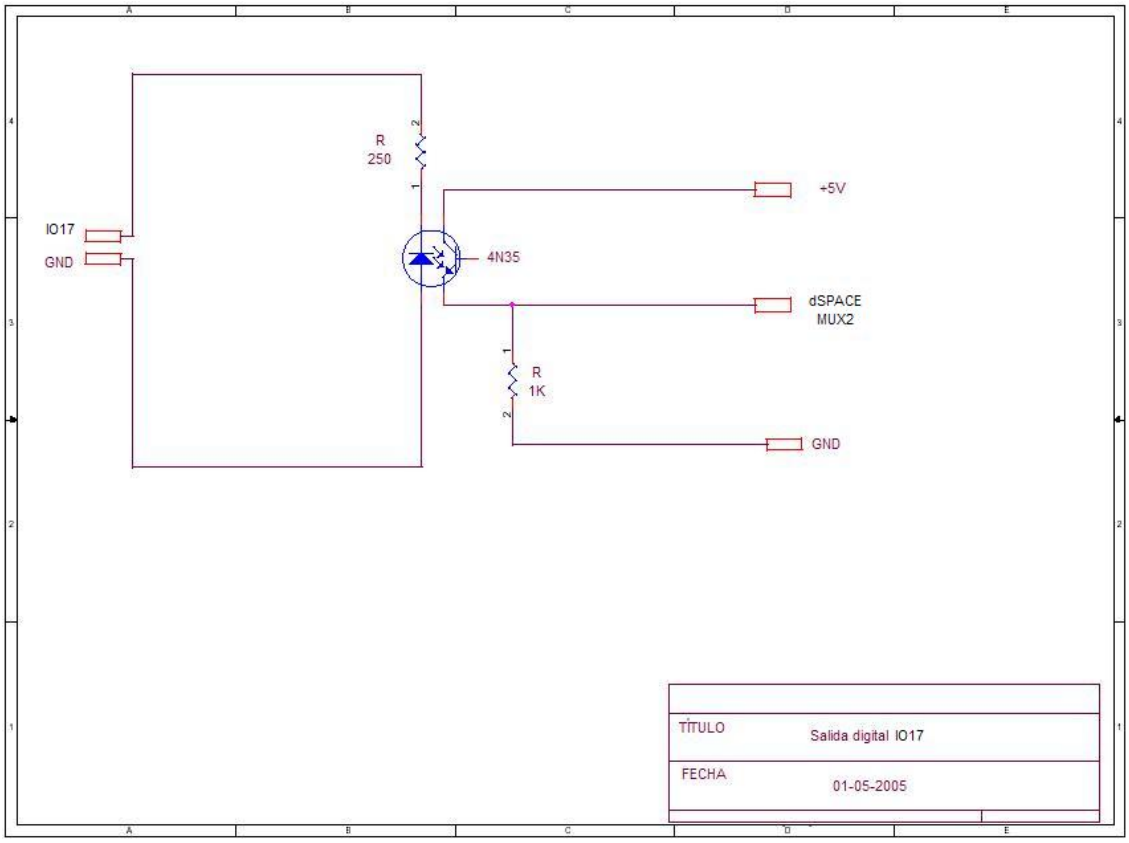
```
                {
                        if(selant==0)                           //flanco de subida
                        {
                                uPtrs = ssGetInputPortRealSignalPtrs(S,1);  /*puntero al vector de
entradas 2*/

                                for(i=0;i<6;i++)
                                        ftbias(i)=T(i);

                                selant=1;
                        }
                }
                //Salidas
                uPtrs = ssGetInputPortRealSignalPtrs(S,1);  /*puntero al vector de entradas 2*/
                y[0]=T(0)-ftbias(0);
                y[1]=T(1)-ftbias(1);
                y[2]=T(2)-ftbias(2);
                y[3]=T(3)-ftbias(3);
                y[4]=T(4)-ftbias(4);
                y[5]=T(5)-ftbias(5);

        }

}


static void mdlTerminate(SimStruct *S)
{
}


#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif




/*********************************************************************************/
/*********************************************************************************/
/*********************************************************************************/




/*********************************************************************************/
/*********************************************************************************/
/** Nombre del Archivo: Sensor.m          ******************************************/
/** Autor: Álvaro Payán Rodríguez         ******************************************/
/** Fecha: 01-08-2005                     ******************************************/
/*********************************************************************************/
/*********************************************************************************/
/** Descripción: función de Matlab que se encarga de inicializar los valores de las constantes que
 *                      se usarán para la implementación del protocolo de control.
 *
/*********************************************************************************/
/*********************************************************************************/
```

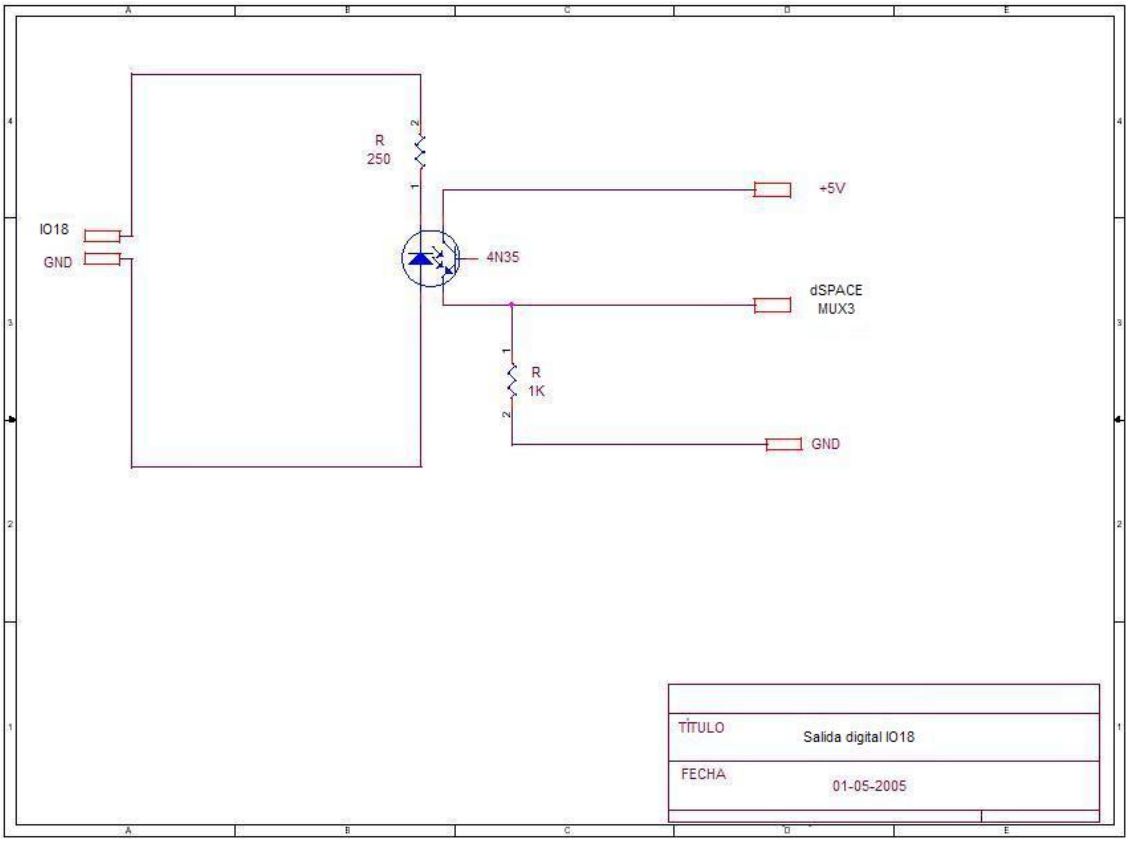
```
%Definición de la matriz de calibracion

cal=[-0.007753  0.004112 -0.018775  0.650968 -0.028382 -0.652283;
    -0.010183 -0.749223 -0.004976  0.381134  0.006373  0.374143;
    -1.183820  0.017224 -1.174941  0.012184 -1.178809 -0.014301;
    0.016062 -0.352207  1.340955  0.165291 -1.334350  0.158848;
    -1.553802  0.005106  0.804629 -0.306570  0.785211  0.326036;
     0.014600  0.810695 -0.020290  0.799408  0.027209  0.788916];

biaslevel=[162 200 197 205 194 150];

cinco=[5 5 5 5 5 5];

biaspos=(10/255)*biaslevel - cinco;
biasneg=(-10/255)*biaslevel + cinco;

%Valores iniciales de fuerzas y torsiones
vini=[0,0,0,0,0,0];

%Tiempo de muestreo
tm=0.00001;
```

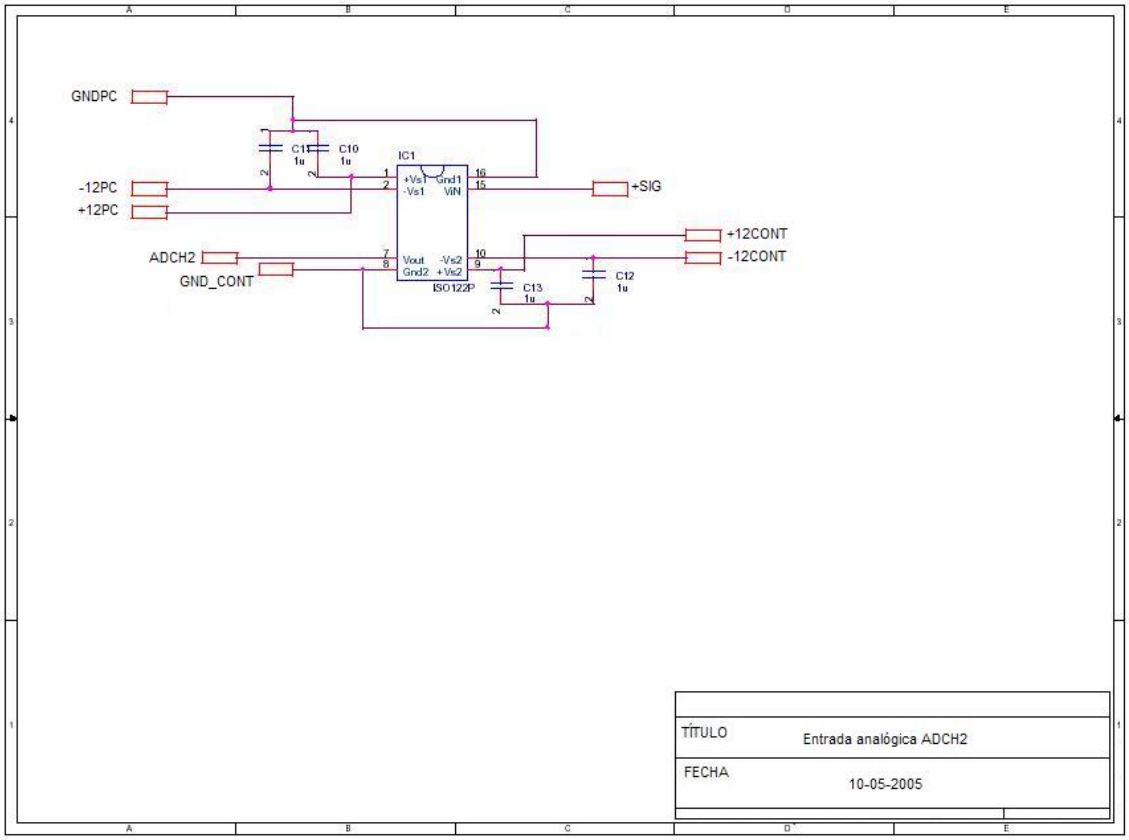
APÉNDICE B

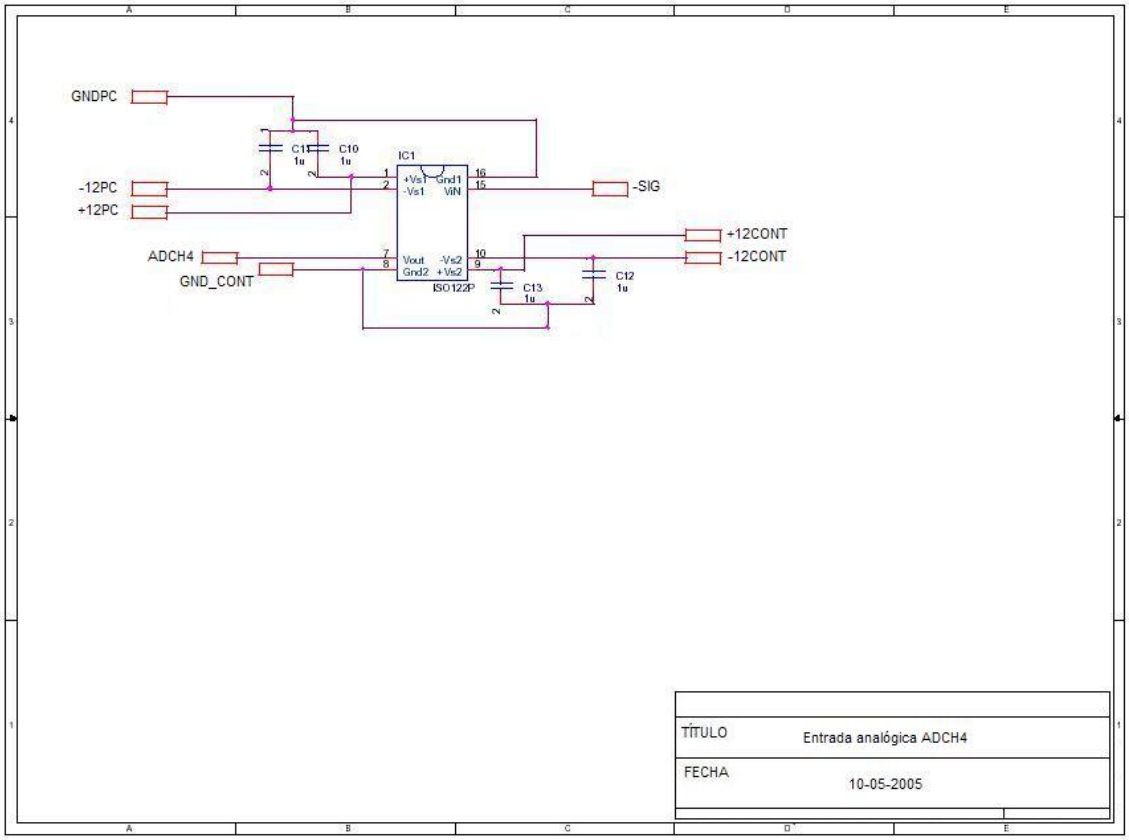
TARJETA DE PROTECCIÓN (ESQUEMAS ELÉCTRICOS)

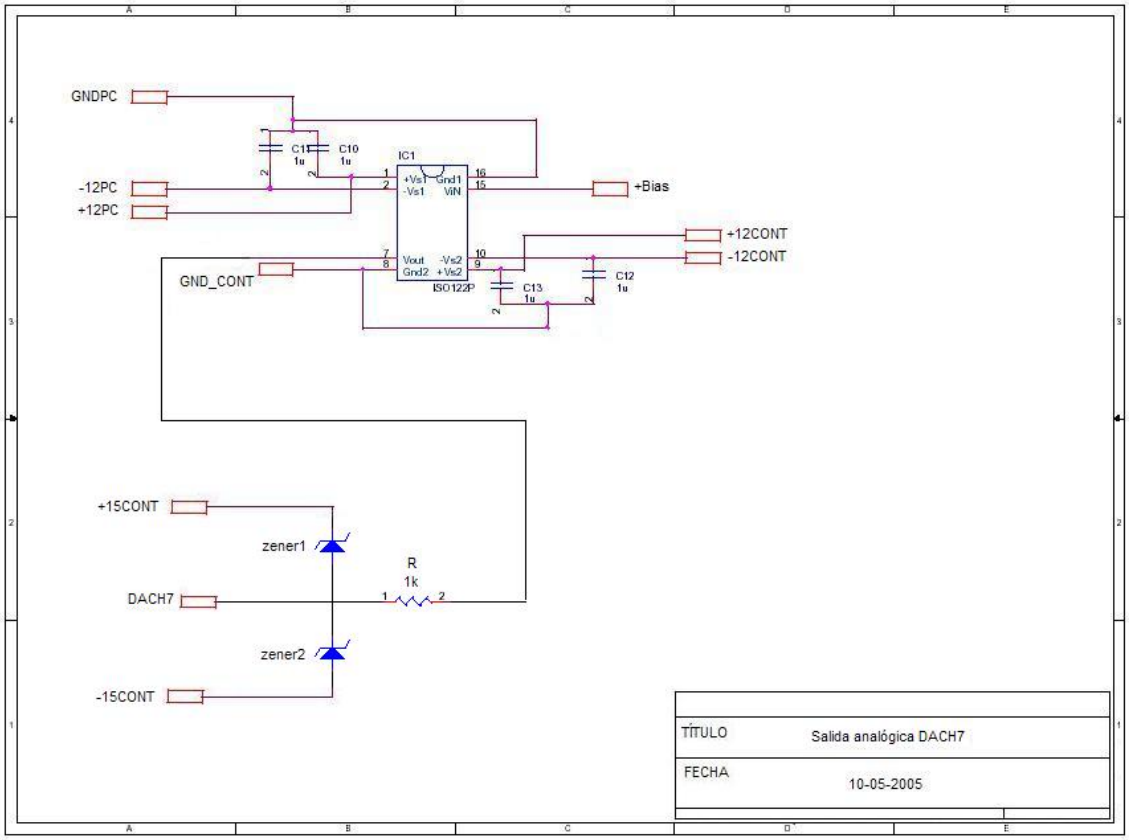

R
250
+5V
IO16
GND
4N35
dSPACE
MUX1
R
1K
GND
TÍTULO
Salida digital IO16
FECHA
01-05-2005

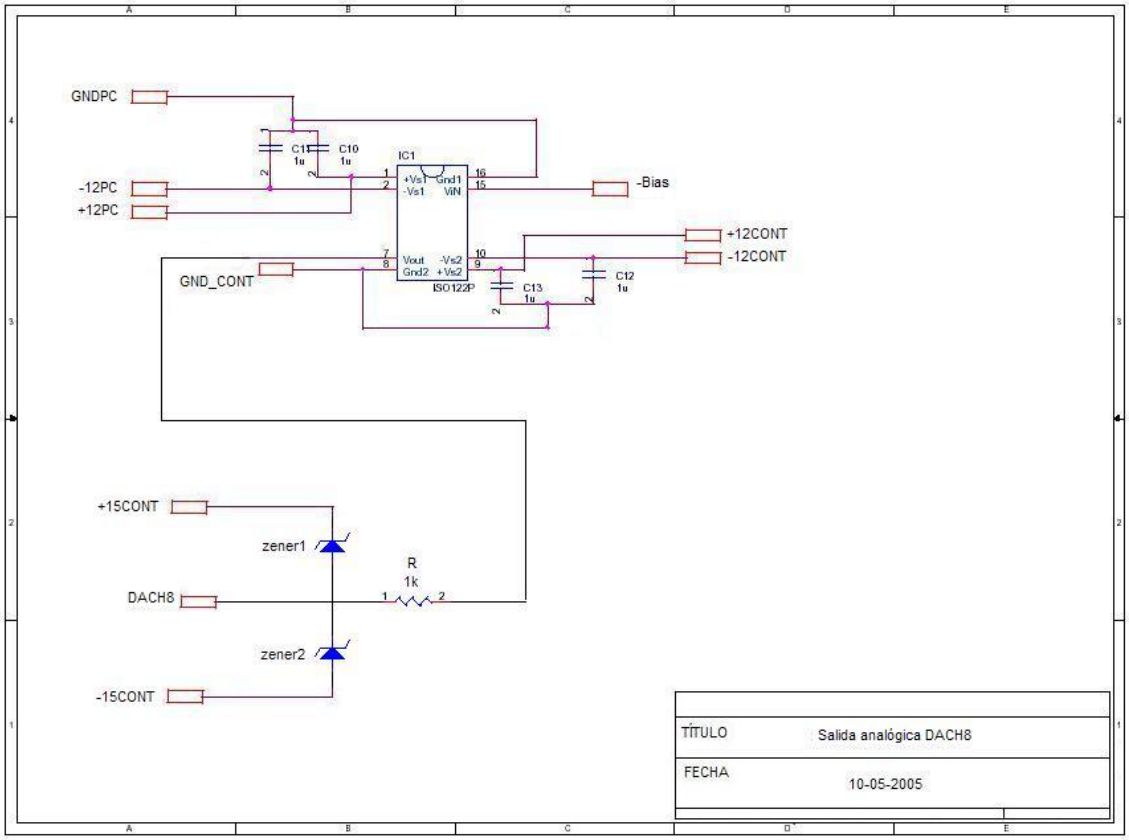

R
250
IO17
GND
4N35
+5V
dSPACE
MUX2
R
1K
GND
TÍTULO
Salida digital IO17
FECHA
01-05-2005


R
250
IO18
GND
4N35
+5V
dSPACE
MUX3
R
1K
GND
TÍTULO
Salida digital IO18
FECHA
01-05-2005

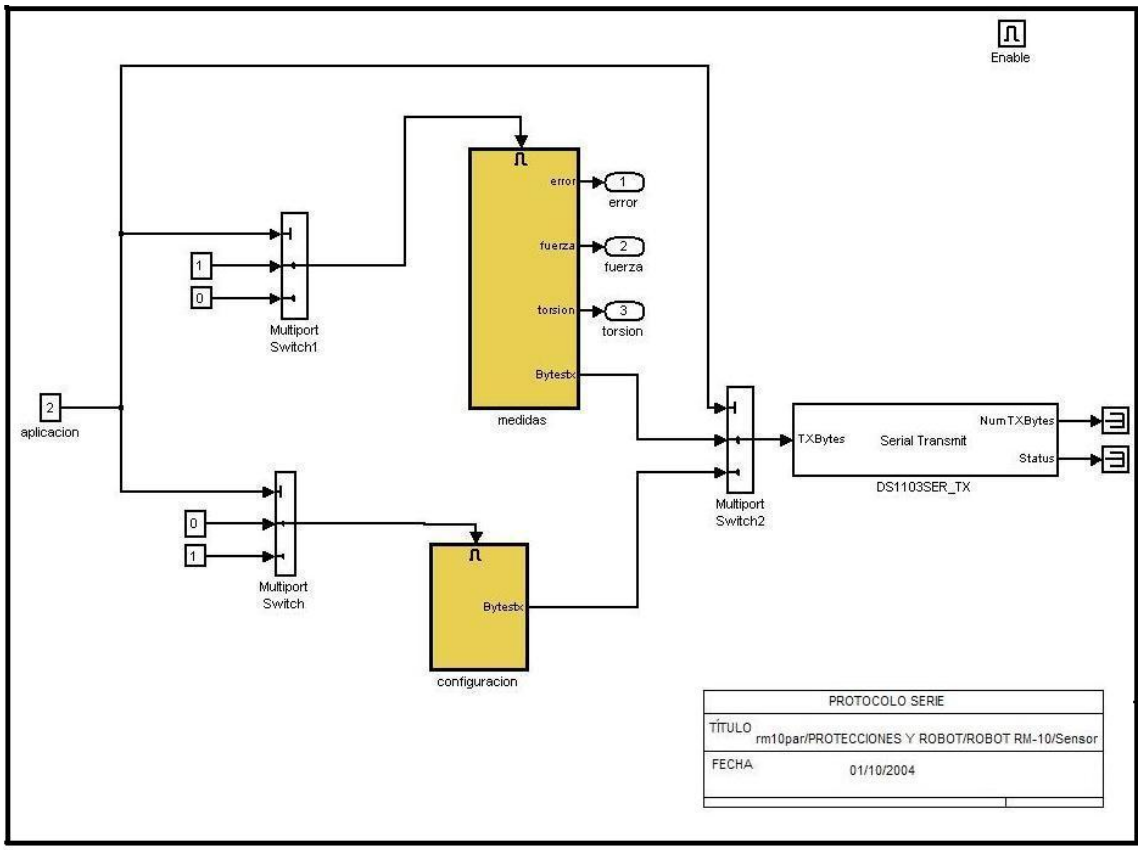

GNDPC
-12PC
+12PC
ADCH2
GND_CONT
C11
1u
C10
1u
IC1
+Vs1
-Vs1
Gnd1
ViN
Vout
Gnd2
-Vs2
+Vs2
ISO122P
C13
1u
C12
1u
+SIG
+12CONT
-12CONT
TÍTULO
Entrada analógica ADCH2
FECHA
10-05-2005

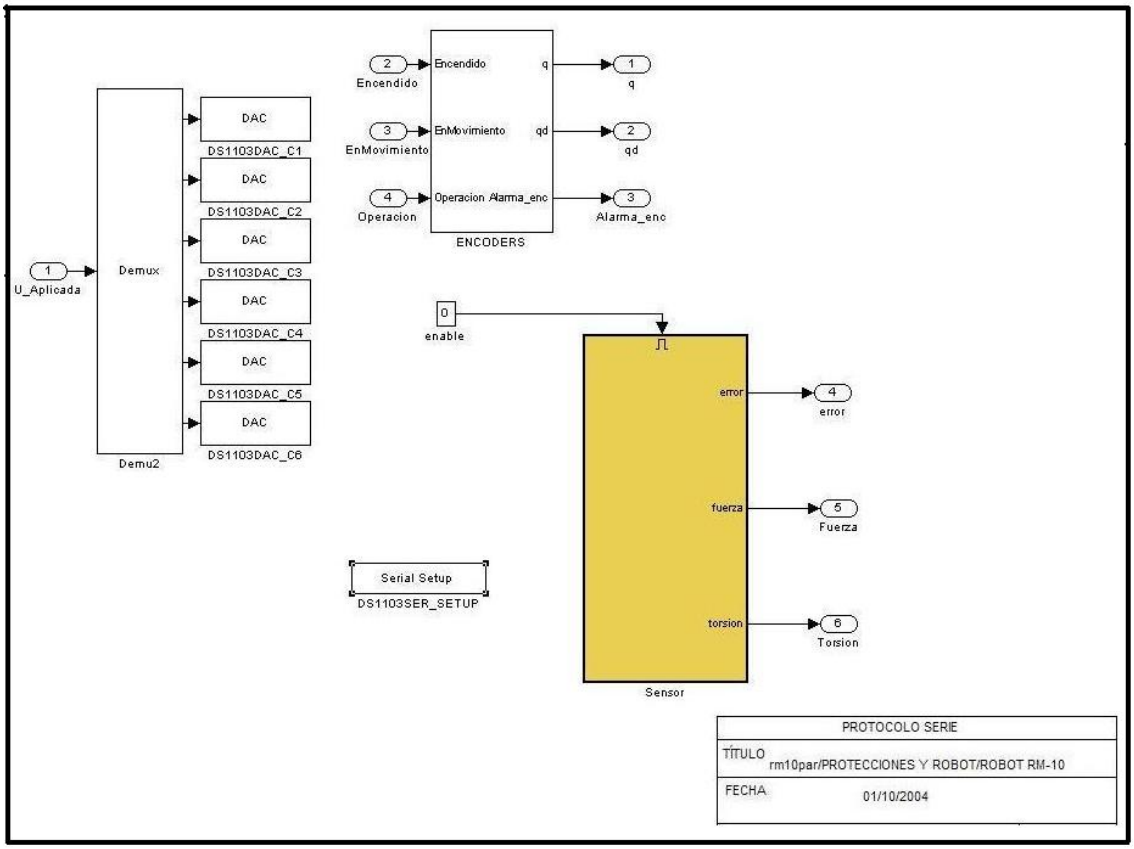

GNDPC
-12PC
+12PC
ADCH4
GND_CONT
C11
1u
C10
1u
IC1
+Vs1
-Vs1
Gnd1
ViN
Vout
Gnd2
-Vs2
+Vs2
ISO122P
C13
1u
C12
1u
-SIG
+12CONT
-12CONT
TÍTULO
Entrada analógica ADCH4
FECHA
10-05-2005

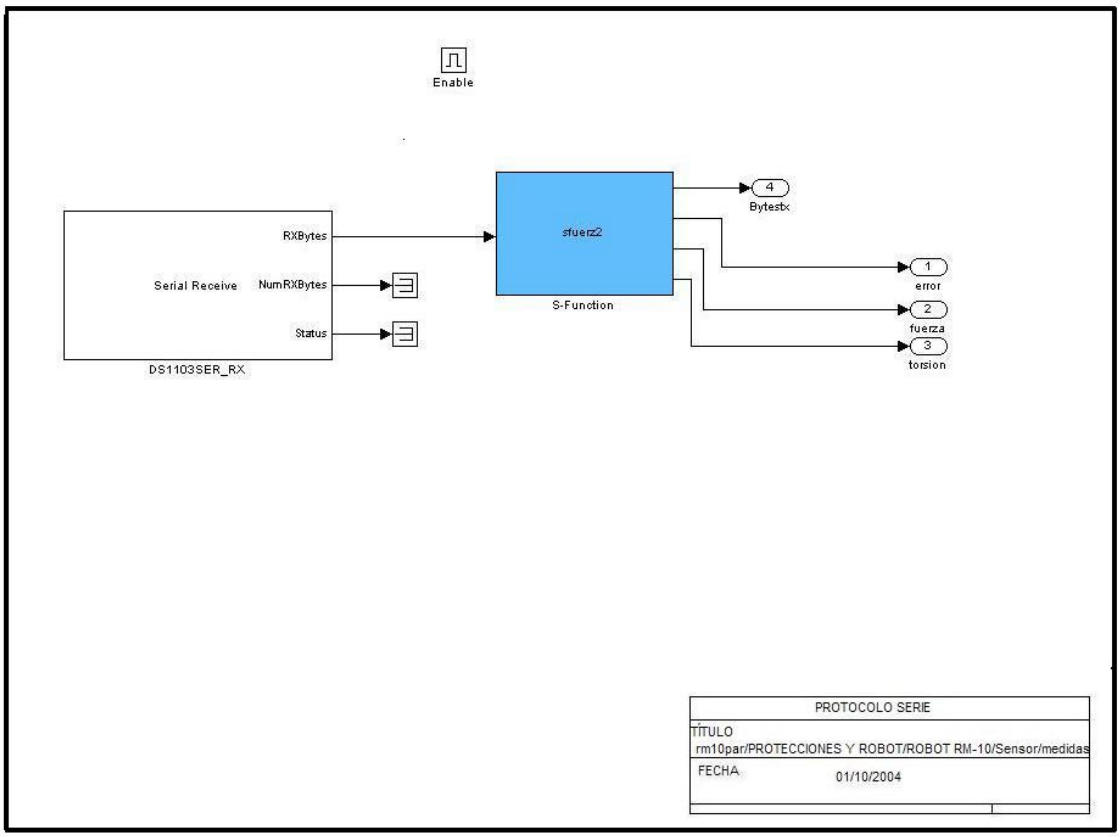

GNDPC
-12PC
+12PC
C11
1u
C10
1u
IC1
+Vs1
-Vs1
Gnd1
ViN
+Bias
+12CONT
-12CONT
GND_CONT
Vout
Gnd2
-Vs2
+Vs2
ISO122P
C13
1u
C12
1u
+15CONT
zener1
R
1k
DACH7
zener2
-15CONT
TÍTULO
Salida analógica DACH7
FECHA
10-05-2005

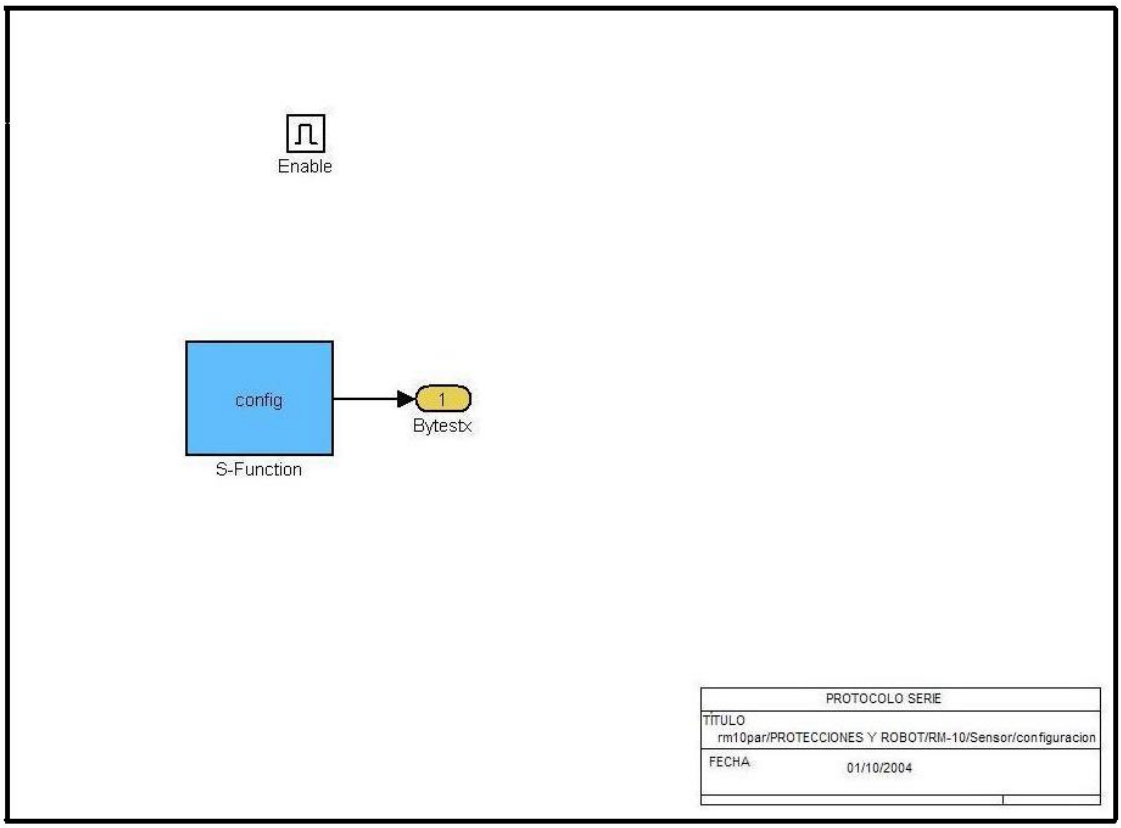

GNDPC
-12PC
+12PC
C11
1u
C10
1u
IC1
+Vs1
-Vs1
Gnd1
ViN
-Bias
+12CONT
-12CONT
GND_CONT
Vout
Gnd2
-Vs2
+Vs2
ISO122P
C13
1u
C12
1u
+15CONT
zener1
R
1k
DACH8
zener2
-15CONT
TÍTULO
Salida analógica DACH8
FECHA
10-05-2005

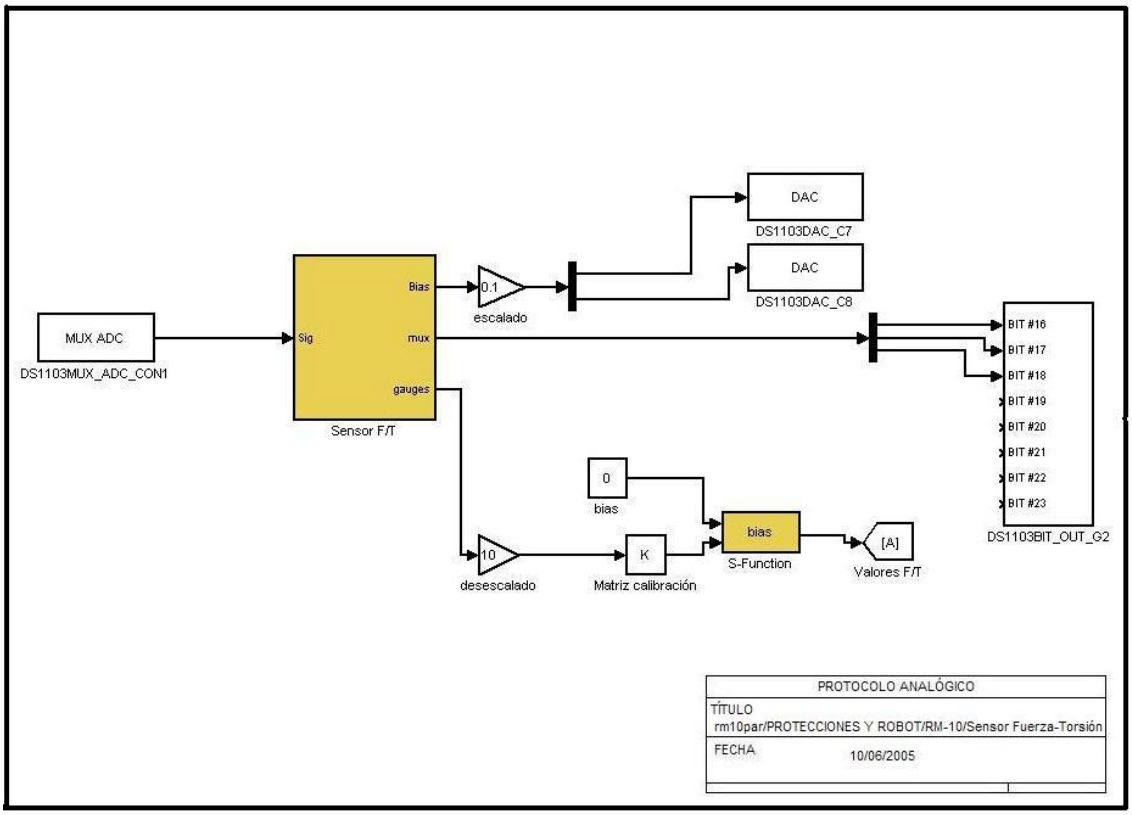
DIAGRAMAS DE BLOQUES DE SIMULINK

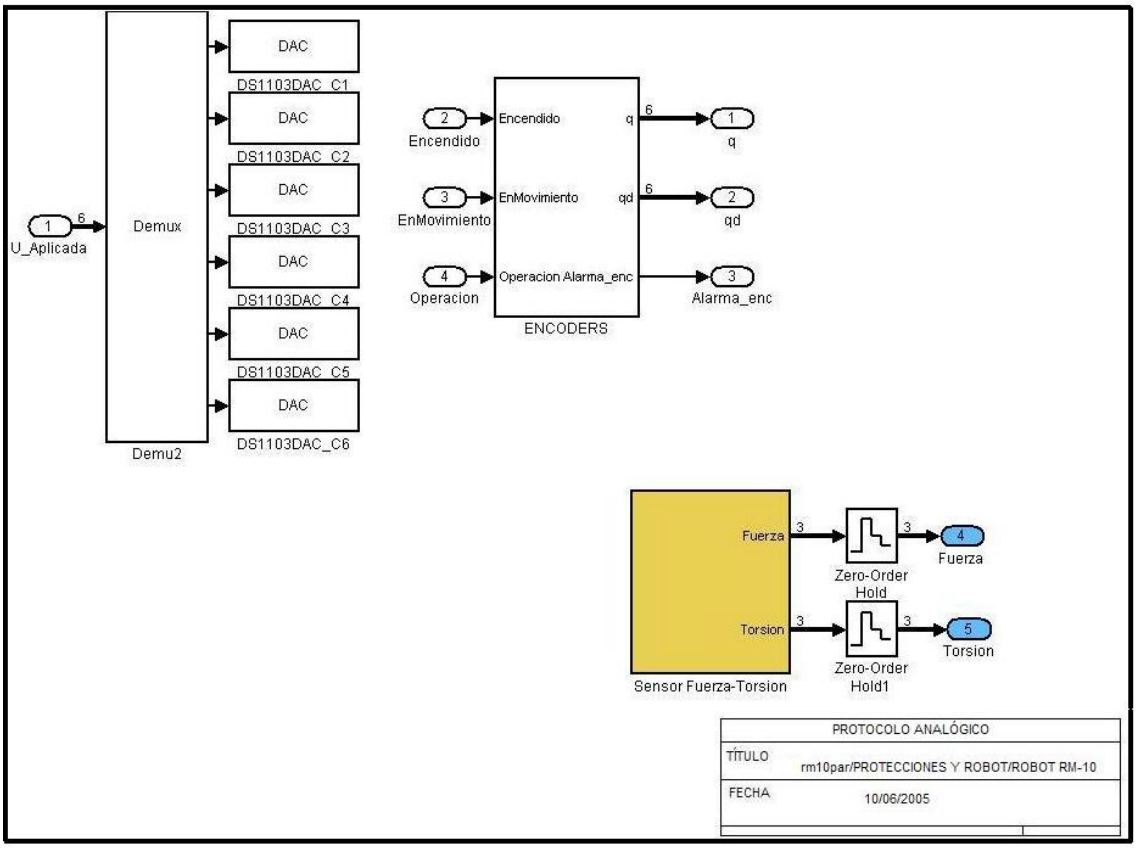

Enable
error
1
error
fuerza
2
fuerza
torsion
3
torsion
Bytestx
medidas
1
0
Multiport
Switch1
2
aplicacion
0
1
Multiport
Switch
Bytestx
configuracion
Multiport
Switch2
TXBytes
Serial Transmit
NumTXBytes
Status
DS1103SER_TX
PROTOCOLO SERIE
TÍTULO
rm10par/PROTECCIONES Y ROBOT/ROBOT RM-10/Sensor
FECHA
01/10/2004


1
U_Aplicada
Demux
DAC
DS1103DAC_C1
DAC
DS1103DAC_C2
DAC
DS1103DAC_C3
DAC
DS1103DAC_C4
DAC
DS1103DAC_C5
DAC
DS1103DAC_C6
Demu2
2
Encendido
3
EnMovimiento
4
Operacion
Encendido
EnMovimiento
Operacion
q
qd
Alarma_enc
1
q
2
qd
3
Alarma_enc
ENCODERS
0
enable
error
4
error
fuerza
5
Fuerza
torsion
6
Torsion
Sensor
Serial Setup
DS1103SER_SETUP
PROTOCOLO SERIE
TÍTULO
rm10par/PROTECCIONES Y ROBOT/ROBOT RM-10
FECHA
01/10/2004


Enable
RXBytes
Serial Receive
NumRXBytes
Status
DS1103SER_RX
sfuerz2
S-Function
4
Bytestx
1
error
2
fuerza
3
torsion
PROTOCOLO SERIE
TÍTULO
rm10par/PROTECCIONES Y ROBOT/ROBOT RM-10/Sensor/medidas
FECHA
01/10/2004


Enable
config
S-Function
1
Bytestx
PROTOCOLO SERIE
TÍTULO
rm10par/PROTECCIONES Y ROBOT/RM-10/Sensor/configuracion
FECHA
01/10/2004


DAC
DS1103DAC_C7
DAC
DS1103DAC_C8
MUX ADC
DS1103MUX_ADC_CON1
Sig
Bias
mux
gauges
Sensor F/T
0.1
escalado
BIT #16
BIT #17
BIT #18
BIT #19
BIT #20
BIT #21
BIT #22
BIT #23
DS1103BIT_OUT_G2
0
bias
bias
S-Function
[A]
Valores F/T
10
desescalado
K
Matriz calibración
PROTOCOLO ANALÓGICO
TÍTULO
rm10par/PROTECCIONES Y ROBOT/RM-10/Sensor Fuerza-Torsión
FECHA
10/06/2005


1
U_Aplicada
6
Demux
DAC
DS1103DAC_C1
DAC
DS1103DAC_C2
DAC
DS1103DAC_C3
DAC
DS1103DAC_C4
DAC
DS1103DAC_C5
DAC
DS1103DAC_C6
Demu2
2
Encendido
3
EnMovimiento
4
Operacion
Encendido
EnMovimiento
Operacion
q
qd
Alarma_enc
6
6
1
q
2
qd
3
Alarma_enc
ENCODERS
Fuerza
Torsion
Sensor Fuerza-Torsion
3
3
Zero-Order Hold
Zero-Order Hold1
4
Fuerza
5
Torsion
PROTOCOLO ANALÓGICO
TÍTULO
rm10par/PROTECCIONES Y ROBOT/ROBOT RM-10
FECHA
10/06/2005