Interfacing the HPC and LM629 for Motion Control

National Semiconductor
Application Note 868
Santanu Roy
January 1993

INTRODUCTION

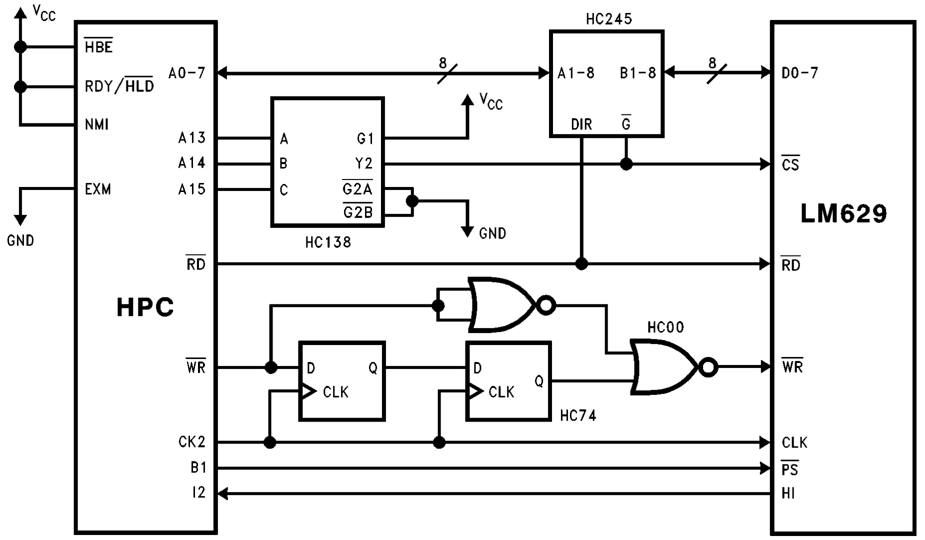
Control of servo motors has a wide range of applications including industrial control, factory automation, position and velocity servomechanism, and robotics. The basic tasks involved in such motion controls include position, velocity, and acceleration measurement, implementation of PID algorithm, detection of overrun and stress conditions, and communication back to a central controller. The HPC high performance microcontroller in conjunction with LM629 motion controller chip provides a solution for handling these required control tasks with extremely high degree of precision and very little software overhead.

The HPC is a highly integrated and high performance member of the HPC™ family of National's microcontroller. The availability of a wide variety of on-chip peripherals such as high speed timers, high speed I/Os, input capture, A/D, PWM, UART and MICROWIRE™ provides a flexible architecture for a wide variety of high performance applications at a reasonable cost. The LM629 is a National's dedicated motion-control processor designed for use with a variety of D.C. and brushless D.C. servomotors which provide a quadrature incremental position feedback signal for close-loop operation. The LM629 stand alone can perform all the intensive, real-time computational tasks required for high performance digital motion control. As a result in a multi-task system using the HPC where one of the tasks requires precision servo control can be achieved with this chip-set with minimal software overhead and very little CPU time.

THE INTERFACE DESIGN

Figure 1 shows the interface between the HPC and the LM629 via an 8-bit parallel bus. As shown in the figure, the HPC (host controller) interfaces with the LM629 data bus (D0–D7) on the lower half of PORTA (MUXed address/data bus). The HPC in this application, is used in 8-bit Expanded-Normal mode (strapping the $\overline{\text{HBE}}$ pin to V_{CC}, EXM pin to ground and EA bit in the PSW set to a one). The advantage of such a configuration is the upper half of the address/data bus (PORTA bits 8:15) will latch the address which is used to generate the required chip select for the LM629. This eliminates the requirement of an address latch for the decoder logic.

The delay and the logic associated with HPC write strobe ($\overline{\text{WR}}$) is added to increase the write-data hold time (as viewed from the LM629 end) effectively causing the $\overline{\text{WR}}$ pulse to rise early. The LM629 clock is provided by the CK2 output of the HPC. Since the LM629 has a maximum running speed of 8 MHz, the fastest HPC could run in such applications will be 16 MHz. The 74HC245 is used to decrease the read-data hold time), which is necessary when interfacing HPC with slower peripherals.



TL/DD/11770–1

FIGURE 1. Interface between the HPC and LM629

HPC™ and MICROWIRE™ are trademarks of National Semiconductor Corporation.

Interfacing the HPC and LM629 for Motion Control
AN-868

TL/DD11770

RRD-B30M75/Printed in U. S. A.

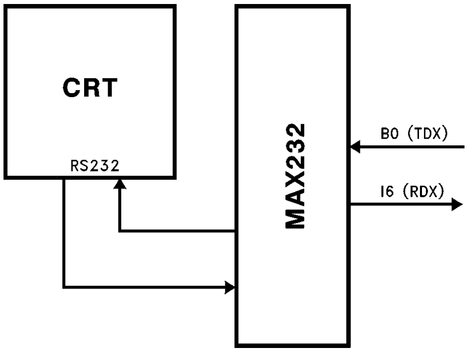
The LM629 Host Interrupt (HI) output could be hardwired to an input capture port (I2 in this case) or the NMI pin of the HPC for handshaking purposes.

The Port Select ($\overline{\text{PS}}$) input is hardwired to the pin B1 of the HPC for selecting command or data when communicating with the LM629.

The LM629 gets its RESET from the HPC which is low for a minimum of 8 CK2 periods to satisfy its reset requirement.

USER-SYSTEM INTERFACE

Figure 2 shows a typical system implemented for handshaking between the user and the motion controller block. In this specific application, this is developed using the HPC's on-chip UART communicating at 4800 baud using a standard 10 MHz HPC input clock (CKI). Any terminal (*e.g.,* a VT100) hooked to the HPC UART via the MAX232 should serve the purpose. The user can then input data required to control the motor trajectory and tuning the PID filter for precision motion control via this interface.

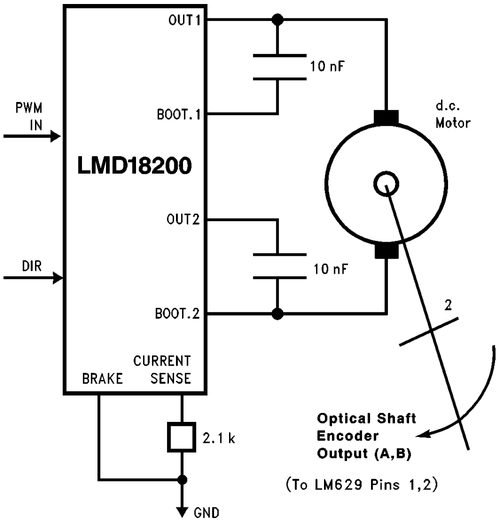


TL/DD/11770–2

FIGURE 2. User-System Interface

PWM MOTOR DRIVE INTERFACE

Figure 3 shows an LMD 18200—a 3A H-bridge driver is interfaced to the LM629 PWM outputs to provide power amplification for driving brush/commutator and brushless D.C. motors. The motor used in this application is a brush D.C. motor with a HP HEDS 5300 incremental optical shaft encoder with 2-channel (without the index pulse) and 500 cps providing TTL compatible digital output.



TL/DD/11770–3

FIGURE 3. DC Motor Output Driver Stage

MOTION CONTROL SOFTWARE

Appended at the end of this application note is a flowchart *(Figure 4)* for initiating a simple motor movement and a sample software (in "C") provided for the benefit of the reader. Note that the requirements of individual routines are very much application specific and left for the user to develop as required.

TUNING THE PID

When connecting up a drive system for the first time, there could be a possibility that the loop phasing is incorrect. As this may cause severe oscillation, it is recommended to use a low value for the proportional gain, say Kp = 1 (with Kd, Ki, and il all set to zero), which will provide a weak level of drive to the motor. If the system does oscillate with this Kp value then the motor connections should be reversed.

Having determined that the loop phasing is correct Kp can now be increased to about 20 to see that the control system basically works. This value of Kp should hold the motor shaft reasonably stiffly, returning the motor to the set position, which will be zero until trajectory values have been input and a position move performed. If oscillation and unacceptable ringing still occurs, Kp should be reduced until it stops. Low values of acceleration and velocity can now be input, of around 100, and a position move commanded to say 1000 counts. All values suggested here are decimal.

It is useful at this stage to try different values of acceleration and velocity to get a feel for the system limitation. These can be determined by reporting desired and actual velocity and acceleration to see that the error is not increasing without bound.

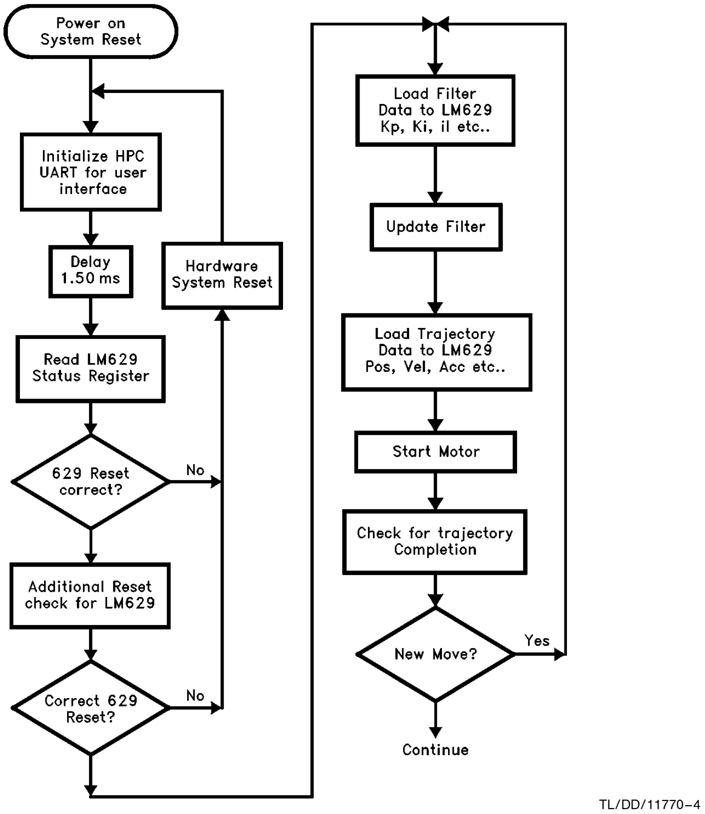

Power on System Reset
Initialize HPC UART for user interface
Delay 1.50 ms
Hardware System Reset
Read LM629 Status Register
629 Reset correct?
No
Additional Reset check for LM629
Correct 629 Reset?
No
Load Filter Data to LM629 Kp, Ki, il etc..
Update Filter
Load Trajectory Data to LM629 Pos, Vel, Acc etc..
Start Motor
Check for trajectory Completion
New Move?
Yes
Continue
TL/DD/11770–4

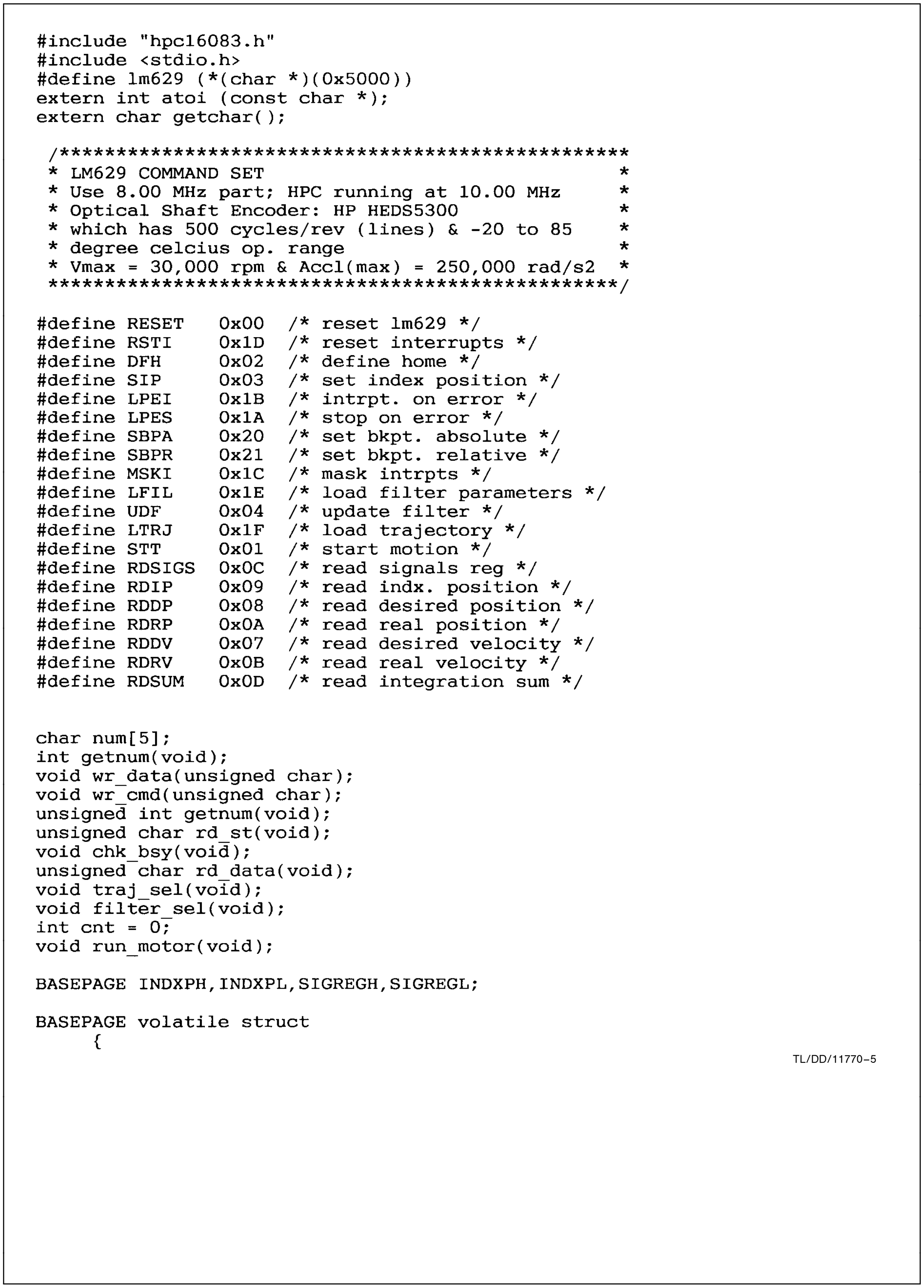
FIGURE 4. Flowchart of a Simple Motor Move

CONCLUSION

The combination chip-set HPC and LM629 (also the LM628) form the core of a powerful solution to position servo problems. Commanded by the HPC microcontroller, the most powerful single-chip microcontroller available, this unique combination is the key to a flexible and easy-to-implement coordinated multi-axis motion system.

REFERENCES

1. Special Purpose Linear Devices Data Book—NSC
2. HP HEDS-5000 series Optical Encoder Data Book
3. Linear Data Book—NSC
4. HPC User's Manual—NSC
5. Motion Control Handbook—NSC

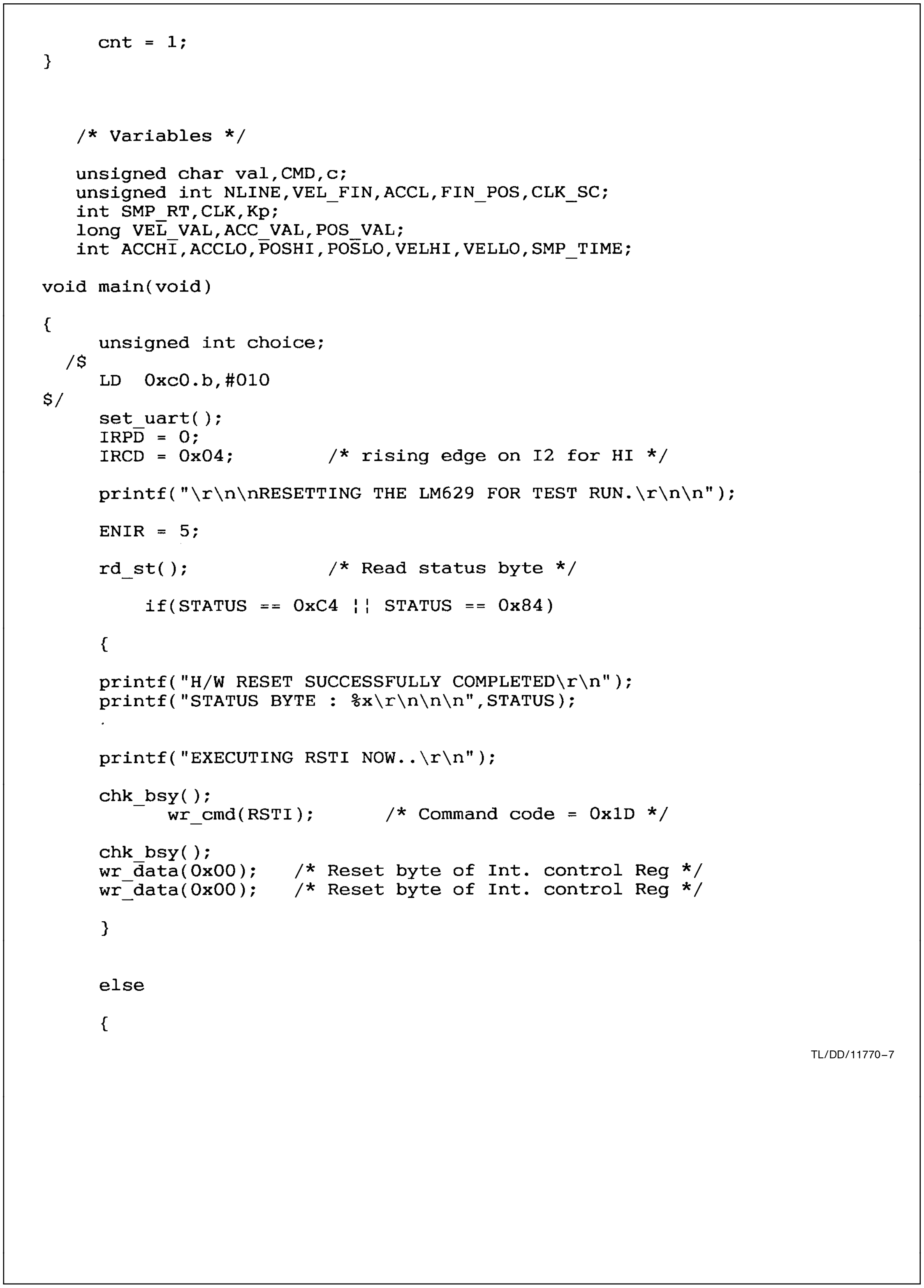


```
#include "hpc16083.h"
#include <stdio.h>
#define lm629 (*(char *)(0x5000))
extern int atoi (const char *);
extern char getchar();

 /***************************************************
 * LM629 COMMAND SET                                *
 * Use 8.00 MHz part; HPC running at 10.00 MHz      *
 * Optical Shaft Encoder: HP HEDS5300               *
 * which has 500 cycles/rev (lines) & -20 to 85     *
 * degree celcius op. range                         *
 * Vmax = 30,000 rpm & Accl(max) = 250,000 rad/s2  *
 ***************************************************/

#define RESET   0x00  /* reset lm629 */
#define RSTI    0x1D  /* reset interrupts */
#define DFH     0x02  /* define home */
#define SIP     0x03  /* set index position */
#define LPEI    0x1B  /* intrpt. on error */
#define LPES    0x1A  /* stop on error */
#define SBPA    0x20  /* set bkpt. absolute */
#define SBPR    0x21  /* set bkpt. relative */
#define MSKI    0x1C  /* mask intrpts */
#define LFIL    0x1E  /* load filter parameters */
#define UDF     0x04  /* update filter */
#define LTRJ    0x1F  /* load trajectory */
#define STT     0x01  /* start motion */
#define RDSIGS  0x0C  /* read signals reg */
#define RDIP    0x09  /* read indx. position */
#define RDDP    0x08  /* read desired position */
#define RDRP    0x0A  /* read real position */
#define RDDV    0x07  /* read desired velocity */
#define RDRV    0x0B  /* read real velocity */
#define RDSUM   0x0D  /* read integration sum */


char num[5];
int getnum(void);
void wr_data(unsigned char);
void wr_cmd(unsigned char);
unsigned int getnum(void);
unsigned char rd_st(void);
void chk_bsy(void);
unsigned char rd_data(void);
void traj_sel(void);
void filter_sel(void);
int cnt = 0;
void run_motor(void);

BASEPAGE INDXPH,INDXPL,SIGREGH,SIGREGL;

BASEPAGE volatile struct
     {
```

TL/DD/11770–5

```
          unsigned   bsy:         1; /* busy bit */
          unsigned   cmd_err:     1; /* command err (int) */
          unsigned   trj_cmp:     1; /* traj. over (int) */
          unsigned   indx_pls:    1; /* indx. pulse (int) */
          unsigned   wrp_occ:     1; /* wraparound (int) */
          unsigned   posn_err:    1; /* excess pos.err(int)*/
          unsigned   bkpt_rch:    1; /* bkpt. reached(int) */
          unsigned   motor_off:   1; /* motor off */

     } status;


#define STATUS (*(volatile unsigned char *)&status)



INTERRUPT2 HOST_INTR(void)
{

     if(cnt == 0)
          goto done;

     printf("\r\n\n");
     printf("*********** HOST INTERRUPT!! **************\r\n");

     rd_st();

        printf("STATUS : %x\r\n",STATUS);

     if(status.bsy)
    printf("ERROR .. RD/WR WHILE LM629 IS BUSY !! \r\n");

     if(status.cmd_err)
    printf("COMMAND ERROR !! \r\n");

     if(status.trj_cmp)
    printf("MOTOR TRAJECTORY COMPLETE!\r\n");

     if(status.indx_pls)
    printf("INDEX PULSE OCCURRED.\r\n");

     if(status.wrp_occ)
    printf("POSITION RANGE ERROR !!\r\n");

     if(status.posn_err)
    printf("EXCESSIVE POSITION ERROR !!\r\n");

     if(status.bkpt_rch)
    printf("MOTOR REACHED SET BRKPT. POSITION.\r\n");

     if(status.motor_off)
    printf("STALLED MOTOR CONDITION.\r\n");

done:
```

TL/DD/11770–6

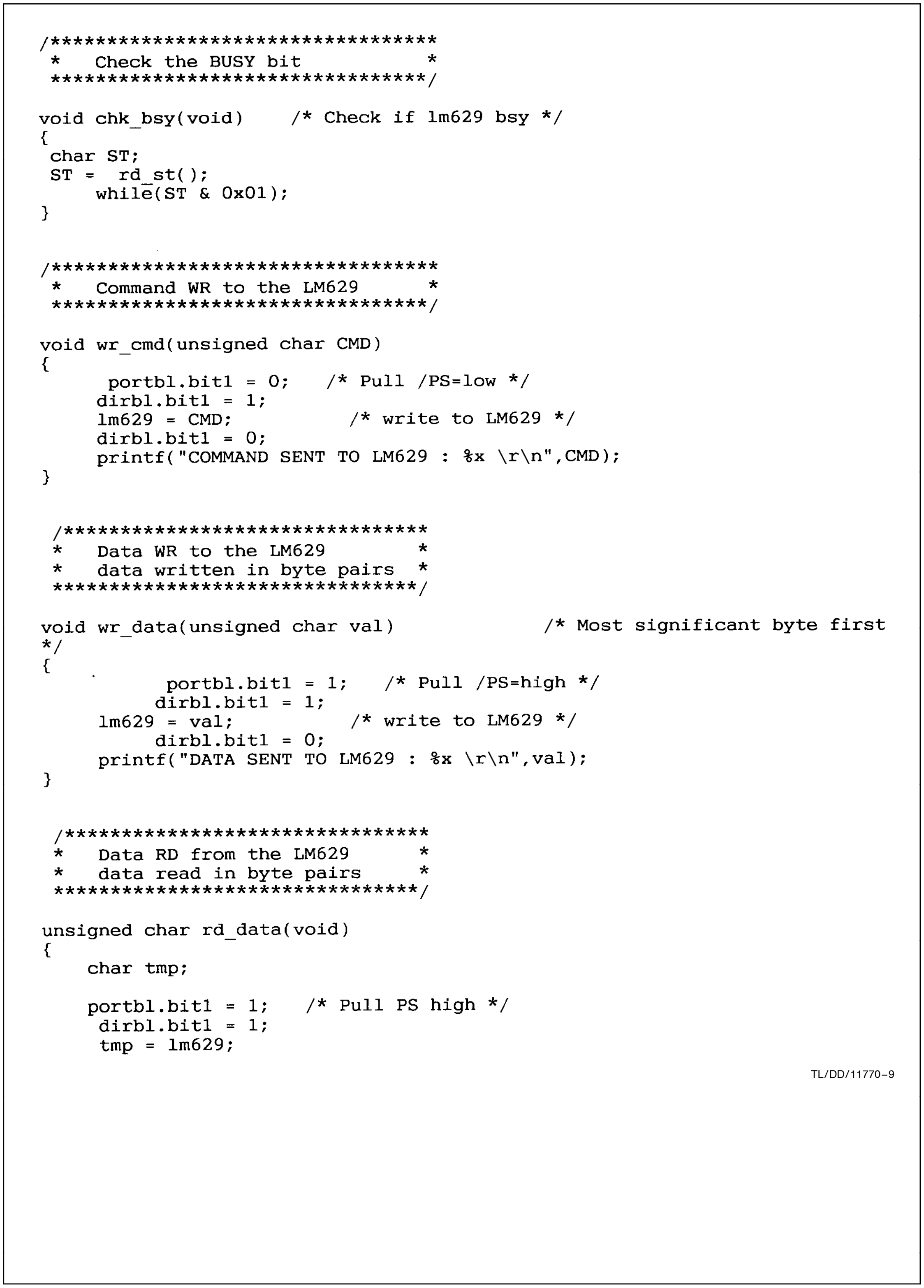


```
     cnt = 1;
}



   /* Variables */

   unsigned char val,CMD,c;
   unsigned int NLINE,VEL_FIN,ACCL,FIN_POS,CLK_SC;
   int SMP_RT,CLK,Kp;
   long VEL_VAL,ACC_VAL,POS_VAL;
   int ACCHI,ACCLO,POSHI,POSLO,VELHI,VELLO,SMP_TIME;

void main(void)

{
     unsigned int choice;
  /$
     LD  0xc0.b,#010
$/
     set_uart();
     IRPD = 0;
     IRCD = 0x04;        /* rising edge on I2 for HI */

     printf("\r\n\nRESETTING THE LM629 FOR TEST RUN.\r\n\n");

     ENIR = 5;

     rd_st();            /* Read status byte */

         if(STATUS == 0xC4 ¦¦ STATUS == 0x84)

     {

     printf("H/W RESET SUCCESSFULLY COMPLETED\r\n");
     printf("STATUS BYTE : %x\r\n\n\n",STATUS);


     printf("EXECUTING RSTI NOW..\r\n");

     chk_bsy();
           wr_cmd(RSTI);      /* Command code = 0x1D */

     chk_bsy();
     wr_data(0x00);   /* Reset byte of Int. control Reg */
     wr_data(0x00);   /* Reset byte of Int. control Reg */

     }


     else

     {
```

TL/DD/11770-7

```
    printf("*** H/W RESET FAILURE, TRY AGAIN ***\r\n");
    printf("STATUS BYTE : %x\r\n\n",STATUS);
    while(1);

    }

    rd_st();

         if(STATUS == 0x80 ¦¦ STATUS ==0xC0)

   {
    printf("LM629 RESET SUCCESSFULLY COMPLETED.\r\n");
    printf("STATUS BYTE (AFTER RSTI) : %x\r\n\n\n",STATUS);
menu:
    printf("\r\n\n\n\n");
    printf("      SELECT ONE FROM THE FOLLOWING NOW:\r\n");
    printf("      ----------------------------------\r\n");

    printf("      1 .............LOAD NEW FILTER DATA \r\n");
    printf("      2 .............LOAD NEW TRAJECTORY DATA \r\n");
    printf("      3 .............RUN THE MOTOR\r\n");



    printf("-->");

    choice = getnum();
         switch(choice)
    {
         case 1 :
         filter_sel();
         break;

         case 2 :
         traj_sel();
         break;

         case 3 :
         run_motor();
         break;

         default:
         printf("\r\nBAD SELECTION,TEST EXITED !!\r\n");
    }
    goto   menu;
   }

    printf("RSTI COMMAND FAILED, RESET LM629 AGAIN\r\n");
    while(1);

}
```

TL/DD/11770–8



```
/**********************************
 *   Check the BUSY bit           *
 **********************************/

void chk_bsy(void)    /* Check if lm629 bsy */
{
 char ST;
 ST =  rd_st();
     while(ST & 0x01);
}


/**********************************
 *   Command WR to the LM629      *
 **********************************/

void wr_cmd(unsigned char CMD)
{
      portbl.bit1 = 0;   /* Pull /PS=low */
     dirbl.bit1 = 1;
     lm629 = CMD;           /* write to LM629 */
     dirbl.bit1 = 0;
     printf("COMMAND SENT TO LM629 : %x \r\n",CMD);
}


 /********************************
 *   Data WR to the LM629        *
 *   data written in byte pairs  *
 ********************************/

void wr_data(unsigned char val)             /* Most significant byte first
*/
{
           portbl.bit1 = 1;   /* Pull /PS=high */
          dirbl.bit1 = 1;
     lm629 = val;           /* write to LM629 */
          dirbl.bit1 = 0;
     printf("DATA SENT TO LM629 : %x \r\n",val);
}


 /********************************
 *   Data RD from the LM629      *
 *   data read in byte pairs     *
 ********************************/

unsigned char rd_data(void)
{
    char tmp;

    portbl.bit1 = 1;   /* Pull PS high */
     dirbl.bit1 = 1;
     tmp = lm629;
```

TL/DD/11770-9

```
     dirbl.bit1 = 0;
     return(tmp);
}

/************************************
 * Reading Status Byte From LM629   *
 ***********************************/

unsigned char rd_st(void)
   {
     portbl.bit1 = 0;    /* Pull /PS low */
     dirbl.bit1 = 1;
     STATUS=lm629;       /* Dummy read to pull /RD and /CS low */
     dirbl.bit1 = 0;         /* Deselect LM629 */
     return(STATUS);
                    /* Save it in status memory */
   }

unsigned int getnum(void)
{
unsigned int i=0,NUM;
char c;
extern char num[];

          while(c=getchar())
     {
          num[i++] = c;
     if( c == '\r'¦¦ c == '\n')    {
     num[i] = '\0';
          break;              }
     }
     NUM = atoi(num);
          return(NUM);
}

void traj_sel(void)
{
     printf("\r\n\n");
     printf("ENTER ENCODER LINES :");
          NLINE = getnum();
          printf("\r\n");
     printf("ENTER FINAL VELOCITY (RPM) :");
          VEL_FIN = getnum();
          printf("\r\n");
     printf("ENTER FINAL POSITION (REVS) : ");
          FIN_POS = getnum();
          printf("\r\n");
     printf("ENTER ACCELERATION (REVS/S-2) : ");
          ACCL = getnum();
          printf("\r\n");


   POS_VAL = NLINE * 4.0 * FIN_POS;
     printf("POSITION VALUE TO LOAD : %lx\r\n",POS_VAL);
```

TL/DD/11770–10

```
   VEL_VAL = NLINE * 4.0 * SMP_TIME * 1.0E-6 * VEL_FIN * (1.0/60.0)*
65536.0;
     printf("VELOCITY VALUE TO LOAD : %lx\r\n",VEL_VAL);

   ACC_VAL = NLINE * 4.0 * SMP_TIME * SMP_TIME * 1.0E-12 * ACCL * 65536.0;
     printf("ACCL. VALUE TO LOAD : %lx\r\n",ACC_VAL);


   ACCLO = (unsigned int)(ACC_VAL & 0x0000ffff);
   ACCHI = (unsigned int)((ACC_VAL >> 16) & 0x0000ffff);


   VELHI = (unsigned int) ((VEL_VAL >> 16) & 0x0000ffff);
   VELLO = (unsigned int) (VEL_VAL & 0x0000ffff);


   POSHI = (unsigned int) ((POS_VAL >> 16)& 0x0000ffff);
   POSLO = (unsigned int) (POS_VAL & 0x0000ffff);



     printf("LOADING NEW TRAJECTORY DATA\r\n");

     chk_bsy();
                wr_cmd(LTRJ);

     chk_bsy();
             wr_data(0x00);
          wr_data(0x2A);
     chk_bsy();

           wr_data((ACCHI >> 8) & 0x00ff);
              wr_data(ACCHI & 0x00ff);
     chk_bsy();
           wr_data((ACCLO >> 8) & 0x00ff);
           wr_data(ACCLO & 0x00ff);
     chk_bsy();
           wr_data((VELHI >> 8) & 0x00ff);
           wr_data(VELHI & 0x00ff);
     chk_bsy();
           wr_data((VELLO >> 8) & 0x00ff);
           wr_data(VELLO & 0x00ff);
     chk_bsy();
           wr_data((POSHI >> 8) & 0x00ff);
           wr_data(POSHI & 0x00ff);
     chk_bsy();
           wr_data((POSLO >> 8) & 0x00ff);
           wr_data(POSLO & 0x00ff);

     printf("TRAJECTORY DATA LOAD COMPLETED.\r\n\n\n");

}
```

TL/DD/11770-11

```
void filter_sel(void)
{

     printf("\r\n\n");
     printf("SELECT CLOCK RATE (2-8 MHz) :");
          CLK = getnum();
     printf("\r\nCLOCK RATE IS : %d MHz\r\n",CLK);

     printf("SELECT CLOCK SCALAR (0-255 DEC) :");
          CLK_SC = getnum();
     printf("\r\nSCALAR TO LOAD: %x\r\n",CLK_SC);

          SMP_RT = 2048/CLK;
          SMP_TIME = (CLK_SC+1) * SMP_RT;
     printf("SAMPLE TIME : %d usecs\r\n\n",SMP_TIME);

     printf("SELECT Kp TO LOAD : ");
               Kp = getnum();
     printf("\r\n");



     printf("LOADING NEW FILTER DATA\r\n");

     chk_bsy();
               wr_cmd(LFIL);  /* Load filter parameter cmd = 0x1e */

     chk_bsy();
          wr_data(SMP_RT);   /* Bits 8-15 (MSB) set the derivative */
               wr_data(0x08);     /*   sampling rate. = 256 uS */

     chk_bsy();
          wr_data(0x00);
          wr_data(Kp);

     printf("FILTER DATA LOADED\r\n");
     printf("UPDATING FILTER DATA NOW (UDF)\r\n");

     chk_bsy();
          wr_cmd(UDF);   /* Update filter */

     printf("UPDATED FILTER PARAMETERS (UDF)\r\n\n\n");

}

void run_motor(void)
{
     printf("EXECUTING STT TO RUN THE MOTOR ..!!\r\n");
     printf("WATCH THE MOTOR NOW .. !!\r\n\n\n");

     chk_bsy();
          wr_cmd(STT);             /* start motion 0x01 */
}
```

TL/DD/11770–12

AN-868 Interfacing the HPC and LM629 for Motion Control

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor
Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor
GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor
Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor
Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores
Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor
(Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.