Freescale Semiconductor
Application Note

Document Number: AN2470
Rev. 1, 01/2007

MC68HC908EY16 Controlled Robot Using the LIN Bus

by Peter Topping
East Kilbride

1 Introduction

The LIN (Local Interconnect Network, reference [1]) protocol was specified for automotive applications, but there is no reason why it cannot be used for other applications. Its single data line facilitates a three wire serial control and power bus that can reduce the wiring to a minimum in many applications. This application note describes the use of the LIN bus to control a small robot powered by the type of servo used in radio-controlled models. The robot has five degrees of freedom and could easily be controlled by a single microcontroller but, as a demonstration of LIN, each servo is a separate slave node and thus five MCUs are employed.

Each of these nodes is a simple LIN slave so the complete system requires a master node and a method of sending position information for the various nodes. A master based on an HCS12 was used. A LIN slave cannot initiate communications. This is the responsibility of the master which on a regular basis, every 100ms in this case, issues header frames that contain the required

Contents

1 Introduction 1
2 The Robot 2
3 Messaging 4
4 Hardware 5
5 Software 6
6 References 9
7 Software listing 10
Appendix AInclude File
(register definitions for the MC68HC908EY16) 15
Appendix BVector.c 16
Appendix CSlave.cfg (LIN configuration file) 18
Appendix DSlave.id (LIN message ID file) 18

freescale semiconductor

message IDs. These messages either include data or invite a slave node to respond with its data. Both of these methods of controlling the robot are incorporated.

The first method involves direct control from the master by sending co-ordinates via a LIN message with an ID of $30 (master to slave communication). The second method utilizes the car door keypad described in application note AN2205 (Car door keypad Using LIN, reference [2]). This LIN slave application has four window keys with fast and slow positions for up and down which are ideal for controlling four of the five servos. It also includes a mirror position adjustment control used to control the fifth servo which operates the robot's grabber. Keypad control demonstrates slave to slave communication (using a message with an ID of $20). The keypad also incorporates a child-lock switch is used in this application to enable and disable direct control from the master.

Figure 1 shows the resultant LIN network. It includes a VCT (Volcano Communications Technologies) LIN spector which, in this arrangement, is not essential for functionality, but can monitor the LIN bus activity. It could alternatively include the functionality of the master thus obviating the need for the HCS12 node. The five slave nodes distinguish themselves by giving different hardware addresses using three I/O pins. This method allows the MCU software to be identical in all five slave nodes. An alternative approach would keep the hardware identical and modify the code so that each node has an address in FLASH or responds only to a message ID unique to that particular node.

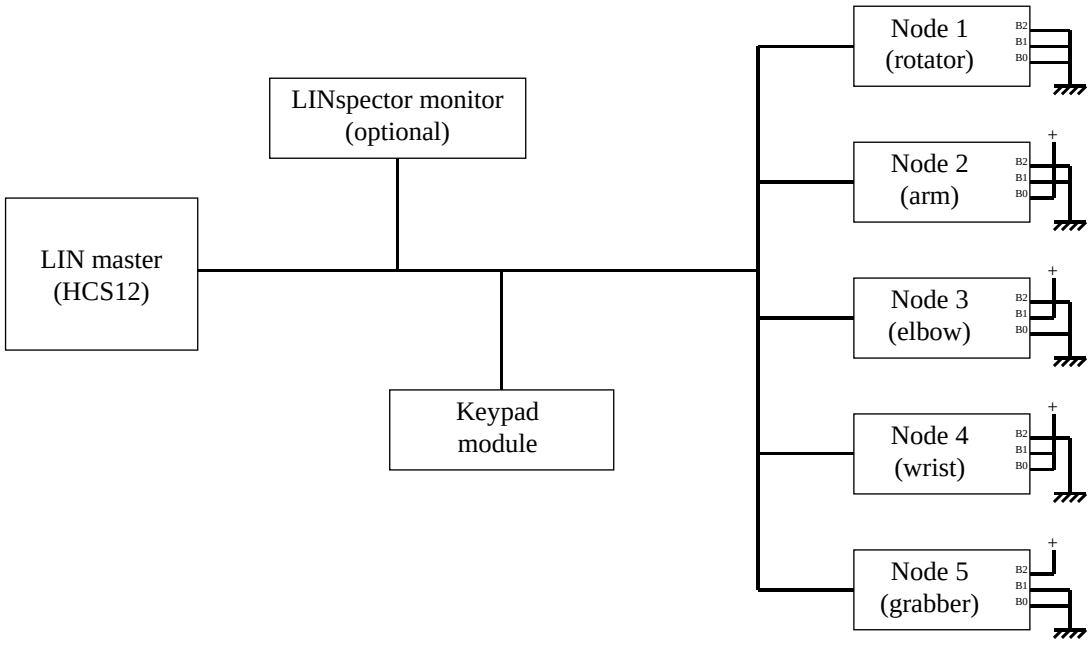

Figure 1. The LIN Network Used for the Robot

2 The Robot

The Robot is a standard kit from Lynxmotion, the only modification being that the supplied controller board is not used. Control is achieved by connecting the control wire for each servo to its LIN node. The robot has five degrees of freedom as shown in Figure 2. The movements are rotation, three arm joints, and the opening and closing of the grabber.

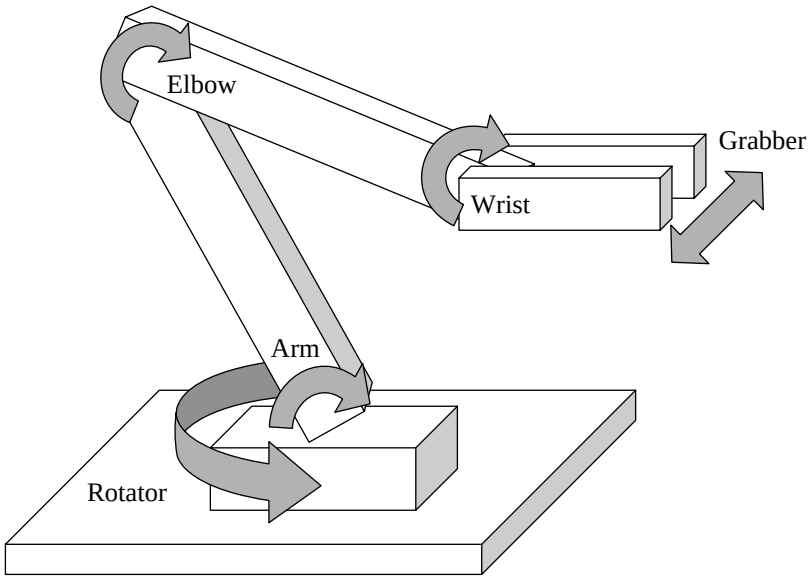

Figure 2. The Robot's Servo Controlled Axes

To control the servos, a pulse is sent with proportional width to the desired position. They have internal feedback to help them achieve a guaranteed position over a range of 90×. With the central position corresponding to a pulse width of 1.5ms, ±45× is achieved by varying the width between 1.0ms and 2.0ms. In practice, the servos can operate over nearly 180× and a larger range is used for the rotation servo. 90× proved just about right for the three arm joints although the grabber requires less. A need arises to use a different gain when converting the 1-byte position information into the pulse width for each servo. Using the same gain (7) for the three arm servos gives the advantage that operating two of the servos in opposite directions at the same rate facilitates movement without any change of the angle of the grabber relative to the surface the robot stands on. This gives it a more co-ordinated appearance and makes it generally easier to control. Because the elbow is driven by a servo that is on the far side of its node, the polarity of its control is reversed so that values above $80 correspond to negative rather than positive angles. Table 1 shows the parameters chosen for each servo along with their initial values.

Table 1. Servo Ranges and Initial Positions

	Function	Gain	Pulse range (0x80 = 1.5 ms)	Angle range	Initial value (0x80 = 0°)	Initial position
Servo 1	Rotation	10	1.5 ± 0.71ms	± 64°	0x80	0°
Servo 2	Arm	7	1.5 ± 0.50ms	± 45°	0xD0	28°
Servo 3	Elbow	7	1.5 ± 0.50ms	± 45°	0xF0	–39°
Servo 4	Wrist	7	1.5 ± 0.50ms	± 45°	0xA0	11°
Servo 5	Grabber	3	1.5 ± 0.21ms	± 19°	0x80	0°

3 Messaging

The format of the LIN message data sent by the master is shown in Table 2. As the standard LIN message lengths incorporate 2, 4, or 8 bytes of data, an 8-byte message was chosen although only 5 are actually used. The data simply incorporates 1 byte of positional data per servo. The software is written so that the 50% value (0x80) corresponds to the central (0×) position.

Table 2. Format of the ID 30 LIN Message

ID $30	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
bit 0	Servo 1 (rotator) absolute position	Servo 2 (arm) absolute position	Servo 3 (elbow) absolute position	Servo 4 (wrist) absolute position	Servo 5 (grabber) absolute position	not used
bit 1						
bit 2						
bit 3						
bit 4						
bit 5						
bit 6						
bit 7						

The keypad used is described in detail in application note AN2205 [2]. It facilitates the control of four windows and two mirrors. Table 3 shows the re-allocation of the original bits for use with the robot, but the keypad was not modified, its hardware and software exactly as described in AN2205.

The manual up and down keys are used to move the various servos slowly while faster movement is facilitated by the express keys. The mirror joystick controls the fifth (grabber) servo. The duplicate allocation of the grabber bits is a result of the keypad having eight mirror control bits (up/down/left/right for two mirrors). Bit 7 in the otherwise unused control byte is controlled by the child lock switch on the keypad to tell the master whether or not to output its automatic sequence by sending messages with ID30 complete with their response data as shown in Table 2.

Table 3. Format of the ID 20 LIN Message

ID $20	Byte 0 (servos 1 & 2)	Byte 1 (servos 3 & 4)	Byte 2 (servo 5)	Byte 3 (control)
bit 0	Rotate right, fast	Elbow up, fast	Close grabber	not used
bit 1	Rotate left, fast	Elbow down, fast	Open grabber	not used
bit 2	Rotate right, slow	Elbow up, slow	Close grabber	not used
bit 3	Rotate left, slow	Elbow down, slow	Open grabber	not used

Table 3. Format of the ID 20 LIN Message (continued)

ID $20	Byte 0 (servos 1 & 2)	Byte 1 (servos 3 & 4)	Byte 2 (servo 5)	Byte 3 (control)
bit 4	Arm up, fast	Wrist up, fast	Close grabber	not used
bit 5	Arm down, fast	Wrist down, fast	Open grabber	not used
bit 6	Arm up, slow	Wrist up, slow	Close grabber	not used
bit 7	Arm down, slow	Wrist down, slow	Open grabber	Mode select

4 Hardware

The LIN evaluation board described in AN2343 (HC908EY16 LIN Monitor, reference [4]) and AN2432 (LIN sample application for the MC68HC908EY16 Evaluation Board, reference [5]) controls the robot. Five boards, identical except for a hardwired address, are used. Each board constitutes a LIN node and controls a single robot servo. The application is a demonstration of a LIN network and not intended as a practical method of controlling a robot. An optimised solution would control all five servos using a single MCU. The MCU used on the LIN evaluation board is the MC68HC908EY16 [3].

The full layout of the LIN evaluation board is shown in application notes AN2343 and AN2432 but the complete circuit diagram of the LIN node used in the robot application is shown here in Figure 3. Apart from the MCU itself, two chips are required to implement a simple LIN node. These are the LIN interface, in this case the Freescale MC33399 (or MC33661) and a 5 volt regulator. A single chip, the MC33689 (LIN SBC), can replace these chips. The voltage regulator used is an MC7805. The MC33399 (or MC33661) includes the 30kohm LIN pull-up so this does not need to be included on the PCB.

The only other components required are an 8MHz crystal with its associated passive components, two LEDs with their current limiting resistors, and a few pull-up resistors and decoupling capacitors (not shown). The servos are controlled by pulse width modulation using a single timer pin. To control the robot, this pin and its ground reference are the only connections to the board apart from the 3-wire LIN bus itself.

The LIN evaluation board includes a socket for a 9.8304MHz oscillator module as an alternative clock source and an RS232 interface. These components and the pull-ups and pull-downs required to enter monitor mode are, for clarity, not shown in Figure 3 but they allow the board to be used to develop code without the requirement for a sophisticated development system like the Metrowerks MMDS/MMEVS. The board also incorporates a 16-pin header to interface with P&E Cyclone or Multilink hardware (http://www.pemicro.com). This allows even easier debugging as the power and reset control required for FLASH programming can be handled by the PC running P&E WINIde or Metrowerks Codewarrior software (http://www.metrowerks.com).

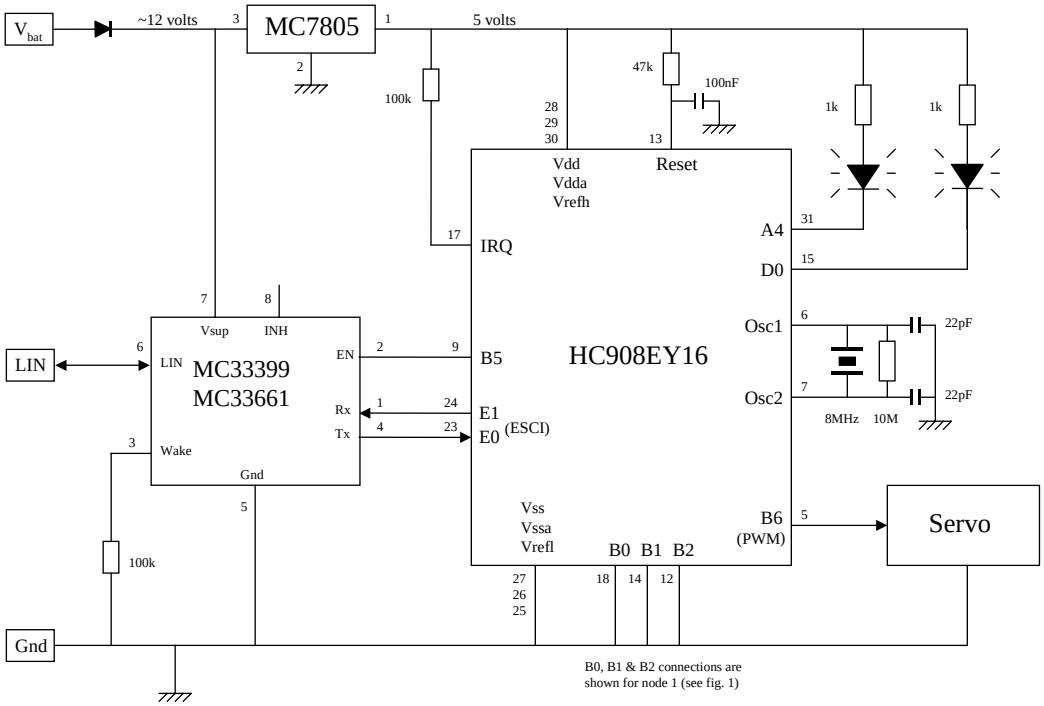

Figure 3. LIN Slave Node Circuit Diagram

5 Software

The software was written and debugged on the prototype PCB using the Mertowerks' Code warrior development environment. Debug was carried out using the serial monitor mode interface included on the LIN evaluation board. As CodeWarrior is available free for code sizes up to 4K, this type of application can be developed without requiring any investment in development software or hardware. The robot slave node uses 2.2K of code.

The free version of CodeWarrior actually comes with a 1K limit but this can be increased to 4K by requesting a free license from Metrowerks. The extension is valid for the Motosil (MMDS/MMEVS) and the P&E target interface. It is not valid for the MON08 interface. For this reason, the P&E target interface should be selected for the simple monitor mode development environment described above.

One disadvantage of this simple development environment is that it requires a 9.8304MHz clock. The LIN evaluation board facilitates this option but as the application was targeted to use an 8MHz crystal, the software had to be modified slightly to function at 8MHz once debugging was complete. This required a change in the SCI set-up to obtain 9600 baud for the different clocks and also different offsets and gains for the pulse width calculations. The values for 9.8304MHz are shown as comments in the relevant places in the code. This inconvenience can be avoided by using a P&E Cyclone or Multilink interface as they have an auto-baud capability that allows the use of an oscillator module that employs the target frequency. This can sometimes be done using the simple monitor mode interface at a non-standard baud rate but reliable operation is not guaranteed as the versatility PC COM ports varies.

The slave node module uses the Metrowerks LIN driver software to handle the LIN protocol without the application code. It only has to use "LIN_GetMsg()" calls (after an initial "LIN_Init()") to obtain the data

required to determine the position of the servos. The use of the LIN drivers leaves the programmer free to concentrate on the application without having to get too involved with the communications protocol.

The servos respond very quickly to a pulse width change and initially moved the robot too fast. That meant the software would have to control the rate to prevent movements being too jerky and also provide the two speeds made possible by the design of the keypad module.

The main software flow chart is shown in Figure 4. After the CONFIG and Port registers have been initialised, the MCU clock is switched from the default (ICG) to use the external crystal. As this setting is also suitable when using an external clock source, no change was required to suit the development method used. The time base module (TBM) is then set up to provide the repetition rate of 244Hz that is used to control the timing of the main software loop.

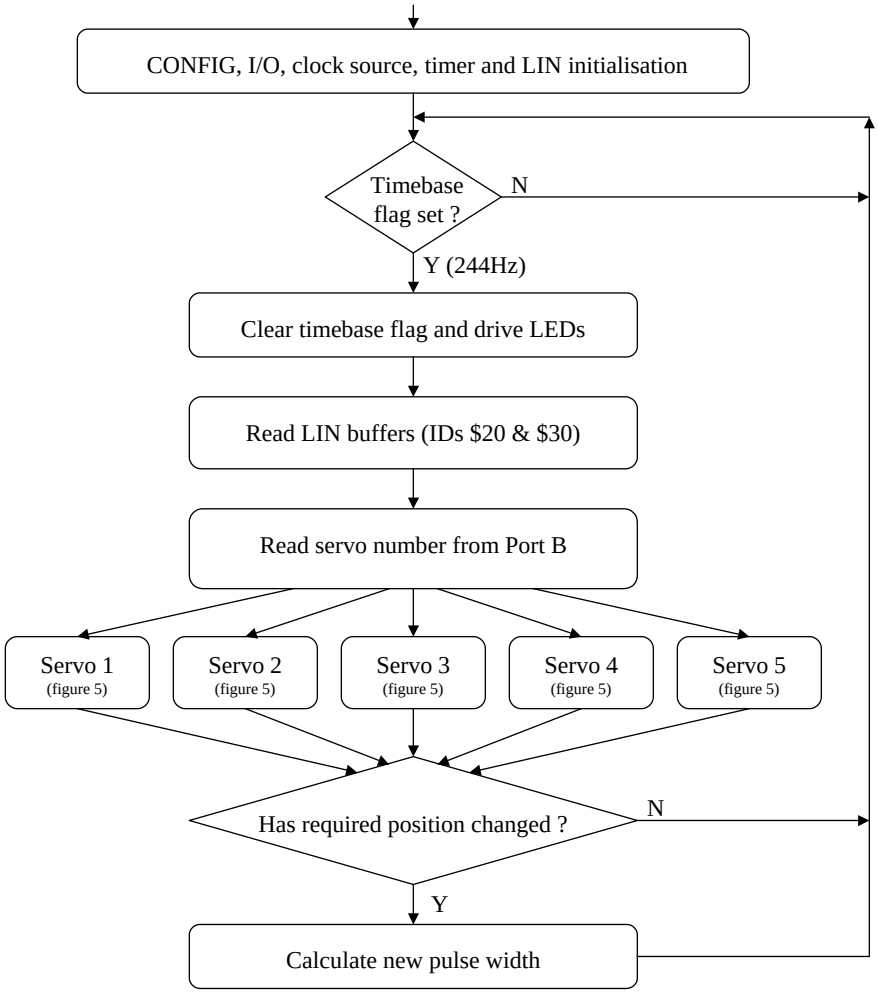

Figure 4. . Flow Chart of Main Software Loop

The timer's modulo divide registers are initialised to provide the nominal 60Hz pulse repetition rate required by the servos on the PWM output. This frequency is not critical so no change was required while using the slightly higher clock rate during debug. In order to prevent the transient generation of any

inappropriate pulse widths, the buffered mode of the timer's PWM output is selected. Interrupts are then enabled prior to the LIN drivers being initialised with the function call "LIN_Init()".

The main loop is then entered and executed under TBM control at 244Hz with an 8MHz clock. The LED connected to port pin D0 is flashed with a 50% duty cycle at this tick rate divided by 256. This provides an indication that the code is running correctly. A second LED connected to port pin A4 is used to indicate that the position of the servo is changing. The variable "position" is used together with the flags "led_flag" and "move_flag" to flash this LED at a rate proportional to the speed of movement of the servo and to ensure that it is not left illuminated when the movement ceases. The LIN buffers for ID20 and ID30 messages are then read and a case statement with code specific to each servo is entered. The switch statement "switch (PTB & 0x07)" reads the three port B input lines that are used in each node to determine its address.

The flowchart for the code within the case statement, which is specific to each particular node, is shown in Figure 5. The relevant byte from the ID30 message is transferred to the variable "abs_pos" from the LIN information in the array "Servo_data[x]". This is followed by the interpretation of the ID20 message keyboard data in the array "Kpm_data". This is more complicated as the four keypad bits relevant to the particular servo have to be treated separately. There are two increment bits corresponding to fast and slow incrementing. If a bit is set, the rate of incrementing is determined by using the value in the tick counter. This results in fast incrementing of the variable "position" at 122Hz (244/2) and slow incrementing at 15.25Hz (244/16). The decrementing bits are treated similarly. Note the interchange of "position + +" and "position - -" in the code for the third (elbow) server which is required to enable it to behave intuitively as described above. If no move bits are set for this servo, then the move LED is switched off and its toggling disabled.

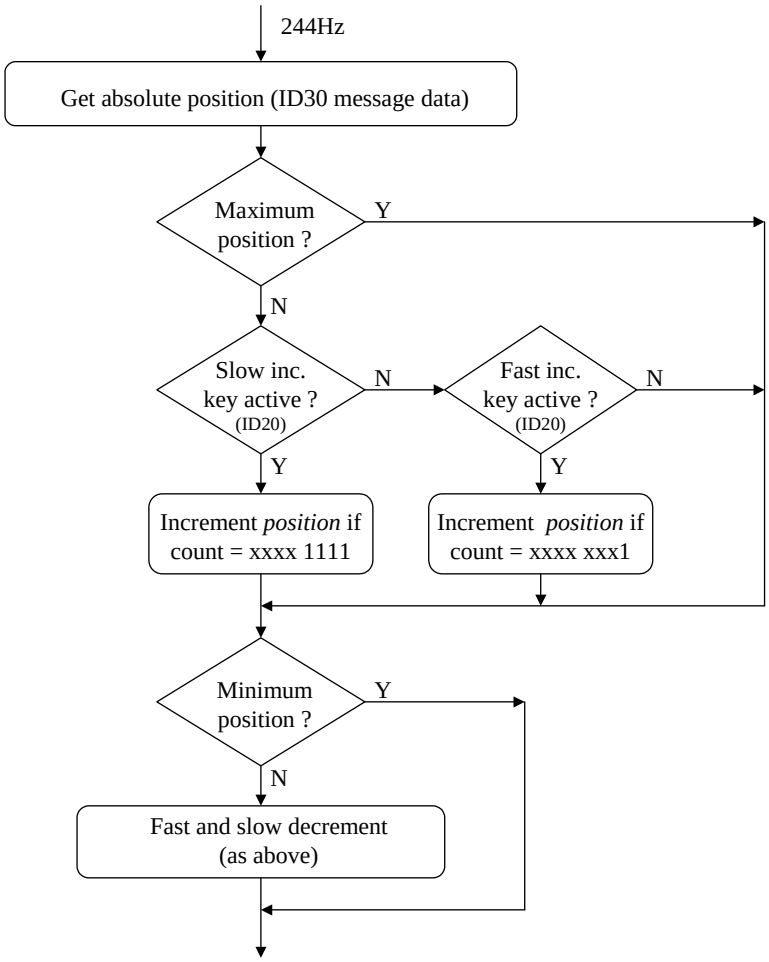

Figure 5. . Flow Chart of Servo Specific Code

The function “Width(x)” is then executed. If “move_flag” is set, the variable “position” is modified in the direction indicated by the absolute position derived from message ID30. If “move_flag” is clear, the absolute position information is ignored. This use of “move_flag” allows an ID30 message to be ignored once its position has been achieved. This allows the node to respond to keypad information even if an ID30 message is being continually sent with constant data. This might be the case if a LINspector was being used as a master. The value of “x” is the gain for that particular servo and is used in “Width(x)” along with the fixed offset, to calculate the PWM pulse width. This value is transferred to the timer registers in the interrupt routine “TimerB()”.

At the end of the case statement, the code in the main loop is again common to all servos. The variable “old_pos” is used to free up the lockout of ID30 messages if its absolute position data has changed. This is accomplished by setting “move_flag”. If, however, this flag is set and the desired position has been achieved, it is cleared and the move LED switched off.

6 References

1. LIN Protocol Specification, Version 1.2, 17 November 2000.

2. AN2205/D, Car Door Keypad Using LIN.
3. MC68HC908EY16 Technical Data.
4. AN2343, HC908EY16 LIN Monitor.
5. AN2432, LIN sample application for the MC68HC908EY16 Evaluation Board.

7 Software listing

```
/*****************************************************************************
*                       Copyright (c) Motorola 2002
*
*              908EY16 LIN servo controller for Robot project
*              ==============================================
*
*    Originator:    P. Topping
*    Date:          22nd August 2002
*    Revision:      1.0
*    Function:      Demonstration of LIN functionality using five EY16 LIN
*                   evaluation boards to control a Lynxmotion 5-axis robot.
*                   This robot employs pulsewidth controlled Hitec HS-422
*                   servos specified at +/- 45deg for 1ms +/- 0.5ms.
*
*****************************************************************************/

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

Motorola reserves the right to make changes without further notice to any
product herein to improve reliability, function or design. Motorola does not
assume any liability arising out of the application or use of any product,
circuit, or software described herein; neither does it convey any license
under its patent rights nor the rights of others. Motorola products are not
designed, intended, or authorized for use as components in systems intended
for surgical implant into the body, or other applications intended to support
life, or for any other application in which the failure of the Motorola product
could create a situation where personal injury or death may occur. Should
Buyer purchase or use Motorola products for any such unintended or
unauthorized application, Buyer shall indemnify and hold Motorola and its
officers, employees, subsidiaries, affiliates, and distributors harmless
against all claims costs, damages, and expenses, and reasonable attorney fees
arising out of, directly or indirectly, any claim of personal injury or death
associated with such unintended or unauthorized use, even if such claim
alleges that Motorola was negligent regarding the design or manufacture of the
part. Motorola and the Motorola logo are registered trademarks of Motorola Ltd.

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

/*****************************************************************************
*
*    Header file includes, function prototypes and defines
*
*****************************************************************************/

#include "HC08EY16.h"
#include <linapi.h>

void Width (unsigned char);

//#define offset 3686                           /* offset for 9.8304 MHz   */
#define offset 3000                             /* offset for 8.0000 MHz   */

/*****************************************************************************
*
*    Globals
*
*****************************************************************************/

unsigned char Kpm_data[4];
unsigned char Servo_data[8] = {0x80, 0xD0, 0xF0, 0xA0, 0x80};
unsigned char count = 0;
unsigned char msb;
unsigned char lsb;
unsigned char position = 0x80;
unsigned char old_pos = 0x00;
unsigned char abs_pos;
unsigned char toggle;
unsigned char move_flag;
unsigned char led_flag;
int result;
```

```
/*****************************************************************************
*
*    Function name: Main
*    Originator:    P. Topping
*    Date:          22nd August 2002
*
*****************************************************************************/

void main (void)
{

CONFIG1 = 0x01;                                 /* disable COP             */
CONFIG2 = 0x29;                                 /* ext.clk, fast timebase  */

DDRA    = 0x10;                                 /* enable A4 out for LED   */
DDRB    = 0x20;                                 /* enable B5 out for 33399 */
DDRC    = 0x80;                                 /* enable MCLK (C2)        */
DDRD    = 0x01;                                 /* enable D0 out for LED   */

PTB     = 0x20;                                 /* MC33399 enable high     */

while (ICGCR != 0x13)                           /* switch to ext. clock    */
{ ICGCR = 0x12; }                               /* and wait for switch     */

TBCR = 0x00;                                    /* TBM divide by 32768     */
TBCR = 0x02;                                    /* and switch it on        */

TBSC   = 0x70;                                  /* stop, clear, /64, INTs  */
TBMODH = 0x82;                                  /* 60Hz at 8.0 MHz         */
TBMODL = 0xFF;
TBSC0  = 0x2A;                                  /* Timer B buffered PWM    */
TBSC  &= ~(0x20);                               /* start timer             */

asm CLI;                                        /* enable interrupts       */

LIN_Init();                                     /* initialise LIN drivers  */

while (1)
{
    if (TBCR & 0x80)                            /* is TBM flag set?        */
    {
        TBCR |= 0x08;                           /* yes, clear it           */

        count ++;                               /* increment tick counter  */

        if (count & 0x80)                       /* check MS bit of count   */
        {
            PTD |= 0x01;                        /* tick LED off            */
        }
        else
        {
            PTD &= ~(0x01);                     /* tick LED on             */
        }


        if (position & 0x08)                    /* position dependant LED  */
        {
            PTA |= 0x10;                        /* moving LED off          */
        }
        else if (led_flag)
        {
            PTA &= ~(0x10);                     /* moving LED on           */
        }


        LIN_GetMsg (0x30, Servo_data);          /* get absolute position   */

        LIN_GetMsg (0x20, Kpm_data);            /* get Keypad data         */

        switch (PTB & 0x07)                     /* select servo            */
        {
```

```
/*****************************************************************************
*                                                                            *
*    Servo 1 - Address 0x00 on bits 0-2 of port B    -   Rotator             *
*            - Uses front passenger window keys                              *
*                                                                            *
*****************************************************************************/

            case 0:                             /* Servo 1, Rotator        */

            abs_pos = Servo_data[0];

            if ((position < 255) &&             /* increment allowed ?     */
               (((Kpm_data[0] & 0x04) && !(count & 0x0F)) ||
                ((Kpm_data[0] & 0x01) && !(count & 0x01))))
            {
                position ++;                    /* increment               */
                led_flag = 1;                   /* enable move LED         */
            }

            if ((position > 0) &&               /* decrement allowed ?     */
               (((Kpm_data[0] & 0x08) && !(count & 0x0F)) ||
                ((Kpm_data[0] & 0x02) && !(count & 0x01))))
            {
                position --;                    /* decrement               */
                led_flag = 1;                   /* enable move LED         */
            }

            if (((Kpm_data[0] & 0x0F) == 0) && (move_flag == 0))
            {
                PTA |= 0x10;                    /* switch move LED off     */
                led_flag = 0;                   /* and keep it off         */
            }

            Width(10);                          /* calculate 8.0MHz pulse  */
            break;                              /* width(13) for 9.8304MHz */



/*****************************************************************************
*                                                                            *
*    Servo 2 - Address 0x01 on bits 0-2 of port B    -   Arm                 *
*            - Uses driver window keys                                       *
*                                                                            *
*****************************************************************************/

            case 1:                             /* Servo 2, Arm            */

            abs_pos = Servo_data[1];

            if ((position < 255) &&             /* increment allowed ?     */
               (((Kpm_data[0] & 0x40) && !(count & 0x0F)) ||
                ((Kpm_data[0] & 0x10) && !(count & 0x01))))
            {
                position ++;                    /* increment               */
                led_flag = 1;                   /* enable move LED         */
            }

            if ((position > 0) &&               /* decrement allowed ?     */
               (((Kpm_data[0] & 0x80) && !(count & 0x0F)) ||
                ((Kpm_data[0] & 0x20) && !(count & 0x01))))
            {
                position --;                    /* decrement               */
                led_flag = 1;                   /* enable move LED         */
            }

            if (((Kpm_data[0] & 0xF0) == 0) && (move_flag == 0))
            {
                PTA |= 0x10;                    /* switch move LED off     */
                led_flag = 0;                   /* and keep it off         */

            }

            Width(7);                           /* calculate 8.0MHz pulse  */
            break;                              /* width(9) for 9.8304MHz  */
```

```
/*****************************************************************************
*                                                                            *
*    Servo 3 - Address 0x02 on bits 0-2 of port B    -   Elbow               *
*            - Uses nearside rear window keys                                *
*                                                                            *
*****************************************************************************/

            case 2:                             /* Servo 3, Elbow          */

            abs_pos = Servo_data[2];

            if ((position > 0) &&               /* decrement allowed ?     */
               (((Kpm_data[1] & 0x04) && !(count & 0x0F)) ||
                ((Kpm_data[1] & 0x01) && !(count & 0x01))))
            {
                position --;                    /* decrement               */
                led_flag = 1;                   /* enable move LED         */
            }

            if ((position < 255) &&             /* increment allowed ?     */
               (((Kpm_data[1] & 0x08) && !(count & 0x0F)) ||
                ((Kpm_data[1] & 0x02) && !(count & 0x01))))
            {
                position ++;                    /* increment               */
                led_flag = 1;                   /* enable move LED         */
            }

            if (((Kpm_data[1] & 0x0F) == 0) && (move_flag == 0))
            {
                PTA |= 0x10;                    /* switch move LED off     */
                led_flag = 0;                   /* and keep it off         */
            }

            Width(7);                           /* calculate 8.0MHz pulse  */
            break;                              /* width(9) for 9.8304MHz  */



/*****************************************************************************
*                                                                            *
*    Servo 4 - Address 0x03 on bits 0-2 of port B   -   Wrist                *
*            - Uses offside rear window keys                                 *
*                                                                            *
*****************************************************************************/

            case 3:                             /* Servo 4, Wrist          */

            abs_pos = Servo_data[3];

            if ((position < 255) &&             /* increment allowed ?     */
               (((Kpm_data[1] & 0x40) && !(count & 0x0F)) ||
                ((Kpm_data[1] & 0x10) && !(count & 0x01))))
            {
                position ++;                    /* increment               */
                led_flag = 1;                   /* enable move LED         */
            }

            if ((position > 0) &&               /* decrement allowed ?     */
               (((Kpm_data[1] & 0x80) && !(count & 0x0F)) ||
                ((Kpm_data[1] & 0x20) && !(count & 0x01))))
            {
                position --;                    /* decrement               */
                led_flag = 1;                   /* enable move LED         */
            }

            if (((Kpm_data[1] & 0xF0) == 0) && (move_flag == 0))
            {
                PTA |= 0x10;                    /* switch move LED off     */
                led_flag = 0;                   /* and keep it off         */
            }

            Width(7);                           /* calculate 8.0MHz pulse  */
            break;                              /* width(9) for 9.8304MHz  */
```

```
/*****************************************************************************
*                                                                            *
*    Servo 5 - Address 0x04 on bits 0-2 of port B    -   Grabber             *
*            - Uses mirror joystick                                          *
*                                                                            *
*****************************************************************************/

            case 4:                                 /* Servo 5, Grabber        */

            abs_pos = Servo_data[4];

            if ((position < 255) &&                 /* increment allowed ?     */
                (Kpm_data[2] & 0x55))
            {
                position ++;                        /* increment               */
                led_flag = 1;                       /* enable move LED         */
            }

            if ((position > 0) &&                   /* decrement allowed ?     */
                (Kpm_data[2] & 0xAA))
            {
                position --;                        /* decrement               */
                led_flag = 1;                       /* enable move LED         */
            }

            if ((Kpm_data[2] == 0) && (move_flag == 0))
            {
                PTA |= 0x10;                        /* switch move LED off     */
                led_flag = 0;                       /* and keep it off         */
            }

            Width(3);                               /* calculate 8.0MHz pulse  */
            break;                                  /* width(4) for 9.8403MHz  */

            default:                                /* will only happen with   */
            break;                                  /* an address of 5, 6 or 7 */
        }



/*****************************************************************************
*                                                                            *
*                           All servos                                       *
*                                                                            *
*****************************************************************************/


        if (abs_pos != old_pos)                     /* new absolute position ? */
        {
            move_flag = 1;                          /* yes, enable seek        */
            old_pos = abs_pos;                      /* disable restart         */
            led_flag = 1;                           /* and enable flashing LED */
        }


        if ((position == abs_pos) && (move_flag != 0))     /* got there ?    */
        {
            move_flag = 0;                          /* yes, clear flag         */
            PTA |= 0x10;                            /* switch move LED off     */
            led_flag = 0;                           /* and keep it off         */
        }

    }
}
}
```

```
/*****************************************************************************
*
*    Function name: Width - New absolute position and pulse width calculation
*    Originator:    P. Topping
*    Date:          6th September
*
*****************************************************************************/

void Width (unsigned char gain)
{
    if (move_flag)                              /* && !(count & 0x0X))     */
    {                                           /* to control speed        */
        if (abs_pos > position)
        {
            position ++;                        /* increment               */
        }
        else if (abs_pos < position)
        {
            position --;                        /* decrement               */
        }
    }

    result = gain*(position - 0x80) + offset;
    msb = result/256;
    lsb = result%256;
}


/*****************************************************************************
*
*    Function name: Timer B overflow interrupt routine
*    Originator:    P. Topping
*    Date:          29th August 2002
*
*****************************************************************************/

#pragma TRAP_PROC

void TimerB (void)
{
    TBSC &= ~(0x80);                            /* clear flag             */

    if (toggle & 0x01)                          /* use alternate channels */
    {
        TBCH0H = msb;
        TBCH0L = lsb;
    }
    else
    {
        TBCH1H = msb;
        TBCH1L = lsb;
    }
}


/*****************************************************************************
* Function:        LIN_Command -  User call-back.
*
* Description:     Called after successful transmission or reception of the
*                  Master Request Command Frame (ID Field value '0x3C').
*
* Returns:         never returns
*
*****************************************************************************/

void LIN_Command()
{
    while(1)
    {
    }
}
```

Appendix A Include File (register definitions for the MC68HC908EY16)

```
/*****************************************************
  HC08EY16.H
  Register definitions for the 908EY16

  P. Topping                                24-01-02
*****************************************************/
```

```
#define PTA *((volatile unsigned char *)0x0000)
#define PTB *((volatile unsigned char *)0x0001)
#define PTC *((volatile unsigned char *)0x0002)
#define PTD *((volatile unsigned char *)0x0003)
#define PTE *((volatile unsigned char *)0x0008)

#define DDRA *((volatile unsigned char *)0x0004)
#define DDRB *((volatile unsigned char *)0x0005)
#define DDRC *((volatile unsigned char *)0x0006)
#define DDRD *((volatile unsigned char *)0x0007)
#define DDRE *((volatile unsigned char *)0x000A)

#define CONFIG1 *((volatile unsigned char *)0x001F)
#define CONFIG2 *((volatile unsigned char *)0x001E)

#define TBCR   *((volatile unsigned char *)0x001C)

#define TBSC   *((volatile unsigned char *)0x002B)
#define TBCNTH *((volatile unsigned char *)0x002C)
#define TBCNTL *((volatile unsigned char *)0x002D)
#define TBMODH *((volatile unsigned char *)0x002E)
#define TBMODL *((volatile unsigned char *)0x002F)
#define TBSC0  *((volatile unsigned char *)0x0030)
#define TBCH0H *((volatile unsigned char *)0x0031)
#define TBCH0L *((volatile unsigned char *)0x0032)
#define TBSC1  *((volatile unsigned char *)0x0033)
#define TBCH1H *((volatile unsigned char *)0x0034)
#define TBCH1L *((volatile unsigned char *)0x0035)

#define ICGCR  *((volatile unsigned char *)0x0036)
#define ICGMR  *((volatile unsigned char *)0x0037)
#define ICGTR  *((volatile unsigned char *)0x0038)

#define VECTF (void(*const)()) /* Vector table function specifier */
```

Appendix B Vector.c

```
#define VECTOR_C
/*****************************************************************************
*
*       Copyright (C) 2001 Motorola, Inc.
*
* Functions:    Vectors table for LIN08 Drivers with Motorola API
*
* History:      Use the CVS command log to display revision history
*               information.
*
* Description:  Vector table and node's startup for HC08.
*               The users can add their own vectors into the table,
*               but they should not replace LIN Drivers vectors.
*
* Notes:
*
*****************************************************************************/

#if defined(HC08)                       /* for HC08 */

#if defined(HC08EY16)

extern void LIN_ISR_SCI_Receive();      /* ESCI receive ISR             */
extern void LIN_ISR_SCI_Error();        /* ESCI error ISR               */
extern void TimerB();                   /* Timer Module B Overflow ISR  */

#endif /* defined(HC08EY16) */


/*****************************************************************************
    NODE STARTUP
    By default compiler startup routine is called.
    User is able to replace this by any other routine.
*****************************************************************************/

#if defined(HICROSS08)

#define Node_Startup    _Startup
extern void _Startup();                 /* HiCross compiler startup routine declaration */

#endif  /* defined(HICROSS08) */
```

```
/*****************************************************************************
    INTERRUPT VECTORS TABLE
    User is able to add another ISR into this table instead NULL pointer.
*****************************************************************************/

#if !defined(NULL)
#define NULL    (0)
#endif /* !defined(NULL) */

#undef  LIN_VECTF

#if defined(HICROSS08)

#define LIN_VECTF ( void ( *const ) ( ) )
#pragma CONST_SEG VECTORS_DATA              /* vectors segment declaration */
void (* const _vectab[])( ) =

#endif  /* defined(HICROSS08) */


#if defined(HC08EY16)

/*****************************************************************************/
/*      HC08EY16                                                             */
/*                                                                           */
/*      These vectors are appropriate for the following MC68HC908EY16       */
/*      mask sets:-   0L38H, 1L38H, 0L31N,and 1L31N                          */
/*      These mask sets had a fault in their interrupt vector table and      */
/*      hence in the interrupt priorities.                                   */
/*                                                                           */
/*      For the vector address in the corrected mask set (2L31N) see         */
/*      the MC68HC908EY16 technical data sheet.                              */
/*                                                                           */
/*****************************************************************************/


{
    LIN_VECTF NULL,                         /* 0xFFDC   Timebase          */
    LIN_VECTF NULL,                         /* 0xFFDE   SPI transmit      */
    LIN_VECTF NULL,                         /* 0xFFE0   SPI receive       */
    LIN_VECTF NULL,                         /* 0xFFE2   ADC               */
    LIN_VECTF NULL,                         /* 0xFFE4   Keyboard          */
    LIN_VECTF LIN_ISR_SCI_Error,            /* 0xFFE6   ESCI error        */
#if defined(MASTER)                         /* (used for Master node only)*/
    LIN_VECTF LIN_ISR_SCI_Transmit,         /* 0xFFE8   ESCI transmit     */
#endif /* defined(MASTER) */
#if defined(SLAVE)
    LIN_VECTF NULL,                         /* 0xFFE8   ESCI transmit     */
#endif /* defined(SLAVE) */
    LIN_VECTF LIN_ISR_SCI_Receive,          /* 0xFFEA   ESCI receive      */
    TimerB,                                 /* 0xFFEC   TIMER B overflow  */
    LIN_VECTF NULL,                         /* 0xFFEE   TIMER B channel 1 */
    LIN_VECTF NULL,                         /* 0xFFF0   TIMER B channel 0 */
    LIN_VECTF NULL,                         /* 0xFFF2   TIMER A overflow  */
    LIN_VECTF NULL,                         /* 0xFFF4   TIMER A channel 1 */
#if defined(MASTER)                         /* (used for Master node only)*/
    LIN_VECTF LIN_ISR_Timer0,               /* 0xFFF6   TIMER A channel 0 */
#endif /* defined(MASTER) */
#if defined(SLAVE)
    LIN_VECTF NULL,                         /* 0xFFF6   TIMER A channel 0 */
#endif /* defined(SLAVE) */
    LIN_VECTF NULL,                         /* 0xFFF8   CMIREQ            */
    LIN_VECTF NULL,                         /* 0xFFFA   IRQ               */
//  LIN_VECTF BREAK_Command,                /* 0xFFFC   SWI               */
    LIN_VECTF NULL,                         /* 0xFFFC   SWI               */
    LIN_VECTF Node_Startup                  /* 0xFFFE   RESET             */
};

#endif  /* defined(HC08EY16) */


#if defined(HICROSS08)
#pragma CONST_SEG DEFAULT
#endif  /* defined(HICROSS08) */

#endif  /* defined(HC08) */
```

Appendix C Slave.cfg (LIN configuration file)

```
#ifndef LINCFG_H
#define LINCFG_H

/*******************************************************************************
*
*       Copyright (C) 2001 Motorola, Inc.
*
* Functions:    LIN Driver static configuration file for LIN08 Slave sample
*               with Motorola API
*
* Notes:
*
*******************************************************************************/

#if defined (HC08)


/*
    This definition configures the LIN bus baud rate.
    This value shall be set according to target MCU
    SCI register usage.
    HC08AZ32: the 8-bit value will be masked by 0x37
    and put into SCBR register.
*/

/* Selects 9600 baud rate if using a 9.8304MHz crystal */
//#define LIN_BAUDRATE            0x04u

/* Selects 9600 baud rate if using a 8MHz crystal */
#define LIN_BAUDRATE            0x30u

/*
    This definition set the number of user-defined time clocks
    (LIN_IdleClock service calls), recognized as "no-bus-activity"
    condition.
    This number shall not be greater than 0xFFFF.
*/
#define LIN_IDLETIMEOUT         400u

#endif /* defined (HC08) */

#endif /* !define (LINCFG_H) */
```

Appendix D Slave.id (LIN message ID file)

```
#ifndef LINMSGID_H
#define LINMSGID_H

/*******************************************************************************
*
*       Copyright (C) 2001 Motorola, Inc.
*
* Functions:    Message Identifier configuration for LIN08 Slave sample
*               with Motorola API
*
*
* Notes:
*
*******************************************************************************/

#define LIN_MSG_20  LIN_RECEIVE   /*  Keypad slave ID    */
#define LIN_MSG_30  LIN_RECEIVE   /*  Master's slave ID  */


/* this string is not necessary - just as an example */

#define LIN_MSG_20_LEN        4          /* standard length */
#define LIN_MSG_30_LEN        8          /* standard length */

#endif /* defined(LINMSGID_H)*/
```

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN2470
Rev. 1
01/2007

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

freescale™
semiconductor