Application Note

AN2560/D
9/2003

MC68HC908EY16 IR Receiver for Remote Control of LIN Robot

By: Graham Brown
Motorola TSPG,
East Kilbride, Scotland

Introduction

This document describes an application that uses the EY16 LIN demonstration board. The board contains an MC68HC908EY16 microcontroller and a LIN interface IC, along with a number of other components used for voltage regulation, monitor-mode communication, and for the MCU oscillator. The board acts as an infrared (IR) receiver that gathers data from a standard TV remote-control. It uses the data to control other devices over the LIN bus. In this application, it is used to control a robot arm.

The robot is described in more detail in AN2470/D, *MC68HC908EY16 Controlled Robot Using the LIN Bus*. The LIN IR receiver board can be connected in place of the keypad that was used to control the robot in AN2205/D, *Car Door Keypad Using LIN*. Two other application notes, AN2343/D and AN2432/D, cover LIN monitors based on the LIN demonstration board. These application notes and other helpful documents are listed in the **References** section.

This application demonstrates the following techniques:

- Using input capture interrupts to collect and decode an IR data stream
- Using the SPI module to clock data into shift registers for display on LEDs
- Using the EY16 demo board and Motorola/Metrowerks LIN driver software to control a robot arm

NOTE: *With the exception of mask set errata documents, if any other Motorola document contains information that conflicts with the information in the data sheet, the data sheet should be considered to have the most current and correct data.*

RC-5 Protocol

The Philips RC-5 remote control protocol is used in this application, enabling a standard TV remote-control to control the robot. The RC-5 transmitter uses a 14-bit code to send data to the receiver.

The message sent by the transmitter is encoded using the Manchester biphase method. The encoding uses a rising edge in the center of a bit to represent a logic 1, and a falling edge in the center of a bit to represent a logic 0. **Figure 1** shows the structure of an RC-5 message. Part (a) shows that there is a spacing of slightly less than 90 ms between each message. Part (b) shows a message in greater detail. The expanded section of this waveform shows that the high period of each data bit is modulated with a 36-kHz square wave that has a duty cycle of 25 percent. Part (c) shows the signal received by the MCU pin, due to the use of an inverting sensor IC which also removes the 36-kHz modulation. (More information on the inverting sensor IC is given in the section entitled **Application Circuit**.) Because the signal is inverted by the sensor, this application interprets a low-to-high transition as a logic 0 and a high-to-low transition as a logic 1 (opposite of the RC-5 specification).

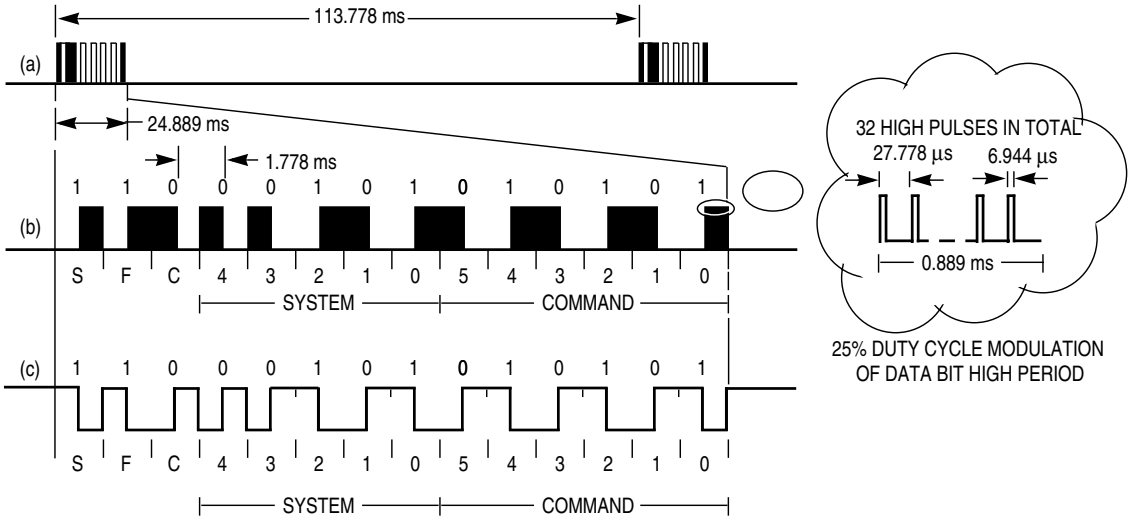


Figure 1. RC-5 Waveforms

The spacing between the individual bits is sufficient to allow the EY16 MCU to process the data bits as they occur. The space between messages allows time for the program to process the received message before the next message arrives. The message is divided into different components, as detailed in **Table 1**.

Table 1. Components of the RC-5 Message

Bit	Purpose
Start (S)	Start bit, always 1
Field (F)	Complement of the MSB (bit 6) of the command code (e.g., 1 for commands 0 – 63 and 0 for commands 64 – 127)
Control (C) (Toggle)	Toggles on each alternate press of a button to allow receiver to distinguish between a button being held down and a button being pressed and released twice in succession
System	Five bits used to select the system for which the command code is intended (in this case, 00000 = TV1, 00101 = VCR1)
Command	Bits 5 – 0 of the actual command code (e.g., 010000 = volume +)

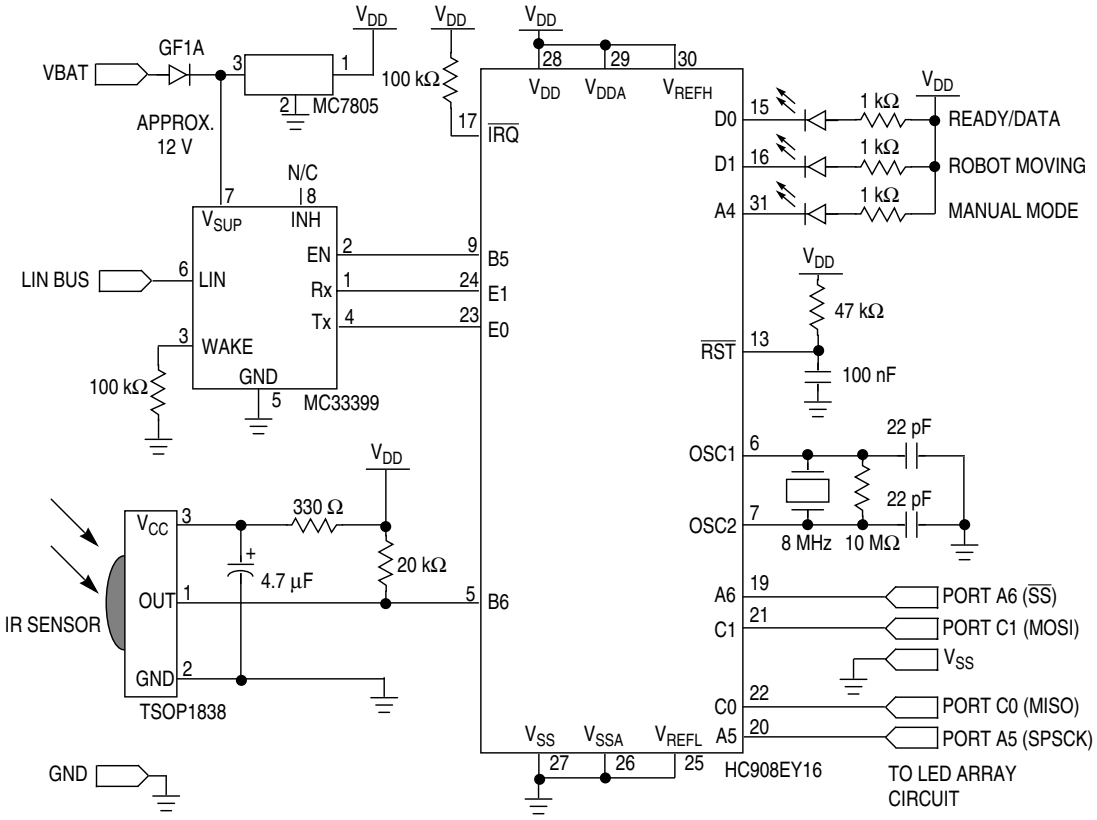
The input capture module is used to examine the timing between edges in the data stream and thereby identify the data bits. This method takes advantage of the incoming signal's embedded clock. The embedded clock is present because of the biphase technique used in the encoding process. The receiver can therefore re-synchronize itself to the data timing as each bit is received. The decoding algorithm is discussed in greater detail later on.

Application Circuit

An IR sensor is required for detecting the signal from the transmitter. For this application, a dedicated IR receiver IC was selected instead of a photodiode. This is because the IR receiver IC manages the optical filtering, pre-amplification, and removal of the 36-kHz carrier frequency. Using the dedicated IR receiver IC reduces the software required to reliably decode the signal. It also outputs a clean waveform that is, in practice, almost identical to the waveform shown in **Figure 1**(c).

The Temic TSOP1838 receiver IC was selected for this application. Although this device is optimized for a 38-kHz modulated wave (whereas the RC-5 transmitter uses 36 kHz), the performance of the sensor is extremely good even when positioned more than 10 meters away from the transmitter. One disadvantage of the IC's high sensitivity is susceptibility to interference (e.g., fast changes in background light level or flashes of light). However, the MCU receive routine is designed to cope with this by aborting and re-starting the message-receive process if a partial message or a timing issue is found.

The IC was used with the external circuit recommended in the Temic TSOP1838 data sheet (**Reference [6]**). The circuit used in this application (excluding the circuitry used for monitor mode entry and communication) is shown in **Figure 2**. This circuitry consists of the components on the LIN demo board and the IR receiver circuit. The full EY16 LIN demo board schematic is shown in AN2432/D (see **Reference [4]**).



For clarity, decoupling capacitors and monitor mode entry connections/components are not shown on this diagram.

Figure 2. Circuit Used in this Application

Application Code

A good way to design a LIN interface using the Motorola/Metrowerks LIN drivers is to clone an existing LIN project. The CodeWarrior® project used for this application was cloned from the EYLEDemo project, part of AN2432/D (**Reference [4]**). (AN2432/D gives instructions for cloning a project.) As part of the cloning process, the LEDemo.c file in the EYLEDemo project was replaced with the LINnode.c file, and the necessary modifications were made to the vector.c file. (These files are shown in **Appendix A – LINnode.c** and **Appendix B – Vector.c**.) The timing values defined at the beginning of the program are meant to be used with an 8-MHz crystal, although values for 9.8304 MHz are also given and are commented out in the program listing.

The code used in this application can be divided into three types:

- LIN driver
- Main program
- IR receiver

The LIN driver and the IR receiver processes work in the background, handling communications with the LIN bus and IR interface, respectively, while the main user code carries out whatever task the user requires.

The main program is used to allow a robot connected to the LIN bus to be controlled using the IR remote-control. The main program takes the message from the IR receiver and compares it to a list of messages assigned to controlling robot axes/functions. If a match is found, it puts the appropriate data into the LIN driver buffer. The data is then sent via the LIN bus to the robot to make it move as required.

The LIN driver used is the Motorola/Metrowerks driver, which has been covered in previous application notes related to the LIN protocol (see the **References** section). The LIN driver's operation is transparent to the main program after it has been initialized by calling the LIN_Init() function. Then, data can be sent to the LIN driver by calling the LIN_PutMsg function. The IR receiver code has also been designed to operate entirely through timer B interrupt service routines (ISR), so it is transparent to the user after it has been initiated. At the beginning of the main program, the user calls a function which starts off the IR receive process. The process then continues without further intervention by the main program. The receive process resets itself if a timing error is detected, and the erroneous data is cleared without being made available. This process detects timing errors only; the RC-5 protocol has no parity bits included, so the integrity of the actual data bits cannot be ensured. A buffer (a 16-bit variable named MESSAGE) is used to transfer data between the IR receiver and the main program. A flag named IR_NEW_DATA is used

to indicate to the main program that new data is available, and the main program can read this data from the MESSAGE buffer.

More information on each part of the program is given in the following sections. **Diagnostic LEDs** covers the 16-LED array which was connected to the SPI port to assist with code development.

Functions

Here is a brief overview of the five main functions/ISRs in this application:

Main — This function contains a loop which runs continuously. It takes new data from the IR receive buffer (when available), compares it to a list of robot control commands, and sets different data bits in the LIN message to control the robot. Calls to the LIN driver functions are used to output this data onto the LIN bus.

IC_Start_Edge — The program uses this function to start the IR receive process. It sets the input capture on timer B channel 0 to detect a falling edge. The receive process initializes itself when a message is complete or when an error occurs, so this function is used only one time after the program begins.

TimerB_Input_Capture — The program uses this ISR every time the required edge is detected on the input pin. Almost all of the IR receiver activity is carried out by this ISR. When searching for data bits, this ISR analyzes the timing of the edges. It uses this information to interpret the data bits being received. After a data message has been fully received or if an error occurs in the middle of a message, this ISR sets up the input capture to detect a new start edge.

TimerB_Overflow — This ISR is used only when the IR receiver is in the middle of a message and no rising edges occur within the maximum time permitted. This constitutes a timing error, so the message currently being received has an error. This ISR cancels the current receive process and sets the input capture to detect a new start edge.

SPI_Transmit — This function is used to send the contents of a 16-bit variable (LED_VALUE) to two 8-bit shift registers connected to the SPI port of the MCU. Each shift register has 8 LEDs connected to its parallel output pins, and so the 16 bits of data are clocked into these LEDs by the SPI. In the final application, the 14-bit RC-5 code received by the IR Receiver routine is put out onto these LEDs, allowing the user to see the individual bits of the code received while any button on the remote control is held down. The code received is displayed, even when the button is not one of the buttons used to control the robot. This allows the code sent by any button to be seen easily.

Robot Control/LIN Bus Interface

The robot, which is covered in greater detail in **Reference [1]**, is controlled using messages sent through the LIN bus. There are two different message IDs that can be used to control the robot, ID20 and ID30.

For the robot application, ID20 messages are used to allow manual control. The LIN master board sends out ID20 message frames and a slave controller (keypad or this IR receiver board) can fill in the 4 bytes of the data response field of the message frame being received by the robot. Every servo has 4 commands, each of which has a separate bit dedicated to it in the first 3 bytes of the ID20 LIN message (See **Table 2** and **Reference [1]**). These bits are used to command the servo to turn clockwise (slow), clockwise (fast), counterclockwise (slow), and counterclockwise (fast). The robot servo will continue to move in the selected direction at the selected speed until the bit is cleared in the message or until the robot servo reaches the end of its allowable travel.

The MSB of the fourth byte is used to select master/slave (manual) control. When this bit is cleared, the robot is controlled by the relevant bits of ID20 message as described above. The rest of the fourth byte is not modified by this application. However, if a slave node sets this bit, the master takes control of the robot directly using ID30 messages by sending positional information to the servos. The IR receiver does not send or modify any of the ID30 messages. The ID30 message structure is discussed in more detail in the aforementioned robot application note. In this application, the master/slave control bit in the fourth byte is given a latching action, with two separate buttons on the handset assigned to setting and clearing of this bit. It is not affected by any other commands from the remote control transmitter.

Main Program

The main program begins by carrying out the initialization of the LIN driver, the IR receiver, and the other MCU modules required by the application. A call to the Lin_Init() function is used to start the LIN driver, and a call to the IC_Start_Edge function is used to set up the input capture to receive its first start edge.

The tasks carried out by the main program are summarized below. They are carried out continuously while the program is running, and are initiated by polling the timebase module (TBM) overflow flag.

- Check whether a new IR message has been received by the IR receiver routine at 2.05 ms intervals. If a new IR message has been received, and it is one of the codes for controlling the robot, manipulate the bits of the LIN message accordingly.
- If no new IR code is received for 130 ms, clear the LIN messages to stop the robot moving. This stops the robot after the user releases the button on the remote control.

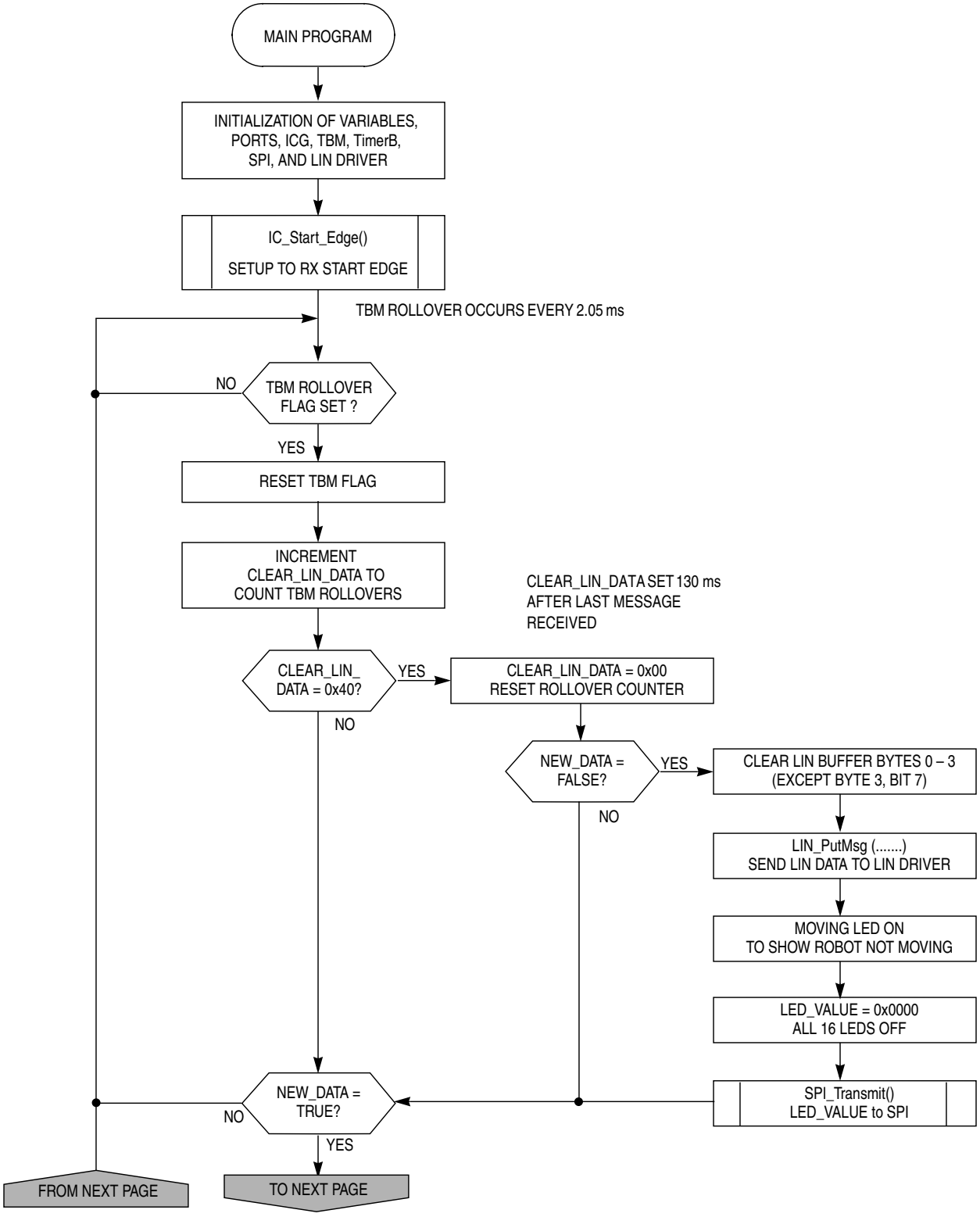
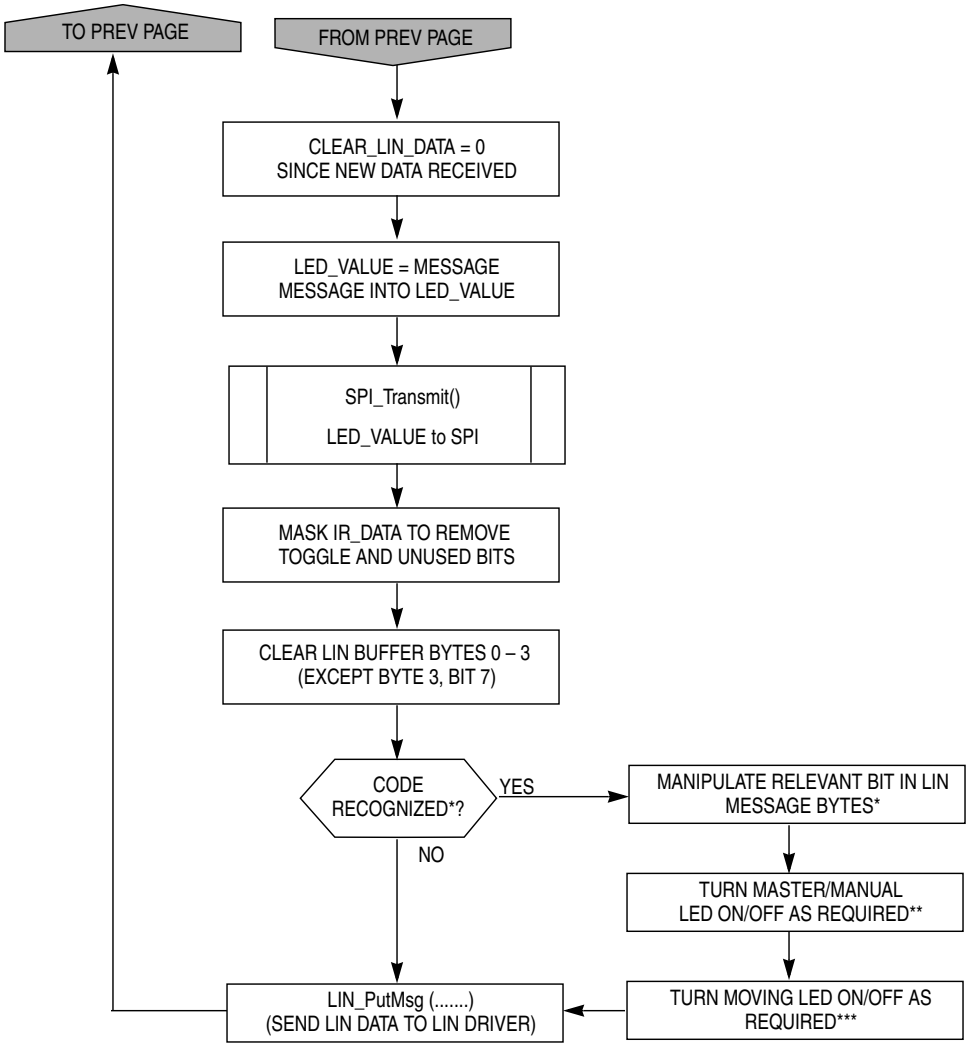


Figure 3(a). Main Program Flowchart (Continued on Next Page)



* See **Table 2** for the RC-5 codes and LIN bits corresponding to each robot function.

** The master/manual LED is turned on when a manual mode select code is received (code 1 or 2 in **Table 2**). It is turned off when a master mode select code is received (code 3 or 4 in **Table 2**). It is not affected by any other codes received.

*** The moving LED is turned off when any code is received which causes a robot servo to mode (codes 5 – 24 in **Table 2**). It is left on when any other codes are received.

Figure 3(b). Main Program Flowchart (Continued from Previous Page)

When a new message is received, the program must determine whether the message is a robot command. The two MSBs of the 16-bit MESSAGE data are not used (the RC-5 message is only 14 bits long) and the state of the C bit (toggle bit) of the message may be a 0 or 1. To ensure that these three bits of the message are in a known state, the MESSAGE value is ANDed with 0x37FF

to clear these bits. Now that the message is in a standard form, it can be compared to all of the known robot control codes using a switch-case statement. The codes for controlling the robot are shown in **Table 2**.

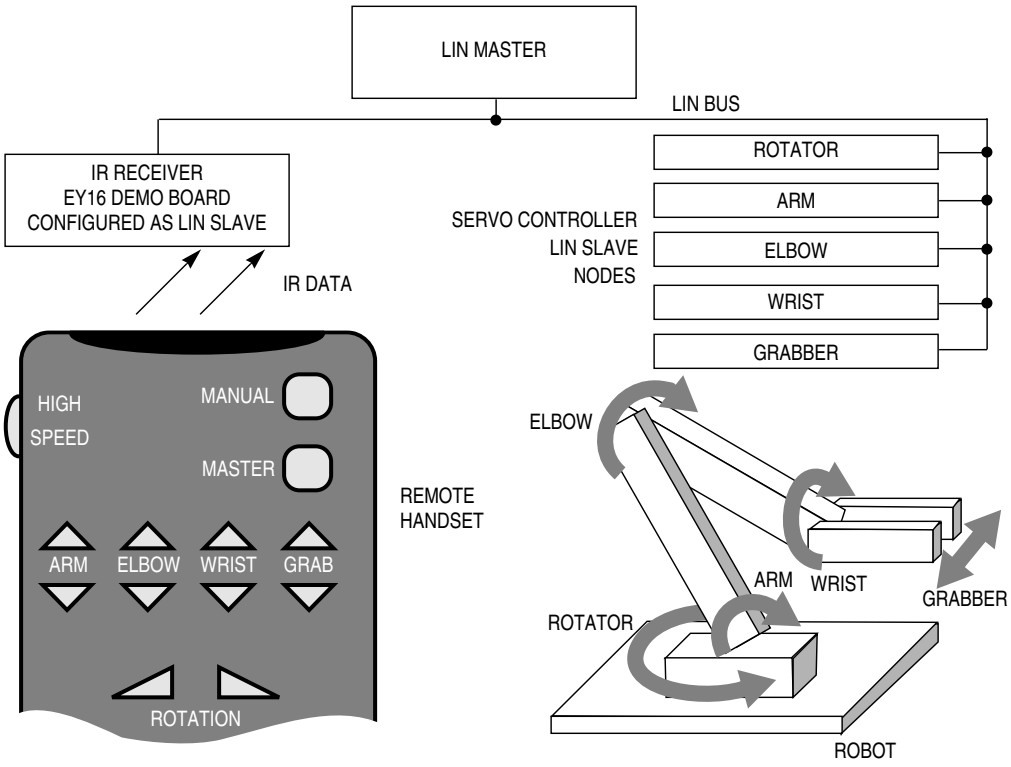


Figure 4. LIN Bus Setup and Remote Handset Controls

The original robot keypad design (which uses a car door keypad, see **Reference [2]**) has a *normal* and an *express* function for each control key. The normal mode makes the robot move in the desired direction at slow speed; the express mode makes the robot move in the same direction but at high speed. The codes shown in **Table 2** were chosen because their corresponding buttons on the remote control have arrow style keys, making the operation of the robot more intuitive.

In the IR controller, the slow-speed movement of each servo is controlled using a pair of arrow keys. An additional pair of keys is used to select manual mode and master mode. When any of these keys are pressed, they send an RC-5 message to the receiver, with a system code of 00000 (TV1) and with the appropriate command code in the lower six bits. To select high-speed movement, the VCR mode button is pressed on the remote-control while the relevant button is pressed. This sends the same command code as the TV code above, but this time with a system code of 00101 (VCR1). The receiver master, manual, and grabber open/close buttons may be used with or without the high-speed button pressed. Therefore, the receiver must accept these commands with either the VCR or TV code in the system field.

Table 2. LIN Data Bytes Used to Control the Robot

No.	RC-5 Code*	Robot Function	LIN Response Data Bytes			
			Byte 0	Byte 1	Byte 2	Byte 3
1	3020	Manual control	00000000	00000000	00000000	0XXXXXXX
2	3160	Manual control	00000000	00000000	00000000	0XXXXXXX
3	3021	Master control	00000000	00000000	00000000	1XXXXXXX
4	3161	Master control	00000000	00000000	00000000	1XXXXXXX
5	3011	Rotate left slow	00001000	00000000	00000000	-XXXXXXX
6	3151	Rotate left fast	00000010	00000000	00000000	-XXXXXXX
7	3010	Rotate right slow	00000100	00000000	00000000	-XXXXXXX
8	3150	Rotate right fast	00000001	00000000	00000000	-XXXXXXX
9	3014	Arm up slow	01000000	00000000	00000000	-XXXXXXX
10	3154	Arm up fast	00010000	00000000	00000000	-XXXXXXX
11	3015	Arm down slow	10000000	00000000	00000000	-XXXXXXX
12	3155	Arm down fast	00100000	00000000	00000000	-XXXXXXX
13	3012	Elbow up slow	00000000	00000100	00000000	-XXXXXXX
14	3152	Elbow up fast	00000000	00000001	00000000	-XXXXXXX
15	3013	Elbow down slow	00000000	00001000	00000000	-XXXXXXX
16	3153	Elbow down fast	00000000	00000010	00000000	-XXXXXXX
17	3018	Wrist up slow	00000000	01000000	00000000	-XXXXXXX
18	3158	Wrist up fast	00000000	00010000	00000000	-XXXXXXX
19	3019	Wrist down slow	00000000	10000000	00000000	-XXXXXXX
20	3159	Wrist down fast	00000000	00100000	00000000	-XXXXXXX
21	3016	Grabber open	00000000	00000000	00000010	-XXXXXXX
22	3156	Grabber open	00000000	00000000	00000010	-XXXXXXX
23	3017	Grabber close	00000000	00000000	00000001	-XXXXXXX
24	3157	Grabber close	00000000	00000000	00000001	-XXXXXXX

* RC-5 code shown in hexadecimal, '–' = Not Affected, 'X' = Don't Care

NOTE: *The MSB of byte 3 (master bit) is set or cleared by sending only the relevant codes (1 – 4) to the receiver. It is therefore shown as not affected when other codes are received.*

The main loop is used to check for new IR data every 2.05 ms and to set the relevant bit in the LIN message. To stop the robot moving when the user releases the control key, the first three bytes of the LIN response data field are cleared approximately 130 ms after the last IR message is received. The 130 ms period corresponds to 64 times around the 2.05 ms loop that is initiated by the TBM flag. Because a new IR message is sent every 114 ms (before 130 ms has passed) while the user presses a button, the LIN message will only be cleared after the user has released the button. This ensures smooth continuous motion of the robot when a robot control button is pressed, but also means that the robot will stop moving soon after the button is released. A counter variable named CLEAR_LIN_DATA is incremented each time the TBM flag is set (every 2.05 ms). It is cleared to 0 every time a new message is received. If the counter reaches a value of 64, it clears the LIN message bits and the robot stops moving.

Once the MESSAGE buffer data has been analyzed, the IR_NEW_DATA flag is put back to FALSE so the IR receive routine can overwrite the data in MESSAGE with the next data it receives. The main program then returns to wait for the next TBM event.

Receiving the RC-5 Protocol

The beginning of each IR message is marked by a start edge. This is received by the MCU as a falling edge (due to the inversion in the IR sensor IC). When the IC_Start_Edge function is called at the beginning of the main code, the input capture for timer B channel 0 is set to detect a falling edge (see the flowchart in **Figure 5**(a)). The input capture is set up to detect a new start edge at the end of receiving each message or detecting an error. The code then returns to the main program until the falling edge occurs, at which point the TimerB_Input_Capture ISR is called.

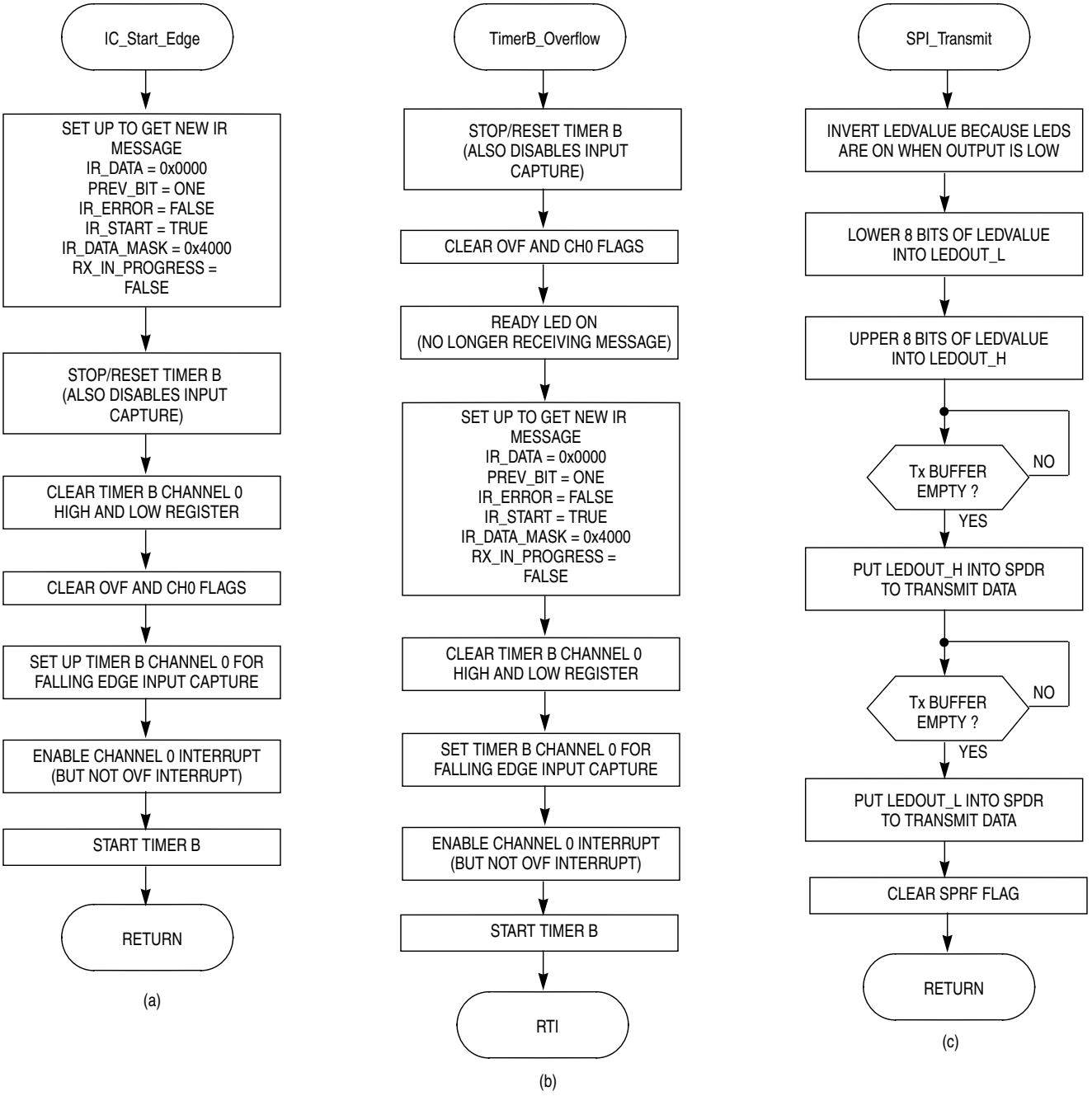


Figure 5. Flowcharts for IC_Start_Edge, TimerB_Overflow, and SPI_Transmit

A variable named IR_START is used to show whether the program is waiting for a start edge or a data bit. When capturing a start edge, the time value in the input capture register is ignored. Only the timing between edges *within* a message is relevant to the decoding process.

The receive routine regards the start bit as an actual data bit and uses the same process to determine its value. The program must monitor the data bits as they are received. This is done using a 16-bit variable named IR_DATA_MASK. The variable has a single bit set to 1, and the position of the 1 in this variable shows the status of the receive process. It is initialized to 0x4000 (0100 0000 0000 0000) at the beginning of the receive procedure for each message. As each bit is received, the 1 is shifted one place to the right. When it rolls off the end, all bits have been received. Also, because the 1 is in the position of the bit which is currently being received, IR_DATA can be ORred with the mask to put a 1 in the relevant position if this data bit is found to be a 1. If the data bit is a 0, no OR is performed, leaving the bit as a 0 in IR_DATA.

Receiving the 14 RC-5 Data Bits

The start edge ISR also sets the input capture to detect rising edges because only rising edges will be captured to decode the remaining 14 bits of the message. The decoding of the message uses the five valid conditions shown in **Table 3**. To get the information required for these rules, the input capture is used (time between edges) along with the PREV_BIT variable (previous data bit value). Acceptable time values fall into three ranges, each of which corresponds to an integer number of RC-5 protocol half-bit times (HBT). Only the upper byte of the 16-bit timer register value is used, which reduces the overhead when comparing the timer value with the time range for two, three, and four HBT. The level of accuracy provided by the full 16-bit timer value is not required in this application. The start of the program listing contains definitions for the maximum and minimum timer input capture values that are allowed for two, three, and four HBT. For example, for rule A in **Table 3** to be met, the timer value must be between HBT_2L and HBT_2H and PREV_BIT must be 1.

Table 3. RC-5 Decoding Rules

Rule	HBT Since Last Rising Edge	Previous Data Bit	Current Data Bit(s)
A	2	1	1
B	3	0	1
C	2	0	0
D	3	1	1 then 0
E	4	0	1 then 0

Each edge may indicate one or two data bits, as shown in **Table 3**. Before setting up to detect each edge, the timer is stopped and reset to 0000. The input capture interrupts are then enabled so the next rising edge can be captured.

Bit Timing

Now that the actual data bits are being received, there are only three time ranges between rising edges that allow one of the above rules to be met. Due to the Manchester biphase encoding used in the IR transmission, there must always be a rising edge at least every 4 HBT in a valid RC-5 message (every 4 x 0.889 ms = 3.556 ms). If no rising edge is detected for more than 4 HBT in the middle of a message, an error has occurred. This could happen if the IR light beam is interrupted during a message, or if the falling edge interpreted as the start edge was actually caused by noise. In this case, the receive routine should be reset to detect the next message. To do this, the timer B modulo value is set to be slightly more than 4 HBT. The timer is cleared when each valid edge is detected. Sometimes the program waits for a rising edge, but one does not occur before the timer reaches the modulo value and overflows. In that case, the TimerB overflow ISR (see **Figure 5**(b)) will be called to initialize all the receiver variables and be set to detect a new start edge. The limit on the time between edges only applies within messages, and so the overflow interrupt is disabled when waiting for a start edge.

Rising Edge Capture

If a rising edge occurs before the timer overflows, the TimerB_Input_Capture ISR will be called. Once the edge is detected, the timer is stopped and reset. This allows the timer to start counting from 0 again when searching for the next rising edge. During the input capture ISR, the combination of the previous data bit and the time since the last rising edge are compared to the five rules mentioned previously. If none of the five rules are met, the message is invalid and the IR_ERROR flag is set TRUE. There is a special case for decoding the start bit itself, which is mentioned in the next section.

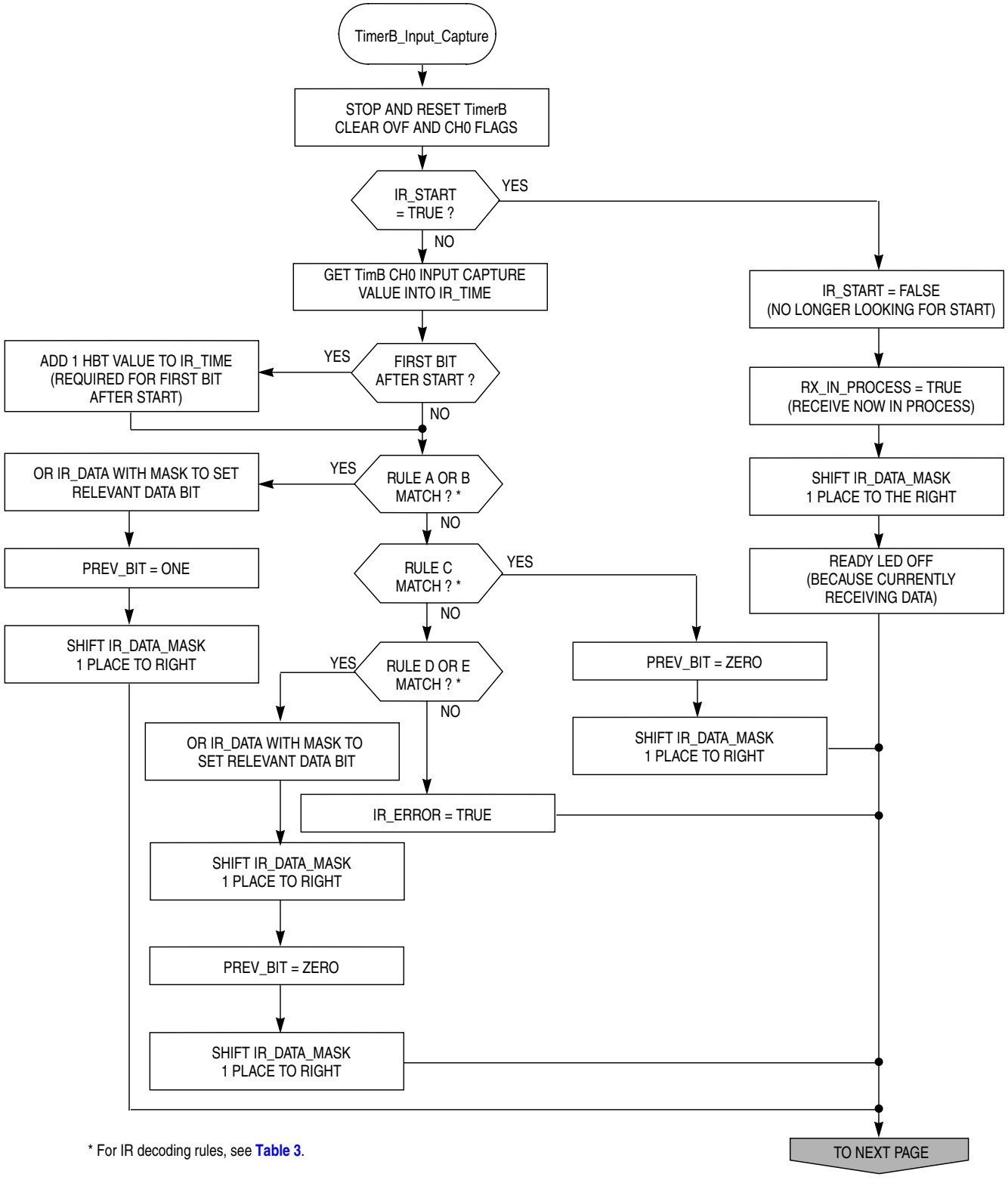


Figure 6(a). Flowchart of TimerB_Input_Capture (Continued on Next Page)

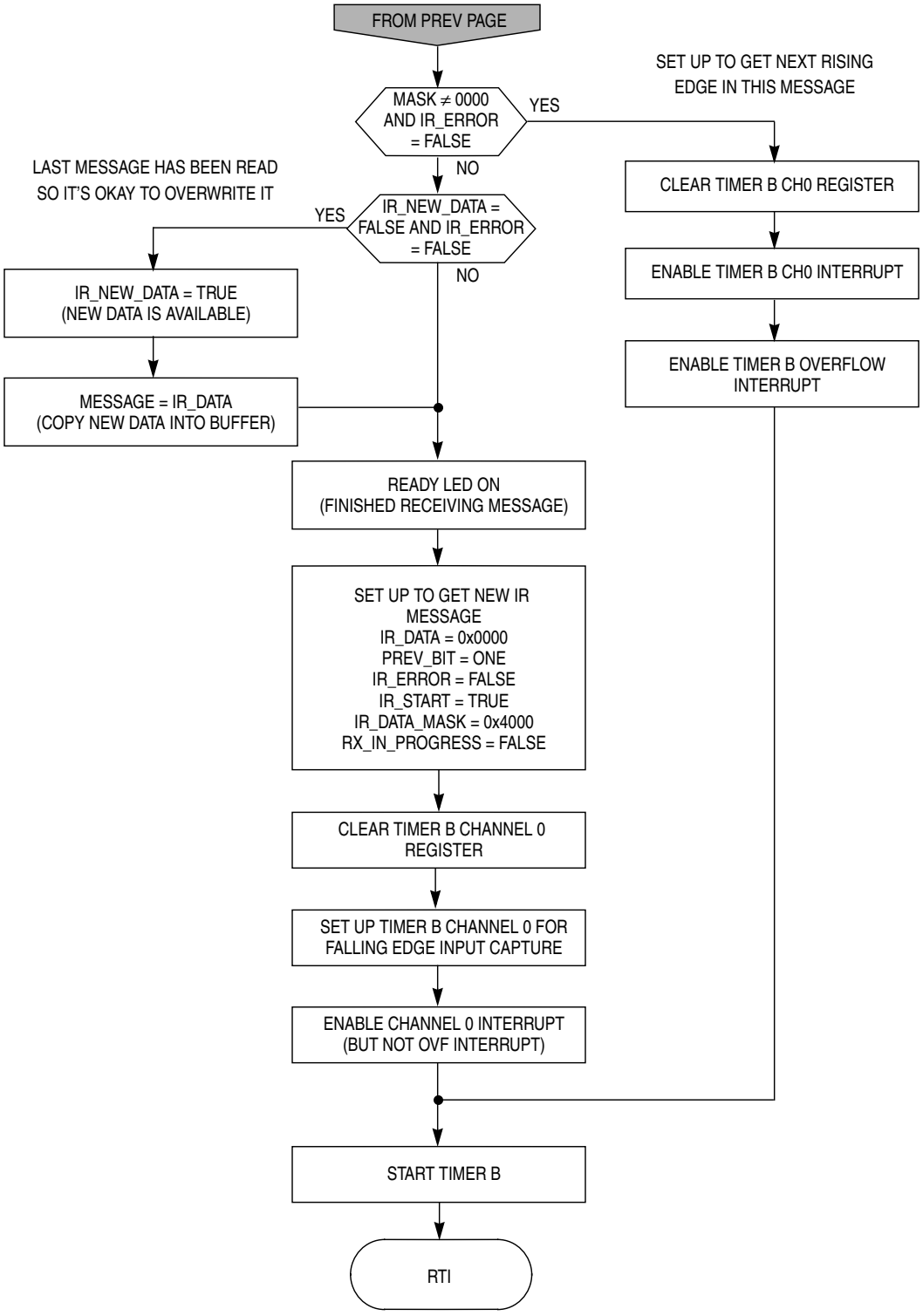


Figure 6(b). Flowchart of TimerB_Input_Capture (Continued from Previous Page)

Depending on whether the program has recorded a valid rising edge or found that an error has occurred, the final part of the input capture ISR either sets the input capture to detect the next rising edge of the current message or resets the receive process so it can receive a new message.

If the data mask is 0000, the last data bit has been received and a complete message is now ready. In this case, the ISR checks to see whether the IR_NEW_DATA flag has been set. If this was set when a previous message was received, and it has not yet been cleared to FALSE, the main program has not read the last data message. The newly received message is *not* copied into the MESSAGE buffer because this would overwrite a previous message, and the newest message is discarded. In this application, the main program should always have read the last message by the time the next one arrives, and so the flag will always be cleared by this time. In other applications, more sophisticated buffering could be used to queue new data if loss of a new message is possible or would cause problems. If the IR_NEW_DATA flag was found to be FALSE, the new data is copied into the MESSAGE buffer and the IR_NEW_DATA flag is set to TRUE to show this. Because this was the end of a message, the input capture is ready to detect a new start edge and the message receive process starts over again.

If the mask is not 0000, the IR receiver is currently receiving a message. If an error has occurred (if IR_ERROR is TRUE), the current message should be discarded and the IR receiver initialized to detect a new start edge. If an error has not occurred, the input capture is set to get the next rising edge in the message, and the input capture and timer overflow interrupts are enabled. The timer is started again to detect the next rising edge.

Receiving the Start Bit

Although the falling edge in the middle of the start bit was the first event to be detected in the message, the actual bit itself has not been recorded into the relevant position in the IR_DATA variable. Ideally, the start bit would be decoded in the same way as the other 13 message bits (even though it is known to always be 1 in this protocol), but a small change is required. The best way to decode the start bit is as a standard message bit because this maximizes the amount of re-usable code in the program.

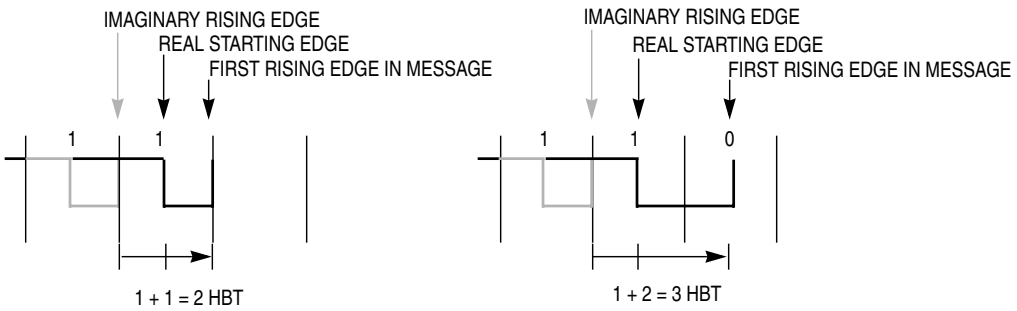


Figure 7. Receiving the Start Bit of the RC-5 Message

To decode the data bits this way, the value of the previous bit and the time since the last rising edge must be known. Because there was no data bit before the start bit, an imaginary logic 1 bit is put in before the start bit. Although this logic 1 is never used as a data bit, it provides an imaginary rising edge so that the time until the next rising edge (the first rising edge in the real data) can be measured. The first rising edge in the message will arrive one or two HBT (depending on the value of the F bit) after the start falling edge, and therefore two or three HBT after the imaginary rising edge. Combining one of these time values with the previous bit value (which was set by default to 1 because the imaginary bit was 1), the two conditions shown in **Figure 7** match rules A and D, respectively (see **Table 3**).

In practice, a value equivalent to 1 HBT must be added to the timer value obtained when the input capture ISR is called as a result of the first rising edge after the start falling edge (when IR_DATA_MASK = 0x2000). In addition to this, PREV_BIT is set to 1 by default when preparing to receive a new message.

Indication LEDs

Three LEDs on the EY16 demo board are used to indicate the status of the system. The operation of these LEDs is summarized in **Table 4**. The other two LEDs have no meaning in this application. The anode of each LED is connected to V_{DD} through a series resistor. The cathode is connected to the relevant MCU port pin. Therefore, the port pin is pulled low to turn the LED on.

Table 4. Status LEDs on the EY16 Demo Board

Port Pin	Indication	Description
D0	READY/ IR DATA	This LED is normally on to indicate that the IR receive routine is ready and waiting for data. It blinks off while data is being received.
D1	ROBOT MOVING	This LED is also normally on. It turns off while the robot is in motion (while any bit is being held high in the first three bytes of the ID20 LIN message)
A4	MANUAL MODE	This LED is on while the remote control has full control of the robot servos (through the bits in bytes 0 to 2 of the LIN ID20 message) and master control is disabled. It is off while master mode is selected (when MSB of the fourth LIN byte is 0). It is on when the program starts, since the default mode is manual.

Diagnostic LEDs

This circuit was connected to the SPI pins of the microcontroller to allow 16-bit data values to be written to a row of 16 LEDs. This enables diagnostic data to be output during the development of the program. It involves very little software overhead because the data is clocked out by the SPI module. It uses only three pins on the MCU for 16 LEDs (the $\overline{SS}$ pin is not used). The circuit is shown in **Figure 8**.

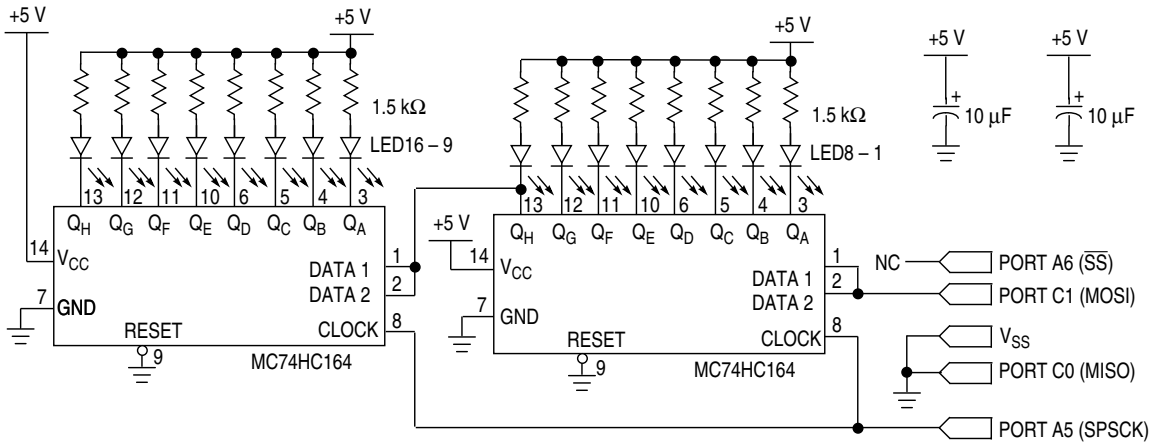


Figure 8. Circuit Used for SPI Diagnostic LED Array

The circuit consists of 16 red low-current (2 mA) LEDs (along with associated current-limiting resistors) and two MC74HC164AN shift register ICs. The shift registers are connected in series by linking the two clock pins together and by connecting the last data output of the first device to the data input of the second device. Both devices are clocked simultaneously by the SPSCK output of the microcontroller, and the data input of the first shift register is connected to the MOSI output of the MCU.

When called, the function that sends data to the LEDs takes a 16-bit value as an input and organizes this into two 8-bit values (because the SPI is designed to work with 8-bit values). A flowchart of this function is shown in **Figure 5**(c). The data value is also complemented because the LED is illuminated when the shift register output is 0 and is off when the shift register output is 1. It then sends the first eight bits to the SPI data register, followed by the next eight bits. In this application, LEDs 15 and 16 are unused. LEDs 14 – 1 show the 14 bits of the IR message received, with an illuminated LED indicating a logic 1 in the RC-5 protocol.

Conclusion

This application note has demonstrated a method of receiving and decoding an IR message using timer input capture interrupts. It has also given an example application of the EY16 LIN demo board and LIN driver software. Finally, it has presented a way of using the SPI module to implement a simple diagnostic tool. This application has been designed to receive RC-5 messages, but it should be possible to modify the decoding algorithm to allow it to be used with many of the other IR protocols.

The appendix contains the main.c and vector.c files used for the application. The other files in the CodeWarrior project are the same as those used in the LEDemo application (see **Reference [4]**).

References

[1] — AN2470/D: MC68HC908EY16 Controlled Robot Using the LIN Bus

[2] — AN2205/D: Car Door Keypad Using LIN

[3] — AN2343/D: HC908EY16 LIN Monitor

[4] — AN2432/D: LIN Sample Application for the MC68HC908EY16 Evaluation Board

[5] — MC68HC908EY16/D: Motorola's EY16 Data Sheet

[6] — Temic TSOP1838 Data Sheet

[7] — MC74HC164A/D: ON Semiconductor MC74HC164A Data Sheet

Acronyms

HBT — Half-bit time

IR — Infra-red

ISR — Interrupt service routine

LIN — Local interconnect network

MISO — Master in slave out (SPI line)

MOSI — Master out slave in (SPI line)

MSB — Most significant bit

RC-5 — Philips remote control protocol

RTI — Return from interrupt

SPI — Serial peripheral interface

SPSCK — SPI serial clock (SPI line)

SS — Slave select (SPI line)

TBM — Timebase module

Appendix A – LINnode.c

```
/*****************************************************************************************
*                        Copyright (c) Motorola 2003
*
*
*                    IR Receiver for controlling LIN Robot
*                    =====================================
*
*   Originator: G. Brown
*   Date:       January 2003
*   Revision:   1.0
*   Function:   IR Receiver for the Philips RC-5 Protocol using an EY16 LIN evaluation board.
*               The commands from the remote control are used to control a robot.
*
*   LED use:    D4 (port D, bit 0): READY / IR DATA
*                                   ON  - Normally ON after power-up
*                                   OFF - Blinks OFF when IR message being received
*               D5 (port D, bit 1): ROBOT IN MOTION (MOVING)
*                                   ON  - Normally ON after power up
*                                       - Robot is not commanded to move (no valid
*                                         key pressed on remote control)
*                                   OFF - Blinks OFF when LIN message has a bit set in it to make
*                                         a servo move...Robot is moving to a new position
*                                       - Remains OFF while a valid key is pressed on remote
*                                         control handset
*               D6 (port A, bit 4): MANUAL CONTROL
*                                   ON  - remote control (LIN ID 20) has control over the robot
*                                   OFF - LIN master (LIN ID 30) has control over the robot
*               D7 (port A, bit 5): SPI CLOCK PIN
*                                   ON  - Always ON
*                                       - ON when SPI clock line low, does not indicate anything
*               D8 (port A, bit 6): NOT USED
*                                   OFF - Always OFF
*
*  N.B. All LEDs have anode connected (through series resistor) to Vdd and Cathode connected
*       to MCU port pins. LED is therefore ON when port pin is LOW.
*
*****************************************************************************************/
```

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

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.

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

/*****************************************************************************************
Port Usage

DDRA = 0111 1000
Bit 6 = output  = SPI /SS......LED D8
Bit 5 = output  = SPI CLOCK....LED shift register CLOCK and LED D7
Bit 4 = output  = LED D6
Bit 3 = output  = no connection to this pin
Bit 2 = input   = no connection to this pin
Bit 1 = input   = used in MON mode entry
Bit 0 = input   = used in MON mode entry

DDRB = 0010 0000
Bit 7 = input   = no connection to this pin
Bit 6 = input   = Timer B CH0..input from IR sensor IC
Bit 5 = output  = LIN transceiver ENABLE output
Bit 4 = input   = used in MON mode entry
Bit 3 = input   = used in MON mode entry
Bit 2 = input   = no connection to this pin
Bit 1 = input   = no connection to this pin
Bit 0 = input   = no connection to this pin

DDRC = 0000 0110
Bit 4 = input   = OSC1.........8MHz crystal
Bit 3 = input   = OSC2.........8MHz crystal
Bit 2 = output  = MCLK output
Bit 1 = output  = SPI MOSI.....LED shift register DATA
Bit 0 = input   = SPI MISO.....grounded

DDRD = 0000 0011
Bit 1 = output  = LED D1
Bit 0 = output  = LED D0

DDRE = 0000 0000
Bit 1 = input   = no connection to this pin
Bit 0 = input   = no connection to this pin

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

```
/*****************************************************************************************
Header file includes and globals
*****************************************************************************************/

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

/******* Definitions *******/

#define  TRUE         0xFF
#define  FALSE        0x00
#define  ON           TRUE
#define  OFF          FALSE
#define  ONE          TRUE
#define  ZERO         FALSE

/******* Definitions to set IR receiver operating parameters....8.0MHz *******/

/* The time is measured in Half-bit times (HBT), where 1 HBT is the time taken for 1 half-bit   */
/* in the RC-5 protocol. This is the unit of measurement used when decoding the RC-5 message in */
/* this program. The RC-5 specification states that 1 bit-time is equal to 1.778ms, and so one  */
/* Half-bit time is equal to approximately 0.889ms. The timer values corresponding to multiples */
/* of 2, 3 and 4 HBT are shown below (for 8MHz)                                                 */

#define     HBT_1        0x07                    /* Timer value equal to 1 HBT               */
#define     HBT_2L       0x08                    /* Lower limit allowed for 2 HBT            */
#define     HBT_2H       0x0F                    /* Upper limit allowed for 2 HBT            */
#define     HBT_3L       0x10                    /* Lower limit allowed for 3 HBT            */
#define     HBT_3H       0x17                    /* Upper limit allowed for 3 HBT            */
#define     HBT_4L       0x18                    /* Lower limit allowed for 4 HBT            */
#define     HBT_4H       0x1D                    /* Upper limit allowed for 4 HBT            */
#define     MAXCOUNT_H   0x1E                    /* Timer ticks before TIMEOUT (high byte)   */
#define     MAXCOUNT_L   0x1C                    /* Timer ticks before TIMEOUT (low byte)    */

/******* Definitions to set IR receiver operating parameters....9.8304MHz *******/

//#define   HBT_1        0x09                    /* Timer value equal to 1 HBT               */
//#define   HBT_2L       0x0A                    /* Lower limit allowed for 2 HBT            */
//#define   HBT_2H       0x13                    /* Upper limit allowed for 2 HBT            */
//#define   HBT_3L       0x14                    /* Lower limit allowed for 3 HBT            */
//#define   HBT_3H       0x1D                    /* Upper limit allowed for 3 HBT            */
//#define   HBT_4L       0x1E                    /* Lower limit allowed for 4 HBT            */
//#define   HBT_4H       0x24                    /* Upper limit allowed for 4 HBT            */
//#define   MAXCOUNT_H   0x25                    /* Timer ticks before TIMEOUT (high byte)   */
//#define   MAXCOUNT_L   0x00                    /* Timer ticks before TIMEOUT (low byte)    */

/******* LIN Buffer and counter *******/

unsigned char   LIN20_Buf[8] = {0,0,0,0};        /* Robot control bits to be put onto bus    */
unsigned char   CLEAR_LIN_DATA = 0x00;           /* Counter - controls resetting of LIN data */
                                                 /* to all zeros (to stop movement)          */
/******* IR receiver variables *******/

unsigned char   CLEARFLAG       = 0x00;          /* Used to read registers to clear flags    */
unsigned char   CHECKFLAG       = 0x00;          /* Used to check status of flags            */

unsigned char   IR_ERROR        = FALSE;         /* TRUE when error occurred during IR       */
                                                 /* receive routine                          */
unsigned char   IR_TIME         = 0x00;          /* Time since last rising edge              */
```

```
unsigned char   PREV_BIT        = ONE;              /* Value of the previous data bit           */
unsigned char   IR_START        = FALSE;            /* TRUE when looking for START falling edge */
unsigned char   RX_IN_PROGRESS  = FALSE;            /* TRUE when IR message is being received   */

unsigned int    IR_DATA         = 0x0000;           /* Holds IR data message as it is received  */
unsigned int    IR_DATA_MASK    = 0x4000;           /* Mask to show where each data bit should  */
                                                    /* be put in IR_DATA variable               */

unsigned int    MESSAGE         = 0x0000;           /* Buffer where latest IR message is put by */
                                                    /* receive routine. User program reads it   */
                                                    /* from here                                */
unsigned char   IR_NEW_DATA     = FALSE;            /* TRUE when new message is available in the*/
                                                    /* MESSAGE buffer                           */

/*****************************************************************************************
Function definitions
*****************************************************************************************/

void SPI_Transmit(unsigned int);                    /* Transmit data using the SPI              */
void TimerB_Init(void);                             /* Initialize Timer B                       */
void IC_Start_Edge(void);                           /* Set up Timer B CH0 Input Capture         */
                                                    /* to detect a falling edge                 */
void SPI_Init(void);                                /* Initialize the SPI                       */
void TBM_Init(void);                                /* Initialize the TBM                       */

/*****************************************************************************************
Function Name : Main
Engineer :      G. Brown
Date :          February 2003
Parameters :    none
Returns :       none
Notes :
*****************************************************************************************/

void main(void)
{
    /******* Local variables *******/

    unsigned int    LEDVALUE        = 0x0000;       /* Value to send to LEDs by SPI             */

    /******* Set up the CONFIGURATION registers *******/

    CONFIG1 = 0x01;                                 /* Disable COP                              */
    CONFIG2 = 0x2D;                                 /* External oscillator, timebase prescaler  */
                                                    /* enabled, SPI SS pullup disabled          */

    /******* Set up the registers for port pins that are configured as outputs *******/

    PTA     = 0x50;                                 /* x101 0xxx..turn off LEDs on pins A4 & A6 */
    PTB     = 0x20;                                 /* xx1x xxxx..MC33399 enable high           */
    PTC     = 0x00;                                 /* xxxx x00x..all outputs off               */
    PTD     = 0x03;                                 /* xxxx xx11..Turn LEDs off on pins D0 & D1 */
    PTE     = 0x00;                                 /* xxxx xxxx..No pins configured as outputs */

    /******* Set up the Data Direction for each port pin *******/

    DDRA    = 0x78;                                 /* 0111 1000 see 'port usage' above         */
    DDRB    = 0x20;                                 /* 0010 0000 see 'port usage' above         */
    DDRC    = 0x06;                                 /* 0000 0110 see 'port usage' above         */
```

```
DDRD    = 0x03;                                 /* 0000 0011 see 'port usage' above         */
DDRE    = 0x00;                                 /* 0000 0000 see 'port usage' above         */

/******* Set up the Clock Source *******/

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

/******* Initialize modules *******/

SPI_Init();                                     /* Initialize the SPI                       */
TimerB_Init();                                  /* Initialize Timer B                       */

/******* Clear Interrupt mask *******/

asm CLI;                                        /* Enable interrupts                        */

/******* Initialize LIN bus drivers *******/

LIN_Init();                                     /* Initialize LIN                           */

LIN_PutMsg (0x20, LIN20_Buf);                   /* Send the LIN20_Buf buffer contents       */
                                                /* (initialized to all zeros at beginning   */
                                                /* of program) to the LIN driver buffer     */

/******* Turn off all diagnostic LEDs connected to SPI shift register circuit *******/

LEDVALUE = 0x0000;                              /* Set LED data variable to 0000            */
SPI_Transmit(LEDVALUE);                         /* Send 0000 to SPI to turn off all LEDs    */

    /******* Set up timebase module *******/

TBM_Init();

/******* Prepare variables to be used in main program and output LEDs *******/

PREV_BIT        = ONE;                          /* Default starting condition               */
IR_ERROR        = FALSE;                        /* Reset ERROR flag                         */
IR_START        = FALSE;                        /* Not yet looking for START edge           */
IR_DATA_MASK    = 0x0000;                       /* Clear mask register                      */
IR_DATA         = 0x0000;                       /* Clear data register                      */
RX_IN_PROGRESS  = FALSE;                        /* No receive in process yet                */
IR_NEW_DATA     = FALSE;                        /* No new data yet                          */

PORTD           = 0x00;                         /* READY LED on, MOVING LED on              */
PORTA           &= 0xEF;                        /* MANUAL LED on - in manual mode initially */

/******* Main Program Loop *******/

IC_Start_Edge();                                /* Set Input Capture to look for start edge */

while (1)
{
    if (TBCR & 0x80)
    {
```

```
/******* If timebase module roll-over has occurred *******/

TBCR |= 0x08;                           /* Clear timebase flag                    */

CLEAR_LIN_DATA ++;                      /* Increment counter to show we've received */
                                        /* another tick                           */

if (CLEAR_LIN_DATA == 0x40)             /* Every 64th roll-over (approx 130ms with  */
                                        /* TBR = 011)                             */
{
    CLEAR_LIN_DATA = 0x00;              /* Reset the counter to 0 again           */

    if(IR_NEW_DATA == FALSE)            /* If no new data received in this time     */
                                        /* (no button pressed on controller).       */
    {
        LIN20_Buf[0] = 0x00;            /* Clear this LIN buffer byte             */
        LIN20_Buf[1] = 0x00;            /* Clear this LIN buffer byte             */
        LIN20_Buf[2] = 0x00;            /* Clear this LIN buffer byte             */
        LIN20_Buf[3] &= 0x80;           /* Clear byte (except master select bit)  */

        LIN_PutMsg (0x20, LIN20_Buf);   /* Update LIN buffer with above values    */

        PORTD &= 0xFD;                  /* Moving LED on                          */

        LEDVALUE = 0x0000;              /* Turn off all SPI LEDs                  */
        SPI_Transmit(LEDVALUE);         /* Send new LED value to 16-LED array     */
    }
}

if (IR_NEW_DATA == TRUE)
{
    /******* If new data has arrived *******/

    CLEAR_LIN_DATA = 0x00;              /* clear counter to start                 */

    /* The value received by the MCU input pin is the COMPLIMENT of the value     */
    /* sent by the transmitter due to the inversion by the sensor.                */
    /* Therefore, a low-to-high transition (e.g. the start bit) sent by the       */
    /* transmitter will be received at the MCU port pin as a high-to-low transition */
    /* The start edge is therefore a falling edge as opposed to the rising edge   */
    /* given in the specification.                                                */
    /* The rules used to decode the IR data take this inversion into account      */

    LEDVALUE = MESSAGE;                 /* Send MESSAGE value to LEDs on SPI port */
    SPI_Transmit(LEDVALUE);             /* to display the RC-5 code received      */

    MESSAGE &= 0x37FF;                  /* 0011 0111 1111 1111                    */
                                        /* Make toggle bit always = 0 and make 2  */
                                        /* unused bits always = 0                 */

    LIN20_Buf[0] = 0x00;                /* Clear buffer location 0                */
    LIN20_Buf[1] = 0x00;                /* Clear buffer location 1                */
    LIN20_Buf[2] = 0x00;                /* Clear buffer location 2                */
    LIN20_Buf[3] &= 0x80;               /* Clear location 3 (but leave bit 7 alone) */

    /******* Find out if message received is a robot control code *******/

    switch (MESSAGE)
    {
```

```
/******* MANUAL or LIN MASTER control *******/

case 0x3020:                    /* CHANNEL +            MANUAL CONTROL      */
    LIN20_Buf[3] = 0x00;        /* Buffer[3] = 0000 0000                    */
    PORTA &= 0xEF;              /* Manual LED on                            */
    PORTD &= 0xFD;              /* Moving LED on                            */
    break;                      /* IR message = XX 11X 00000 100000         */

case 0x3160:                    /* VCR_CHANNEL +        MANUAL CONTROL      */
    LIN20_Buf[3] = 0x00;        /* Buffer[3] = 0000 0000                    */
    PORTA &= 0xEF;              /* Manual LED on                            */
    PORTD &= 0xFD;              /* Moving LED on                            */
    break;                      /* IR message = XX 11X 00101 100000         */

case 0x3021:                    /* CHANNEL -            LIN MASTER          */
    LIN20_Buf[3] = 0x80;        /* Buffer[3] = 1000 0000                    */
    PORTA |= 0x10;              /* Manual LED off                           */
    PORTD &= 0xFD;              /* Moving LED on                            */
    break;                      /* IR message = XX 11X 00000 100001         */

case 0x3161:                    /* VCR_CHANNEL -        LIN MASTER          */
    LIN20_Buf[3] = 0x80;        /* Buffer[3] = 1000 0000                    */
    PORTA |= 0x10;              /* Manual LED off                           */
    PORTD &= 0xFD;              /* Moving LED on                            */
    break;                      /* IR message = XX 11X 00101 100001         */

/******* ROTATION control *******/

case 0x3011:                    /* VOLUME -             ROTATE LEFT SLOW    */
    LIN20_Buf[0] = 0x08;        /* Buffer[0] = 0000 1000                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00000 010001         */

case 0x3151:                    /* VCR_VOLUME -         ROTATE LEFT FAST    */
    LIN20_Buf[0] = 0x02;        /* Buffer[0] = 0000 0010                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00101 010001         */

case 0x3010:                    /* VOLUME +             ROTATE RIGHT SLOW   */
    LIN20_Buf[0] = 0x04;        /* Buffer[0] = 0000 0100                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00000 010000         */

case 0x3150:                    /* VCR_VOLUME +         ROTATE RIGHT FAST   */
    LIN20_Buf[0] = 0x01;        /* Buffer[0] = 0000 0001                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00101 010000         */

/******* ARM control *******/

case 0x3014:                    /* COLOR +              ARM UP SLOW         */
    LIN20_Buf[0] = 0x40;        /* Buffer[0] = 0100 0000                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00000 010100         */

case 0x3154:                    /* VCR_COLOR +          ARM UP FAST         */
    LIN20_Buf[0] = 0x10;        /* Buffer[0] = 0001 0000                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00101 010100         */
```

```
case 0x3015:                    /* COLOR -              ARM DOWN SLOW       */
    LIN20_Buf[0] = 0x80;        /* Buffer[0] = 1000 0000                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00000 010101         */

case 0x3155:                    /* VCR_COLOR -          ARM DOWN FAST       */
    LIN20_Buf[0] = 0x20;        /* Buffer[0] = 0010 0000                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00101 010101         */

/******* ELBOW control *******/

case 0x3012:                    /* BRIGHTNESS +         ELBOW UP SLOW       */
    LIN20_Buf[1] = 0x04;        /* Buffer[1] = 0000 0100                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00000 010010         */

case 0x3152:                    /* VCR_BRIGHTNESS +     ELBOW UP FAST       */
    LIN20_Buf[1] = 0x01;        /* Buffer[1] = 0000 0001                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00101 010010         */

case 0x3013:                    /* BRIGHTNESS -         ELBOW DOWN SLOW     */
    LIN20_Buf[1] = 0x08;        /* Buffer[1] = 0000 1000                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00000 010011         */

case 0x3153:                    /* VCR_BRIGHTNESS -     ELBOW DOWN FAST     */
    LIN20_Buf[1] = 0x02;        /* Buffer[1] = 0000 0010                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00101 010011         */

/******* WRIST control *******/

case 0x3018:                    /* MASTER TREBLE +      WRIST UP SLOW       */
    LIN20_Buf[1] = 0x40;        /* Buffer[1] = 0100 0000                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00000 011000         */

case 0x3158:                    /* VCR_MASTER TREBLE +  WRIST UP FAST       */
    LIN20_Buf[1] = 0x10;        /* Buffer[1] = 0001 0000                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00101 011000         */

case 0x3019:                    /* MASTER TREBLE -      WRIST DOWN SLOW     */
    LIN20_Buf[1] = 0x80;        /* Buffer[1] = 1000 0000                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00000 011001         */

case 0x3159:                    /* VCR_MASTER TREBLE -  WRIST DOWN FAST     */
    LIN20_Buf[1] = 0x20;        /* Buffer[1] = 0010 0000                    */
    PORTD |= 0x02;              /* Moving LED off                           */
    break;                      /* IR message = XX 11X 00101 011001         */
```

```
                    /******* GRABBER control *******/

                    case 0x3016:                    /* MASTER BASS +        GRABBER OPEN        */
                        LIN20_Buf[2] = 0x02;        /* Buffer[2] = 0000 0010                    */
                        PORTD |= 0x02;              /* Moving LED off                           */
                        break;                      /* IR message = XX 11X 00000 010110         */

                    case 0x3156:                    /* VCR_MASTER BASS +    GRABBER OPEN        */
                        LIN20_Buf[2] = 0x02;        /* Buffer[2] = 0000 0010                    */
                        PORTD |= 0x02;              /* Moving LED off                           */
                        break;                      /* IR message = XX 11X 00101 010110         */

                    case 0x3017:                    /* MASTER BASS -        GRABBER CLOSED      */
                        LIN20_Buf[2] = 0x01;        /* Buffer[2] = 0000 0001                    */
                        PORTD |= 0x02;              /* Moving LED off                           */
                        break;                      /* IR message = XX 11X 00000 010111         */

                    case 0x3157:                    /* VCR_MASTER BASS -    GRABBER CLOSED      */
                        LIN20_Buf[2] = 0x01;        /* Buffer[2] = 0000 0001                    */
                        PORTD |= 0x02;              /* Moving LED off                           */
                        break;                      /* IR message = XX 11X 00101 010111         */

                    default:
                                                    /* If message was none of above, ignore it  */
                        break;
                }   /* switch(IR_DATA) */

                LIN_PutMsg (0x20, LIN20_Buf);       /* Update LIN buffer                        */
                IR_NEW_DATA = FALSE;                /* Reset New_data flag                      */

            }   /* if(IR_NEW_DATA == TRUE) */
        }   /* if(TBCR & 0x80) */
    }   /* while(1) */
}   /* main */

/*****************************************************************************************
Function Name : TimerB_Init
Engineer :      G. Brown
Date :          February 2003
Parameters :    none
Returns :       none
Notes :          Initialize Timer B
*****************************************************************************************/

void TimerB_Init(void)
{
    TBSC    = 0x20;                                 /* stop timer B                             */
    TBSC    = 0x10;                                 /* reset timer B                            */

    TBMODH  = MAXCOUNT_H;                           /* TimerB MODULO value (high byte)          */
    TBMODL  = MAXCOUNT_L;                           /* TimerB MODULO value (low byte)           */

    TBCH0H  = 0x00;                                 /* TimerB CHANNEL0 register (high byte)     */
    TBCH0L  = 0x00;                                 /* TimerB CHANNEL0 register (low byte)      */

    TBSC0   = 0x00;                                 /* TimerB Status/Control register channel 0 */
                                                    /* Leave Timer B stopped                    */
}
```

```
/*****************************************************************************************
Function Name : TBM_Init
Engineer :      G. Brown
Date :          February 2003
Parameters :    none
Returns :       none
Notes :          Initialise TimeBase Module
*****************************************************************************************/

void TBM_Init(void)
{
    TBCR = 0x00;                                    /* Clear TBON to turn TBM off              */
    TBCR = 0x30;                                    /* Prescaler = divide-by-16384...roll-over */
                                                    /* every 2.05ms... TBMCLKSEL in CONFIG2 is */
                                                    /* set to 1                                */
    TBCR |= 0x02;                                   /* Set the TBON bit to start TBM           */
}

/*****************************************************************************************
Function Name : SPI_Init
Engineer :      G. Brown
Date :          February 2003
Parameters :    none
Returns :       none
Notes :         Initialise SPI
*****************************************************************************************/

void SPI_Init(void)
{
    /* The SPI drives 2 MC74HC164AN shift registers which are connected together to form a      */
    /*  single 16-bit shift register. Each output has the cathode of an LED connected to it,    */
    /*  with the anode of each LED connected to +5V through a resistor.                         */
    /* The SPI is set up to output a clock signal which is low when idle, and gives a rising    */
    /*  edge in the centre of each data bit as required to clock data into the shift register.  */
    /*  The CPOL and CPHA bits are both 0 to select this mode (see the diagrams in the SPI      */
    /*  section of the EY16 databook for more information). The data is output from the MOSI    */
    /*  (Master_Out_Slave_In) pin. The Master_In_Slave_Out (MISO) and SS pins are not used in   */
    /*  this application.                                                                        */
    /* The baud rate divisor (BD) is set to 00 to give the highest data rate to minimise the    */
    /*  time taken to send the data                                                             */
    /*      BAUD RATE = CGMOUT/(2*BD)                                                           */
    /*      with 8MHz XTAL, BAUD RATE = 2000000/(2*2) = 2000000/4 = 500000 bits per second      */
    /* The baud rate could have been set to a lower value using the prescaler, but this data    */
    /*  rate is still well within the maximum limitations quoted in the shift register datasheet*/

    SPCR    = 0x20;                                 /* 0010 0000 - master mode, SPI disabled    */
    SPSCR   = 0x00;                                 /* 0000 0000 - baud rate divisor /2         */
    SPCR    = 0x22;                                 /* 0010 0010 - enable module, master mode   */
}
```

```
/*****************************************************************************************
Function Name : IC_Start_Edge
Engineer :      G. Brown
Date :          February 2003
Parameters :    none
Returns :       none
Notes :         Set timer B CH0 up to catch FALLING edge (called once when program starts)
*****************************************************************************************/

void IC_Start_Edge(void)
{
    /******* Set up Input Capture for FALLING edge *******/

    PREV_BIT        = ONE;                          /* Prev bit was a 1 (starting condition)    */
    IR_ERROR        = FALSE;                        /* Reset error flag                         */
    IR_START        = TRUE;                         /* Looking for start bit                    */
    IR_DATA_MASK    = 0x4000;                       /* 0100 0000 0000 0000 - put first bit into */
                                                    /* start location                           */
    IR_DATA         = 0x0000;                       /* Clear data register                      */
    RX_IN_PROGRESS  = FALSE;                        /* No receive in progress yet               */
    IR_NEW_DATA     = FALSE;                        /* No new data yet                          */

    TBSC            = 0x20;                         /* Stop timer B                             */
    TBSC            = 0x30;                         /* Reset timer B                            */

    TBCH0H          = 0x00;                         /* Clear TimerB CH0 register (high byte)    */
    TBCH0L          = 0x00;                         /* Clear TimerB CH0 register (low byte)     */

    /******* Clear channel 0 flag, enable channel 0 interupts & set up for FALLING edge *******/

    CLEARFLAG       = TBSC0;                        /* Read TBSC0, allow CH0 flag to be cleared */
    TBSC0           = 0x08;                         /* 0 into INT flag, select FALLING edge     */
    TBSC0           = 0x48;                         /* Enable channel 0 interrupt               */

    /******* Clear overflow flag *******/

    CLEARFLAG       = TBSC;                         /* Read register - allow flag to be cleared */
    TBSC            = 0x20;                         /* Write 0 to clear Int flag, Timer stopped */
    TBSC            = 0x00;                         /* Start timer B, overflow Int disabled     */
}

/*****************************************************************************************
Function Name : TimerB_Input_Capture
Engineer :      G. Brown
Date :          February 2003
Parameters :    none
Returns :       none
Notes :         Input Capture Timer B channel 0 ISR
*****************************************************************************************/

#pragma TRAP_PROC
void TimerB_Input_Capture()
{
    /******* Stop the timer, Clear channel 0 flag *******/

    TBSC        = 0x30;                             /* Stop and reset timer B                   */

    CLEARFLAG   = TBSC;                             /* Read register - allow flag to be cleared */
    TBSC        = 0x30;                             /* ensure OVF flag cleared                  */
```

```
                                                /* Write 0 to clear OVF flag, Timer stopped */
CLEARFLAG    = TBSC0;                           /* Read TBSC0, allow CH0 flag to be cleared */
TBSC0        = 0x04;                            /* 0 into INT flag, select RISING edge      */

if (IR_START == TRUE)
{
    /******* If the START bit was being received (falling edge capture) *******/

    IR_START         = FALSE;                   /* No longer looking for the start bit      */
    RX_IN_PROGRESS   = TRUE;                    /* Receive is now in progress               */
    IR_DATA_MASK     = IR_DATA_MASK >> 1;       /* Shift mask to next position              */
    PORTD            |= 0x01;                   /* READY LED off                            */
}
else
{
    /******* If a DATA bit was being received (rising edge capture) *******/

    /******* Get the time from the timer B high register (low byte is not used *******/
    /******* since only 8-bit resolution is required for this application)     *******/

    IR_TIME = TBCH0H;                           /* High byte into low byte of IR_TIME       */
    CHECKFLAG = TBCH0L;                         /* Read lower byte of TimerB CH0 to unlock  */
                                                /* registers & allow further Input Captures */
    if(IR_DATA_MASK == 0x2000)
    {
        IR_TIME = IR_TIME + HBT_1;              /* If first edge after start edge, add 1 HBT*/
                                                /* value onto time measured to allow use of */
                                                /* standard bit-analysis routines           */
    }

    /******* If after 2 HBT and last bit was 1 OR If after 3 HBT and last bit was 0 *******/
    /*******    ...then data is a 1                                                  *******/

    if( ( (IR_TIME >= HBT_2L) && (IR_TIME <= HBT_2H) && (PREV_BIT == ONE) )
    ||  ( (IR_TIME >= HBT_3L) && (IR_TIME <= HBT_3H) && (PREV_BIT == ZERO) ) )

    {
        IR_DATA = IR_DATA | IR_DATA_MASK;       /* Put a 1 into this bit of data register   */
        PREV_BIT = ONE;                         /* Last bit (i.e. the data just received    */
                                                /* here) was a 1                            */
        IR_DATA_MASK = IR_DATA_MASK >> 1;       /* Shift mask to point to next data bit     */
    }

    /******* If after 2 HBT and last bit was 0                                      *******/
    /*******    ...then data is a 0                                                 *******/

    else if ( (IR_TIME >= HBT_2L) && (IR_TIME <= HBT_2H) && (PREV_BIT == ZERO) )

    {
        PREV_BIT = ZERO;                        /* Last bit (i.e. the data just received    */
                                                /* here) was a 0                            */
                                                /* Do NOT set this data bit since data is 0 */
        IR_DATA_MASK = IR_DATA_MASK >> 1;       /* Shift mask to point to next data bit     */
    }
```

```
    /******* If after 3 HBT and last bit was 1 OR If after 4 HBT and last bit was 0 *******/
    /*******    ...then data is a 1 then a 0 (2 bits received)                      *******/

    else if( ( (IR_TIME >= HBT_3L) && (IR_TIME <= HBT_3H) && (PREV_BIT == ONE) )
    ||       ( (IR_TIME >= HBT_4L) && (IR_TIME <= HBT_4H) && (PREV_BIT == ZERO ) ) )

    {
        IR_DATA = IR_DATA | IR_DATA_MASK;       /* Put a 1 into this bit of data register   */
        IR_DATA_MASK = IR_DATA_MASK >> 1;       /* Shift mask to point to next data bit     */

                                                /* Do NOT set this data bit since data is 0 */
        PREV_BIT = ZERO;                        /* Last bit (i.e. the last data bit received*/
                                                /* here) was a 0                            */
        IR_DATA_MASK = IR_DATA_MASK >> 1;       /* Shift mask to point to next data bit     */
    }

    /******* If none of the above apply *******/

    else
    {
         IR_ERROR = TRUE;                       /* An error has occurred as the combination */
                                                /* of time since last edge and previous bit */
                                                /* value does not meet any of the 5 rules   */
    }
}

if((IR_DATA_MASK != 0x0000) && (IR_ERROR == FALSE))
{
    /******* In middle of a message and no errors have occurred, so set up timer    *******/
    /******* again to look for next rising edge                                     *******/
    /******* Start timer with CH0 (input capture) and Overflow interrupts enabled   *******/

    TBCH0H          = 0x00;                     /* Clear TimerB CH0 register (high byte)    */
    TBCH0L          = 0x00;                     /* Clear TimerB CH0 register (low byte)     */

    TBSC0           = 0x44;                     /* Enable channel 0 interrupt               */
    TBSC            = 0x40;                     /* Start timer B, OVF interrupts enabled    */
}
else
{
    /******* If this was the last bit of the current message or if in the middle     *******/
    /*******  of a message but an error has occurred, set up to look for the start   *******/
    /*******  of the next message                                                    *******/
    /*******  Look for FALLING edge and enable CH0 interrupt for input capture       *******/
    /*******  Do not enable overflow interrupts, since not yet looking for data bits *******/
    /*******  in the middle of a message                                             *******/
    /******* If this message was received with no errors AND if the user program     *******/
    /*******  has read the previous message (user program has put IR_NEW_DATA back   *******/
    /*******  to FALSE, so it's OK to overwrite it), then copy the new message into  *******/
    /*******  the user buffer (MESSAGE variable) and set the IR_NEW_DATA flag to     *******/
    /*******  TRUE. This indicates to the main (user) program that new data is ready *******/

    if((IR_ERROR == FALSE) && (IR_NEW_DATA == FALSE))
    {
        IR_NEW_DATA     = TRUE;                 /* New data is ready                        */
        MESSAGE         = IR_DATA;              /* copy new data into user buffer           */
    }
```

```
        PORTD           &= 0xFE;                    /* Finished receiving this message so       */
                                                    /* READY LED back on (bit D0 low)           */

        PREV_BIT        = ONE;                      /* Prev bit was a 1 (starting condition)    */
        IR_ERROR        = FALSE;                    /* Reset error flag                         */
        IR_START        = TRUE;                     /* Looking for start bit                    */
        IR_DATA_MASK    = 0x4000;                   /* 0010 0000 0000 0000 - 1 in mask set to   */
                                                    /* position of start bit                    */
        IR_DATA         = 0x0000;                   /* Clear data register                      */
        RX_IN_PROGRESS  = FALSE;                    /* No receive in progress yet               */

        TBCH0H          = 0x00;                     /* Clear TimerB CH0 register (high byte)    */
        TBCH0L          = 0x00;                     /* Clear TimerB CH0 register (low byte)     */

        /******* Enable channel 0 interupts & set up for FALLING edge *******/

        TBSC0           = 0x08;                     /* 0 into INT flag, select FALLING edge     */
        TBSC0           = 0x48;                     /* Enable channel 0 interrupt               */

        /******* Start TimerB with overflow interrupts disabled *******/

        TBSC            = 0x20;                     /* Write 0 to clear Int flag, Timer stopped */
        TBSC            = 0x00;                     /* Start timer B, overflow Int disabled     */
    }
}

/*****************************************************************************************
Function Name : TimerB_Overflow
Engineer :      G. Brown
Date :          February 2003
Parameters :    none
Returns :       none
Notes :         Timer B overflow has occurred -> timeout
*****************************************************************************************/

#pragma TRAP_PROC
void TimerB_Overflow()
{
    /* An overflow of the timer has occurred with no edge being encountered before this time.   */
    /* The counter value at which this occurs is set in the Timer B modulo registers (defined   */
    /* as MAXCOUNT_H and MAXCOUNT_L at the top of this program) before Timer B is set up.       */

    /******* STOP timer B (also disables IC) *******/

    TBSC            = 0x30;                     /* Stop, Reset timer B, (also disables      */
                                                /* Input Capture)                           */

    /******* Clear overflow flag, disable overflow interupts, timer remains stopped *******/

    CLEARFLAG       = TBSC;                     /* Read register, allow flag to be cleared  */
    TBSC            = 0x30;                     /* Clear ovf flag                           */

    CLEARFLAG       = TBSC0;                    /* Read TBSC0, allow CH0 flag to be cleared */
    TBSC0           = 0x04;                     /* 0 into INT flag, select RISING edge      */
                                                /* Input Capture                            */

    PORTD           &= 0xFE;                    /* READY LED back on - no longer receiving, */
```

```
    /******* Set up to detect a new start edge *******/

    PREV_BIT        = ONE;                          /* Prev bit was a 1 (starting condition)    */
    IR_ERROR        = FALSE;                        /* Reset error flag                         */
    IR_START        = TRUE;                         /* Looking for start bit                    */
    IR_DATA_MASK    = 0x4000;                       /* 0100 0000 0000 0000 - put first bit into */
                                                    /* start location                           */
    IR_DATA         = 0x0000;                       /* Clear data register                      */
    RX_IN_PROGRESS  = FALSE;                        /* No receive in progress yet               */

    TBCH0H          = 0x00;                         /* Clear TimerB CH0 register (high byte)    */
    TBCH0L          = 0x00;                         /* Clear TimerB CH0 register (low byte)     */

    /******* Enable channel 0 interupts & set up for FALLING edge *******/

    CLEARFLAG       = TBSC0;                        /* Read TBSC0, allow CH0 flag to be cleared */
    TBSC0           = 0x08;                         /* 0 into INT flag, select FALLING edge     */
    TBSC0           = 0x48;                         /* Enable channel 0 interrupt               */

    /******* Start TimerB with overflow interrupts disabled *******/

    CLEARFLAG       = TBSC;                         /* Read register - allow flag to be cleared */
    TBSC            = 0x20;                         /* Write 0 to clear Int flag, Timer stopped */
    TBSC            = 0x00;                         /* Start timer B, overflow Int disabled     */
}

/*****************************************************************************************
Function Name : SPI_Transmit
Engineer :      G. Brown
Date :          February 2003
Parameters :    LEDVALUE              value to put onto LEDs
Returns :       none
Notes :         Transmits 2 bytes of data using the SPI to display RC-5 data bits on LEDs
*****************************************************************************************/

void SPI_Transmit(unsigned int LEDVALUE)
{
/******* Local variables *******/

unsigned int LEDOUT16 = 0;                          /* Function puts copy of LEDVALUE in here   */
unsigned char LEDOUT_H = 0;                         /* Variable containing upper 8 bits of LED  */
                                                    /* data                                     */
unsigned char LEDOUT_L = 0;                         /* Variable containing lower 8 bits of LED  */
                                                    /* data                                     */

    /******* LEDs are on when low so we need to invert the data value so that a 1 = on *******/

    LEDVALUE = ~(LEDVALUE);                         /* Invert the LED value                     */

    /******* Prepare the 2 bytes of data to be sent to the LEDs *******/

    LEDOUT16 = LEDVALUE;                            /* Take copy of LEDVALUE                    */
    LEDOUT16 &= 0x00FF;                             /* Use mask to clear upper 8 bits           */
    LEDOUT_L = 0x00;                                /* Initialize LEDOUT low byte to 0          */
    LEDOUT_L = LEDOUT_L | LEDOUT16;                 /* Lower 8 bits of LEDOUT16 -> LEDOUT_L     */

    LEDOUT16 = LEDVALUE;                            /* Take copy of LEDVALUE                    */
    LEDOUT16 = LEDOUT16 >> 8;                       /* Shift value right by 8 places            */
    LEDOUT16 &= 0x00FF;                             /* Use mask to clear upper 8 bits           */
```

```
    LEDOUT_H = 0X00;                                /* Initialize LEDOUT high byte to 0         */
    LEDOUT_H = LEDOUT_H | LEDOUT16;                 /* Lower 8 bits of LEDOUT16 -> LEDOUT_H     */

    /******* Wait until Tx buffer available (empty) and send the HIGH byte to the   *******/
    /******* shift register                                                         *******/

    CHECKFLAG = 0x00;                               /* Clear variable                           */

    while (CHECKFLAG != 0x08)
    {
        CHECKFLAG = SPSCR;                          /* Wait until SPTE flag (SPSCR bit 3) set   */
        CHECKFLAG &= 0x08;                          /* Indicates data moved to SPI shift reg.   */
                                                    /* Tx buffer now empty - can take more data */
    }

    SPDR = LEDOUT_H;                                /* Upper LED byte -> SPI data register      */
                                                    /* clocked out MSB first                    */

    /******* Wait until Tx buffer available (empty) and send the LOW byte to the    *******/
    /******* shift register                                                         *******/

    CHECKFLAG = 0x00;                               /* Clear variable                           */
    while (CHECKFLAG != 0x08)
    {
        CHECKFLAG = SPSCR;                          /* Wait until SPTE flag (SPSCR bit 3) set   */
        CHECKFLAG &= 0x08;                          /* Indicates data moved to SPI shift reg.   */
                                                    /* Tx buffer now empty - can take more data */
    }

    SPDR = LEDOUT_L;                                /* Lower LED byte -> SPI data register      */
                                                    /* clocked out MSB first                    */

    /******* Clear the SPRF flag before returning *******/

    CLEARFLAG = SPSCR;                              /* read SPSCR....                           */
    CLEARFLAG = SPDR;                               /* and then SPDR to clear SPRF flag         */
}

/*****************************************************************************************
Function Name : LIN_Command
Engineer :
Date :          February 2003
Parameters :    none
Returns :       none
Notes :         User call-back. Called by the driver after transmission or reception
                of the Master Request Command Frame (ID: 0x3C)
*****************************************************************************************/

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

Appendix B – Vector.c

```
#define VECTOR_C
/*****************************************************************************
*
*       Copyright (C) 2001 Motorola, Inc.
*
* Functions:    Vectors table for LIN08 Drivers with Motorola API
*
* 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();

extern void TimerB_Overflow();              /* Timer Module B Overflow ISR  */
extern void TimerB_Input_Capture();         /* Timer Module B Channel 0 ISR */

// extern void TimerB0();
// extern void BREAK_Command();             /* SWI 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 2L31N mask set of the          */
/*      MC68HC908EY16 and all subsequent versions.                           */
/*                                                                           */
/*      Older mask sets, e.g. 0L38H, 1L38H, 0L31N and 1L31N had a fault      */
/*      in their interrupt vector table and hence in the priorities.         */
/*      For these older mask sets the order of the SCI vectors was:          */
/*                                                                           */
/*      SCI_Error_ISR,                      // 0xFFE6   ESCI error           */
/*      SCI_Transmit_ISR,                   // 0xFFE8   ESCI transmit        */
/*      SCI_Receive_ISR,                    // 0xFFEA   ESCI receive         */
/*                                                                           */
/*      All other vectors are unchanged.                                     */
/*                                                                           */
/*****************************************************************************/

{
    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             */

#if defined(MASTER)                         /* (used for Master node only)   */
    LIN_VECTF LIN_ISR_SCI_Transmit,         /* 0xFFE6   ESCI transmit        */
#endif /* defined(MASTER) */

#if defined(SLAVE)
    LIN_VECTF NULL,                         /* 0xFFE6   ESCI transmit        */
#endif /* defined(SLAVE) */

    LIN_VECTF LIN_ISR_SCI_Receive,          /* 0xFFE8   ESCI receive         */
    LIN_VECTF LIN_ISR_SCI_Error,            /* 0xFFEA   ESCI error           */

    LIN_VECTF TimerB_Overflow,              /* 0xFFEC   TIMER B overflow     */
    LIN_VECTF NULL,                         /* 0xFFEE   TIMER B channel 1    */
    LIN_VECTF TimerB_Input_Capture,         /* 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) */
```

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

freescale™
semiconductor