Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

PSoC CY8C20x34 TRM

PSoC® CY8C29x66, CY8C27x43, CY8C27x43E, CY8C21x34

LIN Bus 2.0

PSoC Reference Design. Revision **

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone (USA): 800.858.1810
Phone (Intnl.): 408.943.2600

http://www.cypress.com

Copyrights

Disclaimer

Flash Code Protection

Note the following details of the Flash code protection features on Cypress devices.

Cypress products meet the specifications contained in their particular Cypress Data Sheets. Cypress believes that its family of products is one of the most secure families of its kind on the market today, regardless of how they are used. There may be methods, unknown to Cypress, that can breach the code protection features. Any of these methods, to our knowledge, would be dishonest and possibly illegal. Neither Cypress nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Cypress is willing to work with the customer who is concerned about the integrity of their code. Code protection is constantly evolving. We at Cypress are committed to continuously improving the code protection features of our products.

Contents

1. LIN Bus 2.0 Kit

1.1 LIN Bus 2.0 Demonstration Kit Description

1.1.1 Introduction

The LIN Bus Demonstration Kit demonstrates the ability of the PSoC® Programmable System-on-Chip™ to implement LIN bus, Local Interconnect Network, standard protocol. The LIN bus was developed to fill the need for a low cost automotive network to complement existing networks. LIN bus also finds many uses in non-automotive distributed systems where a robust, low-speed and low-cost protocol is required. Additional information is located on the LIN consortium web site at http://www.lin-subbus.org where you can also find the complete LIN specifications for version 2.0.

This design provides a flexible development environment for creation of either slave or master LIN device applications using the PSoC. The demonstration board has one master and two slave nodes. Using dynamic reconfiguration, hardware resources are minimized with low CPU overhead.

Design details on specific implementation provided with the demonstration board are included in the supplied Lin Master Node Design IP, Lin Slave Design IP, Application Note AN2045, and in the corresponding project comments inside PSoC Designer™.

1.2 Kit Contents

- LIN Bus Demonstration Board
- International Power Supply (110-220 VAC to 12V DC)
- Serial Cable (DB-9)
- Software CD with Documentation, Example Project, and Design IP

1.3 Getting Started

The LIN bus demonstration board is preprogrammed to demonstrate the LIN bus directly out of the box. To demonstrate functionality, follow these steps:

1. Verify contents in design kit.
2. Plug the power supply into a wall outlet (international plug adaptors are included). The power supply automatically adapts to this voltage and frequency range: 100-240 VAC at 50-60 Hz.
3. Connect the barrel plug of the power supply cord into the demonstration board. The green power LED next to the power jack lights.

The demonstration board is now fully operational and demonstrates LIN bus operations. Functional details of the examples running on the board can be found in section 1.4, LIN Bus Demonstration, on page 6.

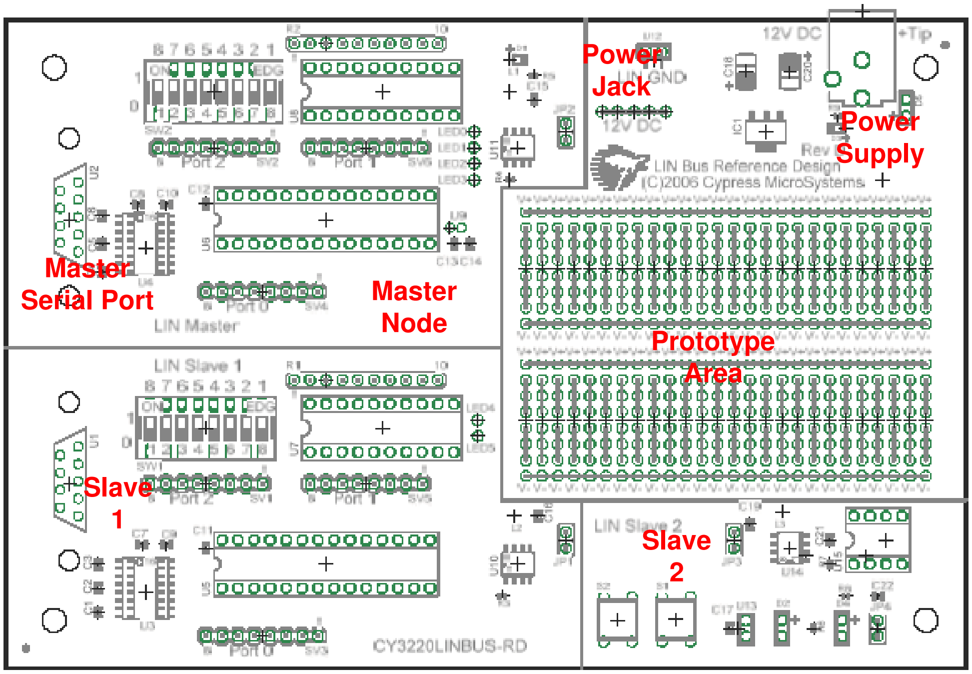

Figure 1-1. LIN Bus Demonstration Board

The master node and slave 1 are both implemented in a 28-pin part, CY8C27443-24PXI. Slave 2 is implemented in an 8-pin part, CY8C27143-24PXI.

The CD-ROM that is included with this kit has all project files for the designed-in devices as well as project files for automotive grade devices.

1.4 LIN Bus Demonstration

The LIN bus demonstration board is divided into four regions: master node, slave 1, slave 2, and the prototype area.

The master node has a bank of 8 dip-switches, SW2, and a bank of 10 LEDs, U8. Slave 1 also has a set of 8 dip-switches, SW1, and a bank of 10 LEDs, U7. Slave 2 has 2 push-button switches, S1 and S2, and 2 individual green LEDs, D2 and D4. Figure 1-1 shows the positions of these components.

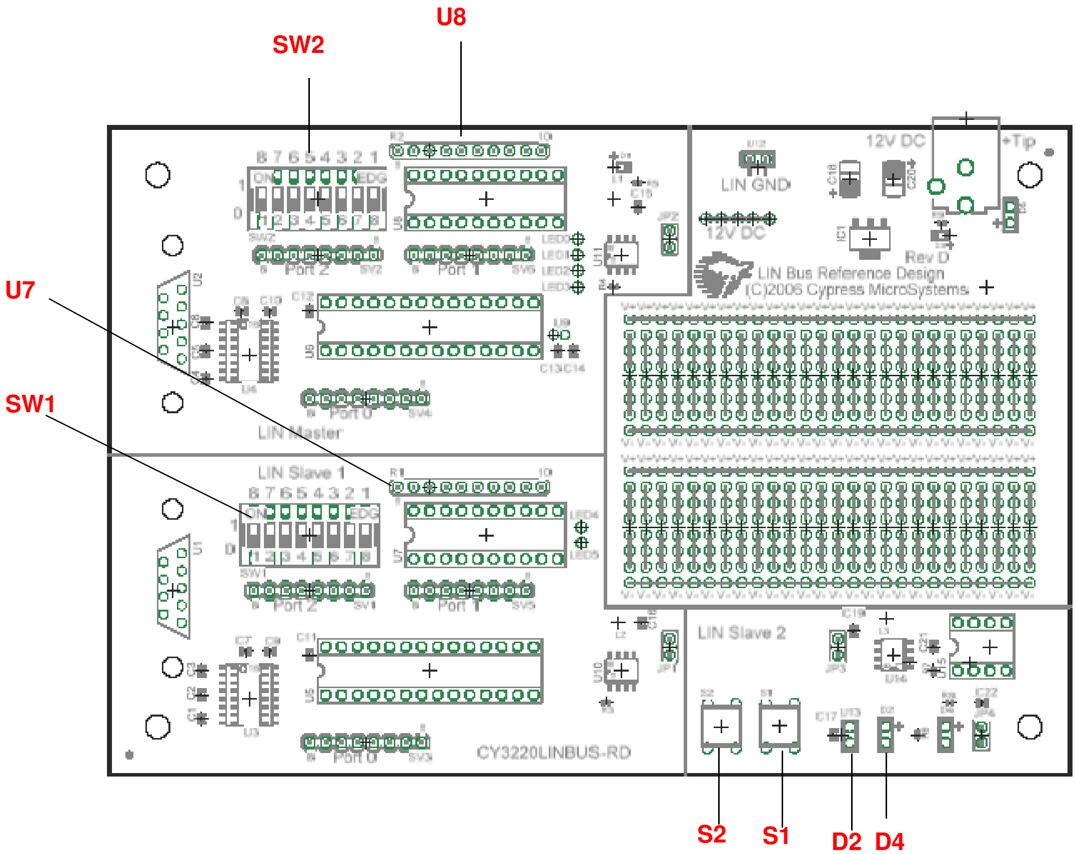

Figure 1-2. Layout of Node-Specific Switches and LEDs

Periodically, the master node sends its switch state information to slave 1 and then polls both slaves for their switch state information. In response, the master and slaves display the state of the information as specified by the switch-to-display relationship. Figure 1-3 and the following list show the switches and the LEDs that they control:

- Master node dip-switches 8 to 1 control slave 1's LEDs 1 to 8. Note that the dip-switch numbering is reversed but is oriented such that the left most switch, numbered 8, controls the left most LED of slave 1.
- Slave 1's dip-switches 8 to 5 control master node LEDs 1 to 4.
- Slave 1 measures the resistance connected between P0[1] and P0[3] and sends this information to the master. To make the resistance measurement, a reference resistance of 2.2K is connected between P0[1] and P0[2]. These resistance connections can be made to the header (SV3) meant for port 0 of slave 1.
- Slave 1's dip-switches 2 and 1 control slave 2 LEDs, D2 and D4. These switches are configured to implement a left / right turn indicator. When one of these switches is closed, D2 or D4 blinks.
- Slave 2's push-button switches, S2 and S1, control master node LEDs 5 and 6.
- The remaining switches and LEDs are not used, but board connections are provided for use in the prototype area.

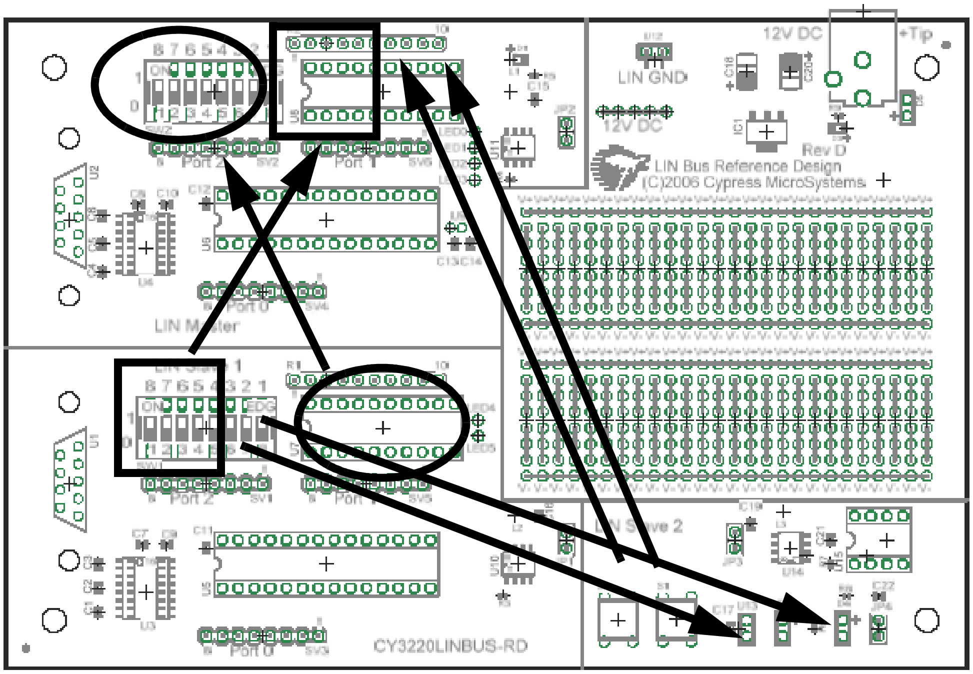

Figure 1-3. Switch-to-LED Control Relationship

The master node performs the following operations.

- Initializes the LIN communication.
- Calls the node configuration function to configure slave 1 and slave 2 nodes.
- Initializes the Schedule table. The frame sequence and time allotted for each frame is configured in the Schedule table.

Then the master node performs the following operations in an infinite loop:

- Checks if the current frame transfer is complete and if the LIN hardware is ready to send the next frame. If yes, calls the l_sch_tick function that loads the next frame due and initiates the transfer.
- Checks if Frame1 completion flag is set. Frame1 carries the master's dip-switch information. If Frame1 flag is set, the master updates the Frame1 buffer with the dip-switch information and sends the dip-switch information over serial port.
- Checks if Frame2 completion flag is set. Frame2 carries the resistance information from slave 1. If Frame2 flag is set, converts the 2-byte HEX integer to an ASCII string and sends this string over the serial port.
- Checks if Frame3 completion flag is set. Frame3 carries the switch status of slave 2. If Frame3 flag is set, updates LED 5 and LED 6 according to the switch status and sends the Slave-2 switch status over serial port.
- Checks if Frame4 completion flag is set. Frame4 carries the dip-switch status of slave 1. If Frame4 flag is set, updates LED 1 to LED 4 as per status of SW8 to SW5 of slave 1. Then updates the Frame5 buffer with the status of SW1 and SW2 of slave 1. When Frame5 is due, this information is sent to slave 2 and slave 2 blinks D4 or D5, accordingly. Then it sends the Slave-1 switch status over serial port.
- The master's data over serial port may be observed by using a HyperTerminal and connecting the master's serial port to the PC. The serial port setting is 38.4 kbps, 8 data bits, no parity, 1 stop bit. The following is an example output on the serial port:

```
Master Switch Status: ON ON ON ON ON ON ON ON
Slave 2 Switch Status: ON OFF
Slave 1 Resistance: 25000
Slave 1 Switch Status: ON ON ON ON ON ON ON ON
```

1.5 Master Node Port Pin Usage

The pin usage for the LIN bus PSoC master node is as follows:

Table 1-1. Port 0 – Pins Connect to User-Accessible Header Row

0	Not used
1	Not used
2	Not used
3	Not used
4	Not used
5	LIN bus RX
4	LIN bus TX
6	UART RX
7	UART TX

Table 1-2. Port 1 – Pins Connect to User-Accessible Header Row and LEDs

0	Crystal out
1	Crystal in
2	LED controlled by slave 2, Port0_7 switch
3	LED controlled by slave 2, Port0_2 switch
4	LED controlled by slave 1, Port2_4 switch
5	LED controlled by slave 1, Port2_5 switch
6	LED controlled by slave 1, Port2_6 switch
7	LED controlled by slave 1, Port2_7 switch

Table 1-3. Port 2 – Pins Connect to User-Accessible Header Row and Dip-Switches

0	Switch controls slave 1, Port1_0 LED
1	Switch controls slave 1, Port1_1 LED
2	Switch controls slave 1, Port1_2 LED
3	Switch controls slave 1, Port1_3 LED
4	Switch controls slave 1, Port1_4 LED
5	Switch controls slave 1, Port1_5 LED
6	Switch controls slave 1, Port1_6 LED
7	Switch controls slave 1, Port1_7 LED

1.6 Slave 1 Port Pin Usage

The section details the pin usage for the LIN bus PSoC slave 1:

Table 1-4. Port 0 – Pins Connect to User-Accessible Header Row

0	Not used
1	Common point of measured resistance and reference resistance
2	Reference resistance
3	Measured resistance
4	LIN bus TX
5	LIN bus RX
6	Not used
7	Not used

Table 1-5. Port 1 – Pins Connect to User-Accessible Header Row and LEDs

0	LED controlled by master, Port2_0 switch
1	LED controlled by master, Port2_1 switch
2	LED controlled by master, Port2_2 switch
3	LED controlled by master, Port2_3 switch
4	LED controlled by master, Port2_4 switch
5	LED controlled by master, Port2_5 switch
6	LED controlled by master, Port2_6 switch
7	LED controlled by master, Port2_7 switch

Table 1-6. Port 2 - Pins Connect to User-Accessible Header Row and Dip-Switches

0	Switch not used
1	Switch not used
2	Switch controls slave 2, Port1_1 LED blinking control
3	Switch controls slave 2, Port1_0 LED blinking control
4	Switch controls master, Port1_4 LED
5	Switch controls master, Port1_5 LED
6	Switch controls master, Port1_6 LED
7	Switch controls master, Port1_7 LED

1.7 Slave 2 Port Pin Usage

The section details the pin usage for the LIN bus PSoC slave 2:

Table 1-7. Port 0 – Pins

2	Push button controls master, Port1_3 LED
4	LIN bus TX
5	LIN bus RX
7	Push button controls master, Port1_2 LED

Table 1-8. Port 1 – Pins

0	Blinking LED controlled by slave 1, Port2_3 switch
1	Blinking LED controlled by slave 1, Port2_2 switch

1.8 Design IP

LIN Master Node and LIN Slave Node Design IP are provided on the CD and on the Cypress Semiconductor web site at http://www.cypress.com.

Design IP in PSoC Designer allows a user to import the required solution, precomposed of configurations and software APIs, to quickly and easily implement a LIN bus node. To import the Design IP into a project, use the *PSoC Designer Design Browser* (under Config >> Import Design).

The LIN Master Node Design IP and the LIN Slave Node Design IP documentation are located in the root directory of the CD.

1.9 Demonstration Projects

Also included on the CD are the three PSoC projects that implement the master and slave nodes on the demonstration board delivered with this design kit. The demonstration projects are in the following directories of the CD:

- Demonstration Projects\Master Node\MasterLinDemo
- Demonstration Projects\Slave 1 Node\CLinSlaveDemo
- Demonstration Projects\Slave 2 Node\CLinSlaveDemo2

1.10 Other Features

In addition to the three LIN nodes, the demonstration board provides several other features:

- Unregulated 12V DC 500 mA power supply for prototype use.
- Regulated 5V DC 500 mA power supply for prototype use.
- U12 header provides access to LIN bus for probing or bus extension.
- Disconnectable LIN nodes from the LIN bus by removing the JP1, JP2, or JP3 jumpers.
- Prototype area provides power and ground connections, and two strips of holes for prototyping. The holes are connected in rows of three to simplify connections, and if required, the traces can be cut.

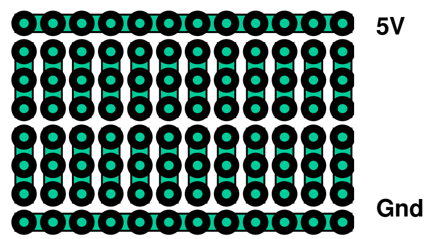

Figure 1-4. Prototype Area Through Hole Connections

- Install header U13 to short LED D2. This allows development of self-diagnostic indicator faults.
- Remove jumper JP4 to provide an open circuit at LED D4. This provides an additional way to develop self-diagnostic indicator faults.
- You can emulate master node and slave 1 using a universal emulation pod, from a PSoC Basic Development Kit, mounted on the standard 28-pin DIP foot.
- You can emulate slave 2 using a universal emulation pod mounted on the standard 8-pin DIP foot.
- The four unused LEDs in the master node LED array are provided on pads LED 0-3 for prototyping.
- The two unused LEDs in the slave 1 LED array are provided on pads LED 4-5 for prototyping.

1.11 Support

Support for the PSoC device, the development tools or the LIN bus demonstration board can be found on our web site at http://www.cypress.com, http://www.cypress.com/support or by calling the Applications Hotline at 425.787.4814.

2. System Architecture

2.1 Overview

The LIN bus, Local Interconnect Network, is an asynchronous, 1 wire, single master, multiple slave network. It is most commonly used in automobile networks.

2.2 Features of the PSoC LIN Bus 2.0 Design

- Single master, multiple slaves - up to 16 slaves.
- Message-based protocol.
- Single wire - maximum 40 m.
- Data rates of 2.4K, 4.8K, 9.6K and 19.2K are supported by master.
- Slaves capable of synchronizing to any baud rate from 2K to 20K.
- Self synchronization of slaves to master's speed.
- Data format similar to common serial UART format.
- Safe behavior with data checksums and bit-error detection.
- 100% LIN Bus 2.0 protocol-compliant.
- Master design uses minimal resources (only three digital blocks) and is easy to implement (using overlapping configurations).
- Slave designs use minimal resources (only four digital blocks) and are easy to implement (using overlapping configurations). The slave design for the CY8C21x34 device family uses the least amount of system resources.
- The PSoC design IP is provided for master and slave nodes in the following device families:
  - CY8C29x66 Industrial
  - CY8C27x43 Automotive
  - CY8C27x43 Industrial
  - CY8C21x34 Industrial

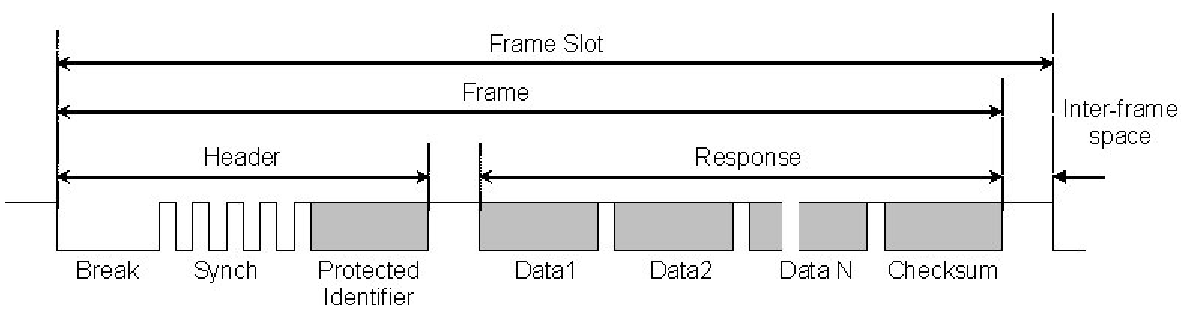

Figure 2-1. Structure of a LIN Frame

2.3 LIN Frame

2.3.1 Basic LIN Frame

The LIN communication takes place in frames. Figure 2-1 shows the structure of a frame.

It is made of a break field followed by 4 to 11 byte fields. Each byte field is transmitted as a serial byte as shown in Figure 2-2.

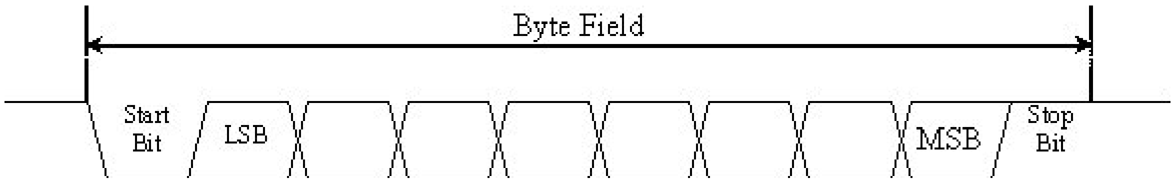

Figure 2-2. Structure of a Byte Field

2.3.2 Break Field

The break symbol is used to signal the beginning of a new frame. It is the only field that does not comply with Figure 2-2. A break is always generated by the master and is at least 13 bits of dominant value, including the start bit, followed by a break delimiter, as shown in Figure 2-3. The break delimiter is at least one nominal bit-time long. A slave node uses a break detection threshold of 11 nominal bit times.

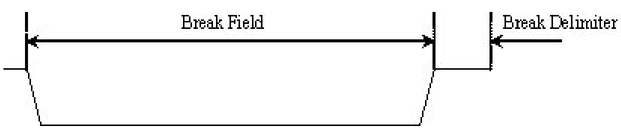

Figure 2-3. The Break Field

2.3.3 Synch Byte

The synch byte is sent to the slave to synchronize to the master's baud rate. The synch byte is nothing but a data field with 0x55 as data. The synch byte is shown in Figure 3-4.

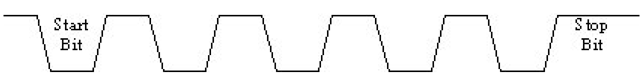

Figure 2-4. The Synch Byte

The slave measures the time between the start bit and the fourth falling edge of the synch byte. Then dividing this by eight, gives the single bit time. Based upon this time, the slave sets the clock to its UART so that it can send/receive the data bytes of the frame at the master's bit rate.

2.3.4 Protected Identifier

The byte that follows the synch byte is the protected identifier. This byte has two parts. Bits 0-5 form the actual identifier (0 to 63). Bits 6 and 7 form the identifier parity. The identifiers can be split into four different categories:

- Identifiers 0 - 59 are used for signal-carrying frames.
- Identifiers 60 (0x3C) and 61 (0x3D) are used for diagnostic frames.
- Identifier 62 (0x3E) is used for user-defined extensions.
- Identifier 63 (0x3F) is used for future protocol enhancements.

More details on protected identifiers are in the LIN Bus 2.0 specifications at http://www.linsubbus.org.

2.3.5 Data

The protected Identifier is followed by 1 to 8 bytes of data. The number of data bytes carried by a frame is defined in the LIN definition file (LDF). This file also defines whether the data bytes are sent from the master to a slave or from a slave to the master. Data that are longer than 1 byte are transmitted LSB first (Little Endian mode).

2.3.6 Checksum

The last field of a frame is the checksum. The checksum contains the inverted 8-bit sum with carry over all data bytes or all data bytes and the protected identifier. Checksum calculation over only the data bytes is called classic checksum and is used for communication with LIN bus 1.x slaves. Checksum calculation over both the data bytes and the protected identifier byte is called enhanced checksum and it is used for communication with LIN bus 2.0 slaves. The checksum is transmitted in a byte field. Use of classic or enhanced checksum is managed by the master node and determined per frame identifier; classic in communication with LIN bus 1.x slave nodes and enhanced in communication with LIN bus 2.0 slave nodes. Identifiers 60 (0x3c) to 63 (0x3f) always use classic checksum.

The complete LIN standard is available at http://www.linsubbus.org.

2.3.7 Frame Transfers on the LIN Bus

Only the master initiates a frame. The master allocates a time slot for each frame. The master also sends the frames in a predetermined sequence. The information sequence of the frames and the time slot for each frame is available in a table called Schedule table. Each entry of this table describes the protected identifier of the frame to be initiated and also the time to be allotted for that frame. When all the frames in the Schedule table have been transmitted, the next cycle starts again from the first frame of the table.

The LIN 2.0 API has many functions to manage the Schedule table. It has functions to select tables, to initiate the transfer of the next frame in the current table, and so on. More details on these APIs are found in section 3, Master Design APIs, on page 25.

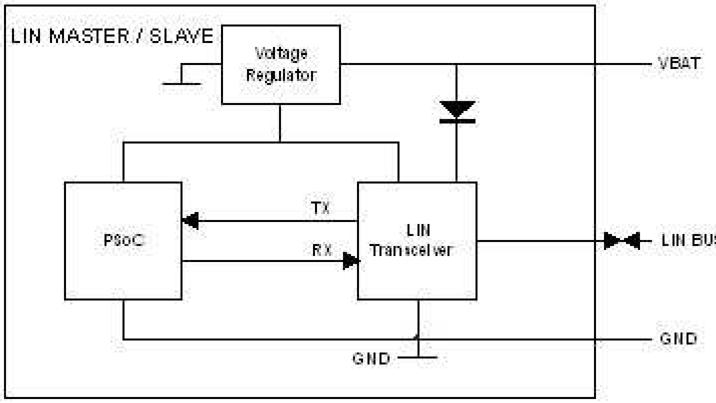

Figure 2-5. Hardware Configuration of a LIN Master/Slave

2.4 Hardware Architecture

Figure 2-5 shows the hardware architecture for the LIN Master/Slave.

2.4.1 LIN Transceiver

Because the physical LIN bus is held at Vbat in the range of 8 to 18 volts, a LIN transceiver device is required to connect the LIN bus with the PSoC chip. The LIN transceiver converts the single wire LIN bus at 8V – 18 volts to TTL-level TX and RX signals, which can be connected to the PSoC.

2.4.2 Voltage Regulator

You must use a voltage regulator to provide the PSoC Vcc supply. LIN transceivers with built-in regulators are available.

2.4.3 External Pin Connections

You have the option to decide which pins to use for the TX and RX pins in the design. These connections are done in the Device Editor of PSoC Designer. Details on how to configure the pins are in section 5, Using the Design IP, on page 35.

3. Master Design IP

3.1 Software Architecture

3.1.1 Overview

The software architecture maximizes interrupt processing to minimize the processing overhead on the end application. All message processing through configurations is performed at the interrupt level. Each stage is designed as a state machine and, upon completion, this state machine unloads itself and loads in the next required configuration to propagate the message to completion through the LIN message protocol sequence. Each message scheduled for processing is identified by the identifier byte in the header. The identifier is defined by the agreed master-slave relationship in the LIN description file (LDF). See the example LDF in section 5, LIN Description File (LDF) on page 43.

The master has a Schedule table where the frames are defined in the sequence in which they are transmitted on the bus. This table also contains an entry for the duration slot for each frame. In addition to the Schedule table, there is a Signal table in which the frames that are used in the system are defined. This table contains parameters such as the protected identifier, transfer type, checksum mode, data count, and the pointer to the frame buffer.

There are three transfer types:

- MASTER_TO_SLAVE where the master sends the data after the protected ID for the slave to process.
- SLAVE_TO_MASTER where the slave responds with data to the master's request.
- SLAVE_TO_SLAVE where the master initiates a frame and the data is transferred from one slave to another.

When a MASTER_TO_SLAVE transaction takes place, the master transmits the content of the frame's buffer to the slave. For SLAVE_TO_MASTER, the master receives the slave's response and deposits the data in the frame's buffer. For a SLAVE_TO_SLAVE transfer, the master discards all the data received at the end of the frame.

There are two types of checksum modes used, classic and enhanced. For LIN 1.x slave nodes, use the classic checksum for all frames. For LIN 2.0 slave nodes, use enhanced checksum for frames with identifiers 1 through 59. For identifiers 60 to 63, use classic checksum. While creating the Signal table, refer to the LDF to determine the slave version before deciding the checksum type.

The data count also depends upon the slave type. For LIN1.x slaves, the data count is fixed for different protected identifiers. For these frames, the data count is set to zero in the Signal table. When the master comes across a zero for data count in this table, it assumes that the default data count is used and extracts the data count from the protected identifier. For LIN 2.0 slaves, the data count can be from one to eight. So the data count entry can have any value from one to eight. Again, this value must be configured after studying the LDF.

The buffer pointer is an entry that has the address of the buffer for the particular frame. The master reads from or writes to the corresponding frame buffer using the buffer pointer parameter.

3.1.2 Foreground Processing

The main process must initialize the LIN function and then set the Schedule table using the l_sch_set function. After this, the main process performs the actual application. The successive frame transfers are initiated either inside the main loop or inside the schedule timer's interrupt service routine (ISR). The schedule timer is configured to generate an interrupt based upon the time base defined in the LDF. When a frame is read from the Schedule table, the time for the frame is also read and a loop counter is updated with this time count. This counter is decremented inside the schedule timer ISR. When it reaches zero, a flag is set to indicate that the next frame is ready for processing. The main function continuously checks this using the LinMaster_fIsLinReady function. When this flag is set, the main function calls the l_sch_tick function to start the next message. Alternatively, the l_sch_tick function can be called from the schedule timer ISR.

The main program is able to perform other functions inside the main loop. It checks the status of each frame transfer by checking the first byte of the frame buffer. It can also update frames or process received data.

More details on the l_sch_tick function are in the API section ahead.

3.1.3 Timing and Interrupts

Automotive applications are often real-time driven. As a result, the LIN driver only uses interrupts with no active loop or blocking functions. Overhead measurements made on a LIN bus with messages transferred at 19200 bauds and the PSoC CPU running at 24 MHz, show a 0% overhead between messages, and a maximum of 5% overhead while sending or receiving messages. Refer to Time Study on page 28 in this chapter.

3.2 Device Configurations

The LIN master design uses dynamic reconfiguration and has three configurations, the Synchro Break Configuration, Data Transmission Configuration and the Data Reception Configuration. The Synchro Break Configuration generates the break field. The Data Transmission Configuration sends the synchronization byte and any data bytes to be transmitted followed by the checksum byte. The Data Reception Configuration receives the slave's response data.

3.2.1 Synchro Break Configuration

Figure 3-1 shows the module placement for the Synchro Break Configuration. This configuration has one 8-bit counter (SB_Baud_Rate_Counter) that generates the baud clock. The output frequency of this clock generator is eight times the baud rate. There is a second 8-bit counter (SB_Bit_time_counter) that is used to generate an interrupt every bit time. Finally, there is a third 8-bit counter (Synchro_Break_Counter) that generates the actual break field. The period and compare values of Synchro_Break_Counter are set in such a way that one full cycle of the counter produces a break time approximately equal to 13 bit times and the break delimiter equal to one bit time. The TX and RX pins are compared to detect any bit error inside the Bit_time_counter ISR.

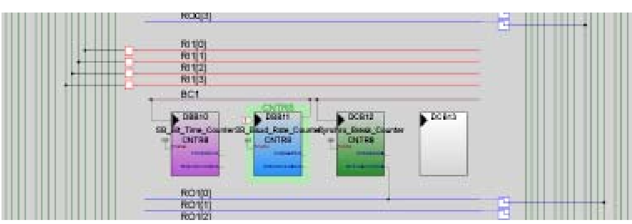

Figure 3-1. Synchro Break Configuration

3.2.2 Data Transmission Configuration

Figure 3-2 shows the user module placement for the Data Transmission Configuration. This configuration has one 8-bit counter that generates the baud rate (DT_Baud_rate_counter), one 8-bit counter that is used to generate interrupts every bit time for detecting bit errors (DT_Bit_time_counter), and one TX8 User Module to transmit data (TX8). The baud rate generator is configured to generate a clock eight times that of the baud clock and feed the TX8 block's clock input. When break field generation is complete, the Data Transmission Configuration is loaded and 0x55 is transmitted as the synch byte. Next, the protected identifier is transmitted. The protected identifier is followed by master's data and the checksum if the frame is MASTER_TO_SLAVE. Also during the data transmission, the Bit_time_counter generates an interrupt every bit time. Inside the Bit_time_counter's ISR, the TX and RX pins are compared. If they are not equal, then the BIT_ERROR flag is set and transmission of the current frame is aborted.

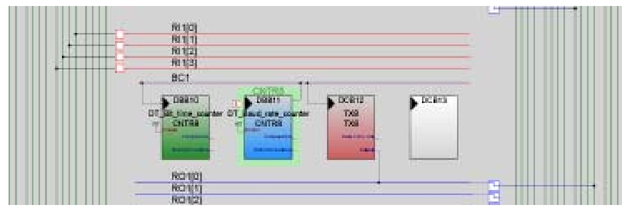

Figure 3-2. Data Transmission Configuration

3.2.3 Data Reception Configuration

Figure 3-3 shows the user module placement for the Data Reception Configuration. This has one 8-bit counter that generates the baud rate (DR_Baud_rate_counter), one 8-bit counter that is used to generate interrupts every five bit times for detecting the slave non-response timeout (DR_Bit_time_counter), and one RX8 User Module that receives data (RX8). The DR_Baud_rate_counter is configured to generate a clock eight times that of the baud clock and feed the RX8 block's clock input. The received bytes are transferred to the temporary buffer inside the RX8 ISR. When all the bytes indicated by the variable bNbDataToReceive have been received, the master processes the received data. Also, the bit time counter generates an interrupt every five bit times and a timeout counter is decremented inside the DT_Bit_time_counter ISR. The timeout is set as number of bit times according to the length of the frame. If the frame is not completed within this timeout (if the concerned slave stops transmitting), the Synchro Break Configuration is loaded and the "Slave Not Responding" error flag is set.

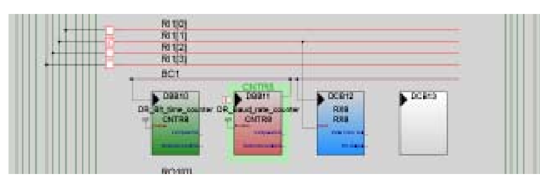

Figure 3-3. Data Reception Configuration

3.3 Firmware

3.3.1 Overview

The initiation of a frame is done by the l_sch_tick function. This function first reads the Schedule table and loads the frame parameters of the frame to transmit. It then loads the Synchro Break Configuration and starts the synchro break timer. This timer is configured to generate a dominant state of 13 bit times and a recessive (logic high level on the bus) state of one bit time. On the terminal count interrupt of this timer, the Data Transmission Configuration is loaded and the synch byte of 0x55 is transmitted. The protected identifier is transmitted next. If the transfer is MASTER_TO_SLAVE, all data bytes are transmitted one by one with the checksum as the last byte. If the transfer type is not MASTER_TO_SLAVE, then the Data Reception Configuration is loaded and the response from the slave is received. Data is processed after all bytes are received. Once the l_sch_tick function loads the Synchro Break Configuration and starts the synchro break timer, the rest of the frame is processed in the background, inside ISRs. More about the ISRs will be explained in the following sections. This enables the main function to run in the foreground. There are four different interrupts processed inside the LIN master. One or more of these interrupts may be active depending upon the active state. The code inside each of these ISRs is well commented so that it is very easy to understand the operation.

3.3.2 Synchro Break Interrupt

The l_sch_tick function loads the Synchro Break Configuration and starts the synchro break counter. The synch counter clock is from the baud rate clock generator, which runs at eight times the bit rate. The period of the synch counter is set to 111. This is equal to 14 bit times. The compare value of the counter is set to eight, which is equal to one bit time. So the output of the counter remains low for 13 bit times and high for one bit time. At the terminal count, the synchro break counter generates an interrupt. The Data Transmission Configuration is loaded inside this ISR. 0x55 is then placed on the TX buffer to generate the synch field. The rest of the frame is continued from the TX interrupt.

3.3.3 TX Interrupt

Inside the TX ISR, the program checks if this is the first interrupt. If this is the first interrupt, 0x55 was placed in the TX shift register and the buffer is empty. The bit time counter is started and its interrupt enabled. This counter's interrupt is used to check for bit errors. The bNbDataToSend variable is then checked. If this variable equals zero, no more bytes are sent and the bfLAST_BYTE_SENT flag is set. The completion of the frame takes place inside the Bit_time_counter's ISR. If the bNbDataToSend is not zero, then the next byte sent is transferred to the TX buffer. Then the bNbDataToSend variable is decremented by one before exiting the ISR.

3.3.4 RX Interrupt

If a response is expected from the slave, the Data Reception Configuration is loaded. This is done inside the Bit_time_counter's ISR for the Data Transmission Configuration. When a byte is received from the slave, this interrupt is generated. Inside the interrupt, the received data is placed on a buffer in the RAM. The bNbDataToReceive variable is decremented and checked if zero. If it is not zero, the ISR is exited. If this value becomes zero, it means that all the bytes were received and the Synchro Break Configuration is loaded to allow for the next frame initiation. Then the bfDATA_TO_COPY flag is checked. This flag is set if this is a SLAVE_TO_MASTER transaction and is not set if this is a SLAVE_TO_SLAVE transaction. For a SLAVE_TO_SLAVE transaction, the master has nothing to do with the received data so the data is discarded. For a SLAVE_TO_MASTER transaction, the checksum of the received data is verified. If the checksum is valid, the received data is transferred to the corresponding frame buffer. The checksum of the data bytes is compared with the last byte of the frame, which is the checksum transmitted by the slave. If they are identical, the data is valid. If the data is a slave's response to a master's diagnostics request, the received data is processed for the RSID, error code etc. of the slave response. Details of RSID may be found in the LIN 2.0 specifications.

3.3.5 Bit Time Interrupt

The bit time interrupt is used in all the configurations.

3.3.5.1 Synchro Break Configuration

In the Synchro Break Configuration, the bit time counter generates an interrupt every bit time. Inside the ISR, the TX and RX pins are compared to check if there is a bit error. If a bit error is found, the frame is aborted and the Synchro Break Configuration is reloaded. Also, when the TX state is sensed as logic high, the TX pin is disconnected from the global bus and made StdCPU and the TX pin's state is made logic high. This is done to prevent the counter output from becoming logic low upon terminal count before it is stopped inside the synchro break ISR. This unwanted low transition could be taken as the falling edge of the synch byte by the slaves connected to the cluster and may lead to communication errors.

3.3.5.2 Data Transmission Configuration

In the Data Transmission Configuration, the bit time interrupt is used to compare the TX and RX pins. The number of bits compared is tracked by the bNbBitsAnalyzed variable. This variable is initially set to 10, including the start and stop bits of a byte. Whenever this variable becomes zero, a byte is analyzed and the bfLAST_BYTE_TRANSMITTED flag is checked. If this flag is set, the last byte of the frame was sent. When this happens, the bNbDataToReceive variable is checked. If this is zero, then the Synchro Break Configuration is loaded. If this is not zero, then the Data Reception Configuration is loaded to receive the slave's response.

3.3.5.3 Data Reception Configuration

In the Data Reception Configuration, the bit time counter is configured to generate an interrupt every five bit times. Inside this ISR, a timeout counter is decremented by five. This timeout counter is initialized by the l_sch_tick function according to the number of data present in the frame. In a normal frame transaction, the frame is completed before this counter becomes zero. However, if the slave stops transmitting in the middle of the frame for any reason, and the timeout counter becomes zero, a timeout is detected, the SLAVE_NOT_RESPONDING error flag is set and the Synchro Break Configuration is loaded.

3.4 Source Code Files

Lin20CoreAPI.asm: This file has all the functions for the LIN core API.

Lin20NodeConfiguration.asm: This file has all the functions for the node configuration.

Lin20PhysicalLayer.asm: This file has all the code related to the proper operation of the LIN firmware. This file has all the ISRs described in section 3.3, Firmware on page 17.

RamVariables.asm: This file has all RAM variable allocations.

SignalTable.asm: This file has the Message table and the Protected ID table. This file must be modified according to the LDF.

ScheduleTable.asm: This file has the Schedule tables used in the master design. This file must be modified according to the LDF.

LinPowerManagement.c: This file has the functions that are required for the go to sleep and wakeup operations of the LIN master.

NodeConfigUtilities.c: This file has some functions that can be used for node configuration functions.

3.5 Header Files

Lin20CoreAPI.h: This file has all the function prototypes for the *Lin20CoreAPI.asm* file.

Lin20NodeConfiguration.h: This file has all the function prototypes for the *Lin20Nodeconfiguration.asm* file.

Lin20Defines.h: This file has the variable types defined in the LIN specifications.

Lin20Master.h: This file has the definitions of different constants and flags used in the firmware.

LinPowerManagement.h: This file has the function prototypes for the *LinPowerManagement.c* file.

NodeConfigUtilities.h: This file has the function prototypes for the *NodeConfigUtilities.c* file.

SignalTable.h: This file has declarations of the signal buffers and frame names used in the *SignalTable.asm* file.

ScheduleTable.h: This file has the declarations of the Schedule table names used in the *ScheduleTable.asm* file.

Lin20Master.inc: This file has the definitions of all the constants and flags used by the *Lin20PhysicalLayer.asm* file.

Of all source code and header files, you must modify the following files according to the LDF.

- *Lin20Master.inc*
- *SignalTable.asm*
- *SignalTable.h*
- *ScheduleTable.asm*
- *ScheduleTable.h*

3.6 Creating a Project Using the Design IP

Follow these steps to create a LIN master PSoC project using the Design IP.

3.6.1 Importing the Design

There are two ways to import the design. One is to create a new project in PSoC Designer and use the design-based project option. The other is to create a project and then import the design using the Design Browser. The recommended method is to create a new design-based project.

1. Select
   File >> New Project >> Create Design-Based Project.
2. Select the directory in which to create the project files.
3. Select the directory and name for a project.
4. The Design Browser opens. The Design Browser has two windows. The window on the left side is the Design Browser itself where you select the design. The window on the right side shows the data sheet for the selected design. On the top of the Design Browser window there are two radio buttons that select between “Browse File System” and “Select From Design Catalog.” Click the "Browse File System" option. Navigate to the "\Design IP\Lin2.0 Master" directory on the CD, and open the folder corresponding to the device that you want to use. Then select the .cfg file in this directory. Now the data sheet window on the right shows the data sheet of the LIN master design.
5. Below the Design Browser window, there are two radio buttons, “Overwrite configurations with same name” and “Resolve configuration name conflicts.” Use these options when importing a design into an already-existing project and if some of the configurations from the existing project have the same name as that of the imported design.
6. Below this there are two windows, “Resolve name conflicts” and the “Specify base configuration.” The “Specify base configuration” window has the Synchro Break Configuration, Data Transmission Configuration and Data Reception Configurations listed. Do not select any of these options.

7. The “Resolve name conflicts” window lists functions in the imported design that have the same name as functions in the existing project. When there is a name conflict, clicking the “Auto Resolve” button automatically renames the conflicting function names.
8. Below this, details of the design such as date of creation, description and the base part number are displayed.
9. Click **OK**.
10. Now in the Device Selection window, select the device for the project.
11. Select “Generate main file using C.”
12. Select “Device Editor” as the Designer State.
13. Click **Finish**.
14. A Design Import Status window opens and displays the import status.
15. When the design is imported, PSoC Designer opens the Device Editor.
16. Four configurations are visible. The base configuration with the project name, the Synchro Break Configuration, Data Transmission Configuration and the Data Reception Configuration.
17. Go to Project >> Settings, Device Editor tab. In the configuration initialization type, select “Direct Write (Speed Efficient).”
18. Now switch to the base configuration and select all the user modules to include in the main application.

3.6.2 Configuring Global Resources

Now switch to the Interconnect View and select the base configuration. First, configure all the global resources related to the LIN design. Whatever changes made to the base configuration, are reflected in the other three loadable configurations.

1. Set CPU speed to 24 MHz. (Set the CPU speed to 12 MHz for the CY8C27x43 automotive grade device.)
2. Set 32 kHz to External.
3. Set the PLL to Enabled.
4. Set VC1 divider to 12.

These are the required global resources for the LIN master. The clock VC1 is used as the source clock to LIN modules. The divider is set to 12 in the firmware so that the output of VC1 is 2 MHz. Take this into account when using VC1 and VC2 in the main application. You can set all the other global resources in your main application.

3.6.3 Configuring GPIO

Next, decide the TX and RX pins of the LIN bus. To properly select their drive modes in all configurations, follow these steps carefully.

1. Switch to the base configuration. Use the Config >> Restore default pinout. All the pins in the GPIO configuration pane become StdCPU, High Z Analog, DisableInt. Repeat this step for the synchro break, data transmission and data reception configurations.
2. Return to the base configuration.
3. In the GPIO configuration pane, rename the port pin you plan to use as the RX pin to “RX.” Then rename the pin you plan to use as the TX pin as “TX.” Capitalize these letters.
4. In the Select column of the RX pin, select the GlobalInOdd_x or GlobalInEven_x. The drive mode automatically becomes High Z.
5. In the Select column of the TX pin, select the GlobalOutOdd_x or GlobalOutEven_x. The drive mode automatically becomes Strong.
6. Switch to synchro break, data transmission and data reception configurations and check that these changes are reflected.

The GPIO configuration is done. After this, modify the GPIO of the other port pins according to the main project requirements. Whenever a modification is done in the base configuration, the same settings are updated in the other three configurations. Thus, regardless of which configuration is active, the GPIO state of the main application is maintained. When the process is complete, the configuration of the TX and RX pins looks like this:

Table 3-1. TX Pin

Configuration	Name	Port	Select	Drive	Interrupt
Base	TX	As selected	GlobalOut	Strong	DisableInt
Synchro Break	TX	As selected	GlobalOut	Strong	DisableInt
Data Transmission	TX	As selected	GlobalOut	Strong	DisableInt
Data Reception	TX	As selected	GlobalOut	Strong	DisableInt

Table 3-2. RX Pin

Configuration	Name	Port	Select	Drive	Interrupt
Base	RX	As selected	GlobalIn	High Z	DisableInt
Synchro Break	RX	As selected	GlobalIn	High Z	DisableInt
Data Transmission	RX	As selected	GlobalIn	High Z	DisableInt
Data Reception	RX	As selected	GlobalIn	High Z	DisableInt

3.6.4 Routing the Signals

The next step is to route the signals to the digital blocks of the LIN configurations.

1. Go to the Synchro Break Configuration.
2. Route the Compare Out of the synchro break counter to the appropriate Row_1_Output_x line. For example, if you have configured P0[3] as TX pin, then route the Compare out to Row_1_Output_3 net.
3. From this Row_1_Output_x net, route the signal to the appropriate GlobalOut bus to which the TX pin is connected.
4. Switch to the Data Transmission Configuration.
5. Route the output of the TX8 to the same Row_1_Output_x line used by the synchro break counter (step 2) and from there to the GlobalOut bus to which TX pin is connected.
6. Switch to the Data Reception Configuration.
7. Route the Global_Input net to which RX is connected, to an appropriate Row_1_Input_x net. Select Synch to SysClk in the Synchronization box. For example, if P0[2] is used as RX, then connect GlobalIn_Even_2 bus to Row_1_Input_2 net.

8. Select Row_1_Input_x (step 7) as the input to the RX8 User Module.
9. Switch to the base configuration.
10. Make the connection from Row_1_Output_x net to the Global bus as used by the Data Transmission and Synchro Break configurations in the base configuration.
11. Make the connection from Global_In bus to the Row_1_Input_x net as used by the Data Reception Configuration.

With this routing of signals, the hardware configuration is complete.

3.6.5 Setting the Baud Rate

In the *Lin20Master.inc* file, there are four constants: BR2400, BR4800, BR9600, and BR19200. These correspond to 2.4K, 4.8K, 9.6K, and 19.2K baud rates, respectively. Set the value of one of these constants to 1 to correspond to the baud rate. This constant is used to select the period and compare values of the baud rate generator. Make only one of these constants 1.

3.6.6 Adding the Schedule Timer

An important module necessary for the proper functioning of the master is the schedule timer. This timer is used to generate the frame slot timings for the LIN bus. This is placed by the user in the base configuration. Follow these steps.

1. Go to the base configuration.
2. Select a Counter8 User Module and add it to the project.
3. Rename it "ScheduleTimer."
4. Place it in any of the available digital blocks. Avoid placing it in a digital block used by the LIN design in any of the other configurations.
5. Configure the parameters for the counter as:
    - Clock: according to the time base
    - Enable: High
    - CompareOut: None
    - TerminalCountOut: None
    - Period: As per time base
    - CompareValue: ½ (Period + 1)
    - CompareType: Less Than or Equal To
    - InterruptType: Terminal Count
    - ClockSync: As per the Clock source
    - InvertEnable: Normal

3.6.7 Setting the Source Clock and Period

Set the source clock and period according to the time base specified in the LDF. In the example, the time base is 1 ms. Make the counter output frequency 1 kHz. Since the configuration of the clock resources is very flexible, there are different combinations of clock source and period that arepossible. For example:

- Clock: VC2.
- VC2 Divider = 10. As VC1's divider is already set to 12 by the LIN firmware, the output frequency of VC2 is 200 kHz.
- Period = 199. VC2 is divided by (Period + 1), i.e., 200 to give an output frequency of 1 kHz.

3.6.8 Configuring the Signal Table

You now need to configure the frames used in the system in the *SignalTable.asm* file. This configuration is done according to the LDF. For this example, refer to the LDF provided in section 5, LIN Description File (LDF) on page 43. According to the LDF file, a total of four frames are used.

- VL1_CEM_Frm1: This frame is published by the master and is subscribed to by the slaves CPM and DIA. The protected ID for this frame is 0xF0. The length of this frame is eight bytes.
- VL1_CPM_Frm1: This frame is published by slave CPM and is subscribed to by the master. The protected ID of this frame is 0x9C. The length of this frame is two bytes.
- VL1_CPM_Frm2: This frame is published by slave CPM and is subscribed to by the master. The protected ID of this frame is 0x32. The length of this frame is one byte.
- VL1_DIA_Frm1: This frame is published by slave DIA and is subscribed to by the master. The protected ID of this frame is 0x80. The length of this frame is two bytes.

3.6.9 RAM Allocation

First the buffers for these frames are allocated in RAM. A name is given to each frame and the buffer is named as Buffer<FrameName>. The frames are named Frame1, Frame2, Frame3, and Frame4. The buffers for these frames are BufferFrame1, BufferFrame2, BufferFrame3, and BufferFrame4. When assigning RAM, one extra byte is allocated for each frame. This byte is used as the status byte of that particular frame. The LIN firmware updates the status of transaction of each frame in this byte. The status byte is the first byte of the array. Another buffer is used by the LIN firmware for diagnostic frames. This buffer is named "abDiagBuffer." The diagnostic frames always carry eight bytes. This makes the total length of this buffer nine bytes.

Here is an example of RAM allocation.

```
area bss(ram)

_abDiagBuffer:
 abDiagBuffer:  BLK 9; Buffer for Diagnostic
frames
_BufferFrame1:
 BufferFrame1:  BLK 9; Buffer for Frame1
_BufferFrame2:
 BufferFrame2:   BLK 3; Buffer for Frame2
_BufferFrame3:
 BufferFrame3:   BLK 2; Buffer for Frame3
_BufferFrame4:
 BufferFrame4:   BLK 2; Buffer for Frame4
```

3.6.10 Frame Definition

Now the frames are defined in the Signal table. Each frame has the following parameters entered in this order:

A. **Checksum Type**: This entry defines the checksum type used for the particular frame. There are two types of checksums, CSUM_CLASSIC and CSUM_EXTENDED. CSUM_CLASSIC is used for frames that belong to LIN slaves of version 1.3 or less and for diagnostic frames. CSUM_EXTENDED is used for LIN 2.0 slaves.
B. **Data Count**: This entry indicates the length of data carried by the frame. For LIN1.x slaves, this parameter is left as zero. When the l_sch_tick function finds that the data count is zero, it calculates the standard length for the frame from the protected ID.
C. **Buffer Pointer**: This entry is the pointer to the buffer for this frame that is reserved in RAM. Enter the name of the buffer in this entry. The compiler will translate this to the RAM address and create the table.
D. **Data Direction**: This entry indicates the direction of data flow. MASTER_TO_SLAVE indicates that the slave must receive data from master and SLAVE_TO_MASTER indicates that the slave must transmit a response to the master. SLAVE_TO_SLAVE indicates that the data flow is from one slave to another. In this type of transaction, the master's job is only to generate the header of the frame.
E. **Protected ID**: This entry is for the protected ID for the particular frame.

```
_Frame1:
db 8                ;Data Count
 Frame1:
db CSUM_EXTENDED    ; Checksum Type
db 0                ; Data count
db BufferFrame1     ; Buffer address
db MASTER_TO_SLAVE  ; Direction
db 0xF0             ; ID

_Frame2:
 Frame2:
db CSUM_EXTENDED    ; Checksum Type
db 2                ; Data count
db BufferFrame2     ; Buffer address
db SLAVE_TO_MASTER  ; Direction
db 0x9C             ; ID

_Frame3:
 Frame3:
db CSUM_EXTENDED    ; Checksum Type
db 1                ; Data count
db BufferFrame3     ; Buffer address
db SLAVE_TO_MASTER  ; Direction
db 0x32             ; ID

_Frame4:
 Frame4:
db CSUM_EXTENDED    ; Checksum Type
db 2                ; Data count
db BufferFrame4     ; Buffer address
db SLAVE_TO_MASTER  ; Direction
db 0x80             ; ID
```

In addition to these user-defined frames, there are some frames used by the master for diagnostics. They are the master request and slave response frames. For both these frames, the data count is eight, the checksum type is extended, and the response buffer is abDiagBuffer.

```
_MasterRequest:
 MasterRequest:
db CSUM_CLASSIC     ; Checksum Type
db 8                ; Data count
db abDiagBuffer     ; Buffer address
db MASTER_TO_SLAVE  ; Direction
db 0x3C             ; ID

_SlaveResponse:
 SlaveResponse:
db CSUM_CLASSIC     ; Checksum Type
db 8                ; Data count
db abDiagBuffer     ; Buffer address
db SLAVE_TO_MASTER  ; Direction
db 0x7D             ; ID
```

Once the frame definition and the buffer allocations are complete, export these names as Global so they are used in the main application and the LIN API. All the frame names and buffer names must be declared with and without an underscore. The name with the underscore is to enable the name to be used in C functions. For the above example, the following names are exported.

```
export _MasterRequest
export  MasterRequest
export _SlaveResponse
export  SlaveResponse
export _Frame1
export  Frame1
export _Frame2
export  Frame2
export _Frame3
export  Frame3
export _Frame4
export  Frame4

export _abDiagBuffer
export  abDiagBuffer
export _BufferFrame1
export  BufferFrame1
export _BufferFrame2
export  BufferFrame2
export _BufferFrame3
export  BufferFrame3
export _BufferFrame4
export  BufferFrame4
```

Once these names are exported, they are available to any assembly function. To use these names in C, they must be declared in a C header file. This is done in the *SignalTable.h* file. All frame names are defined as “const char” as they are in the Flash and the buffer names are defined as “BYTE” as they are in the RAM. The following are the entries in the *SignalTable.h* file.

```
// Definition of Frame Buffers to be used by
the main program
extern BYTE BufferFrame1[];
extern BYTE BufferFrame2[];
extern BYTE BufferFrame3[];
extern BYTE BufferFrame4[];
extern BYTE abDiagBuffer[];

// Definition of Frame names to be used by
the main program
extern const char Frame1[];
extern const char Frame2[];
extern const char Frame3[];
extern const char Frame4[];
```

3.6.11 Schedule Table

3.6.11.1 Structure of Schedule Table

Once the frames used in the cluster are defined, you need to create Schedule tables. The Schedule tables are found in the “*ScheduleTable.asm*” file. To create a Schedule table, you first select a name. For the example, create a Schedule table called Schedule1. The table entries are entered in this order.

A. Frame Name: The name of the frame to be transmitted.

B. Frame Time Constant: The number of schedule timer interrupts before the next frame is transmitted. This value is derived from the “Node Capability File” of the nodes. The node capability file has frames defined with minimum and maximum frame times. If these values are not given in the node capability file, then use the formula given in “Section 2.2 Frame Slots” in the LIN 2.0 protocol specification. The equations are:

$$T_{Header\ Nominal} = 34 * T_{Bit} \quad \textbf{Equation 1}$$

$$T_{Respone\ Nominal} = 10 * (N_{Data} + 1) * T_{Bit} \quad \textbf{Equation 2}$$

$$T_{Frame\ Nominal} = T_{Header\ Nominal} + T_{Respone\ Nominal} \quad \textbf{Equation 3}$$

This calculation does not consider the response space, byte space or inter-frame space. The actual time used is according to the LIN 2.0 protocol specification.

$$T_{Frame\ Maximum} = 1.4 * T_{Frame\ Nominal} \quad \textbf{Equation 4}$$

From this time, calculate the number of schedule timer overflows based upon the schedule timer time base.

Frame Time Constant = Frame Time / Timebase

For example, if the frame time is calculated as 20 ms and the time base is 1 ms, then the frame time constant is 20 ms / 1mS = 20.

3.6.11.2 An Example Schedule Table

Here is an example Schedule table. The name of the table is Schedule1. This table has Frame1, Frame2, Frame3 and Frame4 (which are defined in the *SignalTable.asm* file) in the order they are entered in the Schedule table.

Schedule table example:

```
_Schedule1:
 Schedule1:
dw Frame1, 20
dw Frame3, 10
dw Frame2, 10
dw Frame4, 10
dw 0xFFFF
```

The last entry in the Schedule table is the table terminator. When the l_sch_tick function comes across 0xFFFF, it goes back to the start of the Schedule table.

3.6.11.3 Diagnostic Schedules

In addition to the user-defined Schedule tables defined in the LDF, there are some tables defined in the API that are available for other diagnostic functions. They are listed below.

- **ScheduleNodeConfiguration***:* This schedule contains a master request frame and a slave response frame.
- **ScheduleGoToSleep**: This schedule contains a master-request frame.
- **L_NULL_SCHEDULE**: This schedule is null and does not transmit any frame.

These schedules are at the end of *ScheduleTable.asm* file. Set these tables using the l_sch_set function before calling the node configuration functions or the l_goto_sleep function.

Once the Schedule table definitions are done, export the schedule names so that they are referenced by the LIN functions and the main program. This is done in the beginning of the *ScheduleTable.asm* file.

```
; Export Schedule Names
export _L_NULL_SCHEDULE
export  L_NULL_SCHEDULE
export _Schedule1
export  Schedule1
export _ScheduleNodeConfiguration
export  ScheduleNodeConfiguration
export _ScheduleGoToSleep
export  ScheduleGoToSleep
```

Then declare these names in the *ScheduleTable.h* file so that these schedules are referenced in the C program.

```
// Definition of Schedule Names to be used in
the main program.
extern const char L_NULL_SCHEDULE[];
extern const char Schedule1[];
extern const char ScheduleNodeConfigura-
tion[];
extern const char ScheduleGoToSleep[];
```

3.6.12 Adding the Main Application

Now that the LIN 2.0 master is configured, you can add the main application. Follow the normal procedure of building an application using PSoC Designer. Place the user modules in the base configuration, finish the routing, and generate the application.

In the *main.c* file, follow these steps to properly start the LIN firmware and update the LIN frames.

1. Call the l_ifc_init function to initialize the LIN function.
2. Enable the Global Interrupts using the M8C_EnableGInt macro.
3. Write a 0 to the first byte of all the frame buffers. This is to clear the status bytes of the buffers.
4. Perform node configuration if necessary.
5. Set the schedule that the master must follow.
6. Inside an infinite loop, add the application code.

- Keep checking for a completion of transaction of each frame using the bfLAST_TRANSACTION_OK flag on the first byte of the frame buffer, then process the data.
- If polling is used to initiate a frame transfer, use the LinMaster_fIsLinReady function to check if the current time slot is over before calling the l_sch_tick function.
- If using an interrupt-driven frame transfer, then call the l_sch_tick function inside the ScheduleTimer_ISR function found in the *FrameTiming.c* file.

The example main file given in section 5, Demonstration Projects on page 43, uses the polling method of frame transfer.

3.6.13 Special Features

3.6.13.1 Low Power Management

For power management there are some functions available in the *LinPowerManagement.c* file.

A. **ShutdownLin**: This function properly stops all the active LIN resources and makes the pins HighZ so that the processor enters a low power state. Inside this function, there is an area into which the user must enter code to stop all the resources used by the main application. Also, if the main application uses analog resources, turn off the analog reference and the analog buffers to minimize current consumption during sleep state. It also disables all the interrupts except the GPIO interrupt. Call this function to put the master in power-down mode after it executes the l_goto_sleep function putting all the slaves in the cluster to power-down mode.

B. **SleepLoop**: When this function is entered, the M8C_Sleep macro is executed to put the processor to sleep. Once the processor is put to sleep, it wakes up only upon an interrupt. Since all interrupts are disabled except the GPIO interrupt, when a slave in the cluster issues a wakeup command (dominant state for a time of 250 μs to 5 ms), the processor wakes up and enters a loop where it waits for the bus to go to recessive state. When this happens, it checks the length of the dominant state. If this length is within a specified limit, it returns