AN2059

LIN Basics and Implementation of the MCC LIN Stack Library on 8-Bit PIC® Microcontrollers

Author: *Mary Tamar Tan,*
Brian Bailey,
Han Lin
Microchip Technology Inc.

INTRODUCTION

LIN (Local Interconnect Network) is a low-cost serial communications protocol implemented mostly in automotive networks. It is typically used for mechatronic nodes in automobiles, but is also well-suited for industrial applications. Users new to LIN communication often struggle with the overhead of the high layers of LIN protocol intricacies (e.g., LDF, NCF files). When in fact, the goal is to simply and quickly evaluate LIN MCUs and LIN PHYs hardware with very basic LIN message transmission. To assist with rapid prototyping, the complimentary MCC LIN Code Generator was created. When your LIN application is ready for production, customizable production-proven LIN stacks and support are available from our LIN design and third party partners.

The first part of this application note covers the basics of LIN to provide a brief background on how it works. However, it is highly recommended that the reader is familiar with the LIN specification v2.2A (the most current specification available at the time of this publication).

The second part describes a sample demonstration of the LIN protocol using the Microchip MPLAB® Code Configurator (MCC) LIN Stack Library. This document provides step-by-step procedures on how to generate the LIN master and slave drivers in just a few minutes.

MCC LIN Stack Features

The MCC generates code for following basic LIN functions:

- Constructs LIN message frame by combining EUART bytes
- Automates LIN PID generation with user-defined frame names
- Generates LIN classic checksum value based on LIN message frame configuration
- Generates a scheduler example code showing the use of generated LIN APIs
- Supports only unconditional type frame
- Allows user-defined frame names
- Allows user-defined data length (1 to 8 bytes)
- Configurable time out and period down to one millisecond
- Supports hardware interrupts

Note: The MCC LIN stack is intended for basic LIN introduction and for prototype purposes only. It is limited to low-level LIN layer to allow the most basic LIN message transmission. Refer to the LIN stack section for more detail about the limitations.

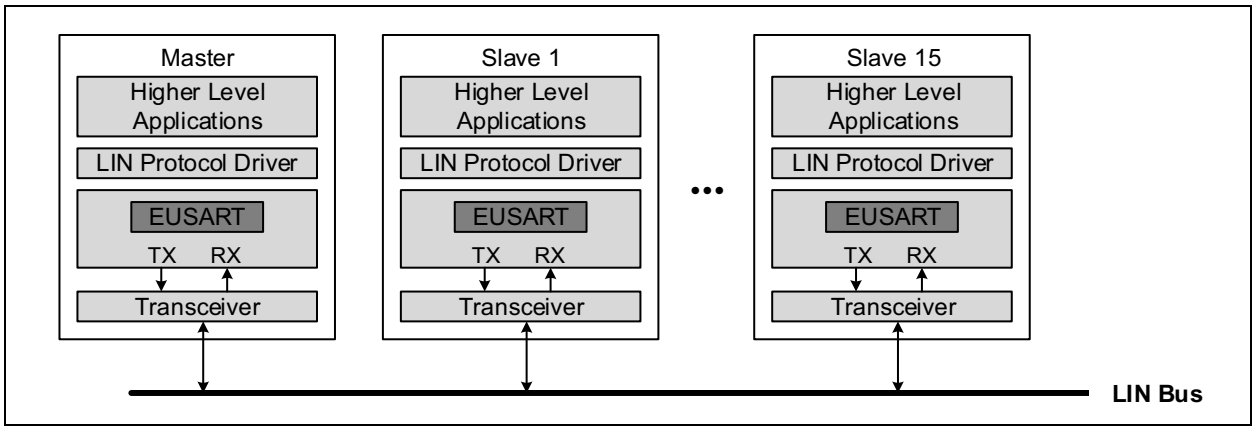
LIN BASICS

The following sections provide a brief description on the basics of LIN to help readers understand the MCC LIN Stack implementation more easily.

LIN Network Configuration

A LIN cluster is composed of a single-wire bus, a master node and up to 15 slave nodes. A typical network connection is shown in Figure 1. The master controls the bus activity while the slaves send or receive information depending on the scheduled tasks.

FIGURE 1: LIN NETWORK CONFIGURATION

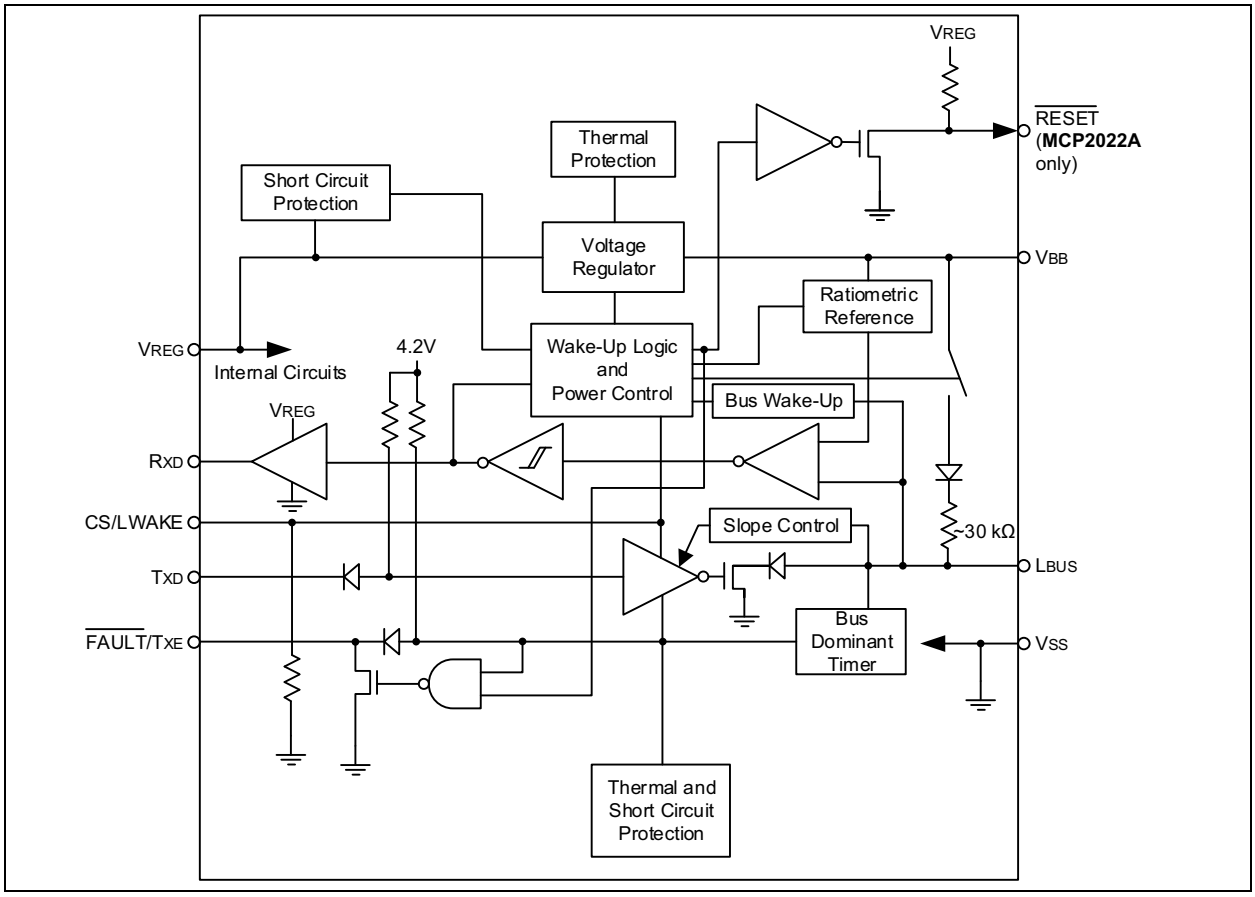


The MCC LIN Stack, which is the core focus of this application note, provides the interface between the physical interface and the higher-level application firmware. The EUSART module is the key element used in LIN communications. It acts as the serial engine for LIN putting serial control in hardware rather than in software. Thus, miscellaneous processing can be done while data is being transmitted or received. The transceiver acts as a bidirectional, half-duplex communication physical interface between the EUSART module and the bus. See **Section "Electrical Connections"** for more information regarding the LIN physical layer.

Electrical Connections

The LIN bus operates between 9V and 18V. Typically, the microcontroller LIN I/O pins voltage levels are adjusted to the LIN bus levels by a transceiver. This allows the microcontrollers to operate at 5V levels, while the bus operates at higher levels. Microchip offers a portfolio of LIN transceivers, some having additional built-in features, such as Internal Voltage Regulator (V_{REG}) and Windowed Watchdog Timer (WWDT). A sample block diagram for a MCP202XA LIN transceiver is shown in Figure 2.

FIGURE 2: LIN TRANSCEIVER MCP202XA BLOCK DIAGRAM

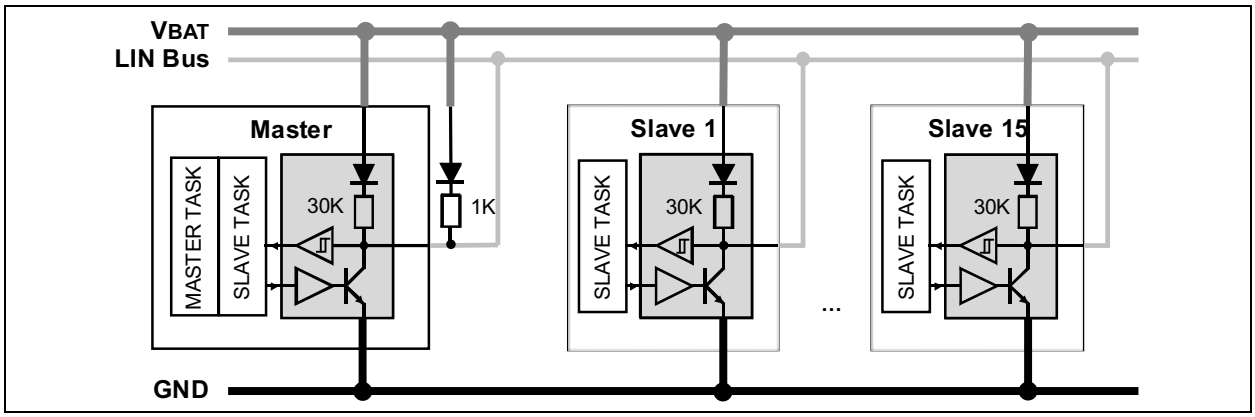


Refer to www.microchip.com/lin for the complete list of LIN transceivers from Microchip.

The bus is terminated to V_{BAT} at each node. The master should be terminated through a 1 kΩ resistor while the slaves through a 30 kΩ resistor. As shown in Figure 2, the MCP202XA LIN transceivers already have an internal 30 kΩ terminating resistor, hence external termination for slaves are no longer required. The maximum bus length is designed to be 40 meters. Figure 3 shows a sample LIN physical connection.

Refer to MCP202XA data sheet (DS20002298) for "Typical Application Circuit" at www.microchip.com/MCP2021A.

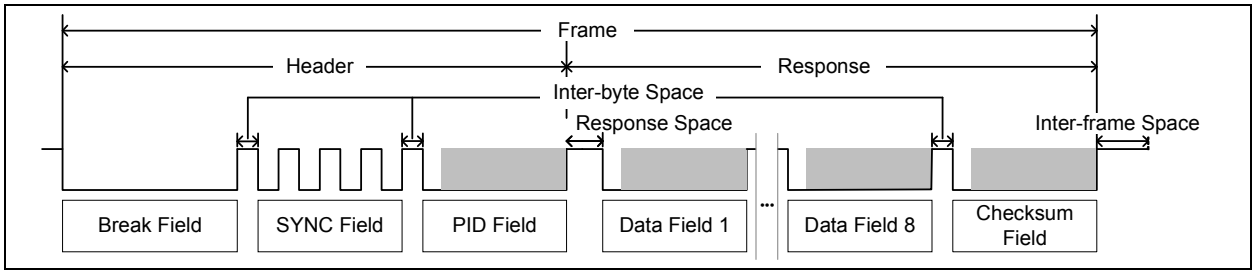
FIGURE 3: LIN PHYSICAL LAYER



Frames and Signals

A frame is defined as the entity that is being transferred on the LIN bus. A frame is divided into two parts: the frame header and the frame response. Each serial data byte starts with a Start bit (Dominant signal) and is terminated by a Stop bit (Recessive signal). When signal logic is zero, it is called the dominant signal. When signal logic is one, it is called the recessive signal.

FIGURE 4: FRAME STRUCTURE



The frame header is always sent by the master and it consists of the Break Field, the Sync Field and the Protected Identifier (PID) Field.

The Break Field consists of at least 13-bit time long of dominant signal followed by a Break delimiter. The Break delimiter is at least one-bit time long of recessive signal. This is followed by a Sync Field with a data value of 0x55. The Break Field sequence enables a slave to identify the beginning of a new frame. The Sync Field allows the slave to synchronize with the master clock.

The PID field consists of six frame identifier bits and two parity bits. The frame ID denotes a unique message address, but does not necessarily define a specific destination of the message. The MCC LIN stack includes an algorithm to automatically calculate the parity of the frame ID.

The frame response can be from the master itself or any slave. Only the node with the matching IDs will send the response. The response consists of the Data Field and the Checksum Field.

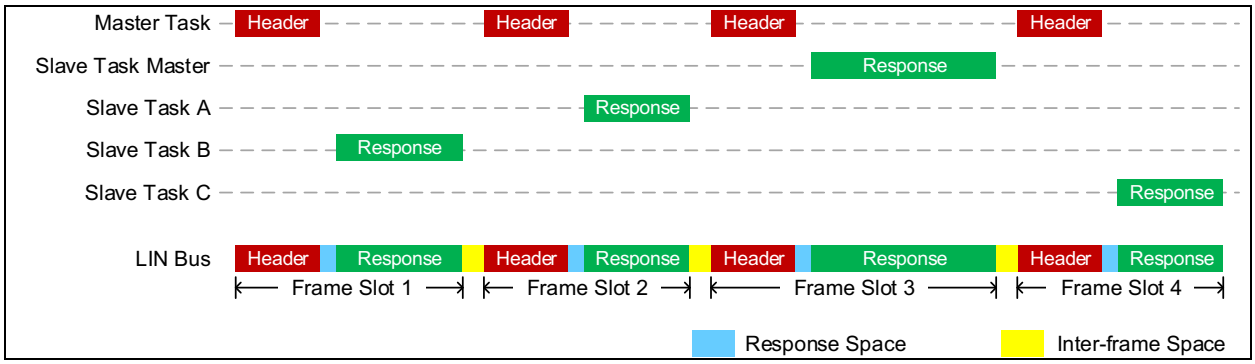
The Data Field is composed of signals. A signal is either a scalar value or a byte array. A scalar signal carries value between 1 bit to 16 bits. The byte array signal carries value between 1 byte to 8 bytes. A signal is always present at the same position of the data field for all frames with the same frame identifier. It should also be sent by a single publisher, though there are no limits to the number of subscribers. These signals are used in the higher-level application. A frame is always ended by the checksum field. The MCC LIN Stack implements the classic checksum calculation which is performed on the data bytes only.

The inter-byte space is the time between the end of the Stop bit of the previous field and the Start bit of the succeeding byte. The response space is the time between the header and response field. Lastly, inter-frame space is the time between the end of a frame to the start of a next frame.

LIN Bus Timing

The transmission and reception in a LIN bus are considered to be both deterministic and periodic. This requirement is made possible through the use of schedule tables. The schedule table lists all the frames that need to be present on the bus at a predefined time interval. This prevents the bus from overloading and also ensures that each frame is properly transmitted and received. Figure 5 shows a sample transmission of frames with different response publishers. The subscribers of the signals packed in these frames can be any of the nodes connected to the bus.

FIGURE 5: FRAMES ON A LIN BUS



The minimum time unit used in LIN is called the time base. This controls the timing of frames in the schedule table. The time slot for a single frame has a period which is an integer multiple of the time base. The time base can be generated by any timing source such as the Timer0, Timer1/3/5 or Timer2/4/6 peripherals of PIC® microcontrollers. Currently, the MCC LIN stack time base is set to 1 ms.

Now that the basics of the LIN protocol have been covered, the following sections will walk the user through the steps required to configure and generate master and slave LIN demo applications using the MCC LIN tool.

MCC LIN STACK LIBRARY

The MPLAB Code Configurator (MCC) is a software plug-in in the MPLAB X Integrated Development Environment. It utilizes a Graphical User Interface (GUI), making it an easy-to-use tool in generating drivers for different peripherals of PIC® MCUs. Aside from peripheral modules, it also includes libraries for different stacks and interfaces. Included in these libraries are the LIN Stacks for both master and slave nodes.

MCC LIN Stack System Requirements

- MPLAB X IDE V3.10 or newer
- MPLAB Code Configurator (MCC) v3 or newer
- Java JRE v1.8 or newer (Follow MCC release note to setup MPLAB X for latest Java)

Advantages and Limitations

The MCC LIN Stack has the following advantages and limitations.

Advantages:

- Implements on-chip EUSART and timer module allowing the program to run in the background. This also reduces firmware. Selectable EUSART module (for devices with more than one EUSART) and timing source (e.g., Timer1, Timer2/4/6).
- Easy to create schedule table.
- Automatically handles the transmitted or the received frames, making them easily available for the application.

Limitations:

- Does not define signal names in the data fields. For example, a frame may have two data bytes that contain signals for two bits of window status, six bits of door status and eight bits of temperature status. The user will have to further define these signal names in the code to interpret the two bytes of data. However, the MCC LIN generator does define name to the frame PID. So for the purpose of prototyping, the user may use the frame name to help identify what each frame carries.
- Does not support the full LIN protocol. Only implements basic low layer LIN frame transmissions.

MCC LIN Stack PID Assignment

Each frame's name listed on both the master and the slave tables is allocated with a unique PID. The PID is comprised of six frame identifier bits (LSBs) and two parity bits (MSBs). The MCC LIN Stack automatically assigns the frame identifier value to the frames listed on the LIN table. The frame identifier allocation starts with '0' for the topmost frame on the list and increments by one for the frame on the next row, and so on.

Note:	Only 60 frames (0 to 59) should be listed to conform to the LIN specification for signal (data) carrying frames. Frame identifiers 60 to 63 are reserved for other special functions.

The parity bits P0 and P1 are automatically calculated by the stack. Alternately, an ID to PID look-up table is provided in **Appendix A: "Frame PID Look-Up Table"**. This table can be useful when the user uses testing tools such as LIN Serial Analyzer from Microchip. The user can find the corresponding PID value based on the frame ID to transmit frames.

Associated Peripherals

Aside from the LIN Stack Library, two peripheral modules also need to be configured. These are the EUSART module for the transmission and the reception of frames and a timer module for the LIN bus time base.

EUSART MODULE

The Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART) module of 8-bit PIC microcontrollers is a serial communications peripheral that contains all the clock generators, shift registers and data buffers necessary to perform an input or output serial data transfer independent of device program execution. Besides the basic USART features, the module exhibits a few capabilities which make it ideal for use in LIN bus systems:

- Automatic detection and calibration of the baud rate
- Wake-up on Break reception
- 13-bit Break character transmit

To comply with the LIN standard, each character transmission should be configured to have one Start bit followed by one or up to eight data bytes and always terminated by a Stop bit. The Start bit is always a '0' and a Stop bit is always a '1'. Each bit persists for a period of 1/(baud rate). The Baud Rate Generator of the EUSART module automatically derives the baud rate from the system oscillator frequency. The EUSART transmits and receives the LSB first.

Moreover, enabling the EUSART interrupt allows the LIN application program to run in the background while the microcontroller is performing other important tasks. This allows the MCU to perform other applications and improve time and power performance.

TIMER2 MODULE

The timing source can be selected from the available timer modules. Timer1, for example, is a 16-bit timer that increments on every instruction cycle when implemented with no prescaler and generates an interrupt on overflow. Timer2, on the other hand, is an 8-bit timer which generates an interrupt when a match between the timer and the period register occurs. Selection of timing source depends upon the availability of the module and the requirements of the specific application. The selected timer's period should always be set to 1 ms. This 1 ms time base is used to control the timing of each message frame in a schedule table.

The step-by-step instructions on how to configure the LIN library, as well as its associated peripherals are covered in the subsequent sections.

MCC LIN STACK DEMO

The demo programs presented in this application note use MCC 3.0. Refer to www.microchip.com/mcc for details on how to download and install the latest version of MCC.

The demo hardware for both the LIN master and the LIN slave stacks make use of the PICkit™ 28-pin LIN Demo Board (Part Number DM164130-3) and the LIN Serial Analyzer (Part Number APGDT001). A simplified setup is shown in Figure 6.

FIGURE 6: LIN DEMO SETUP

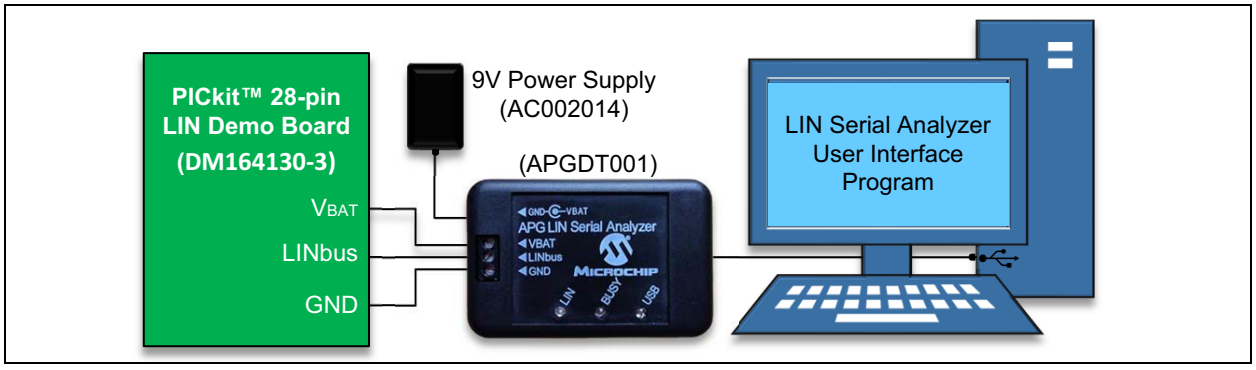
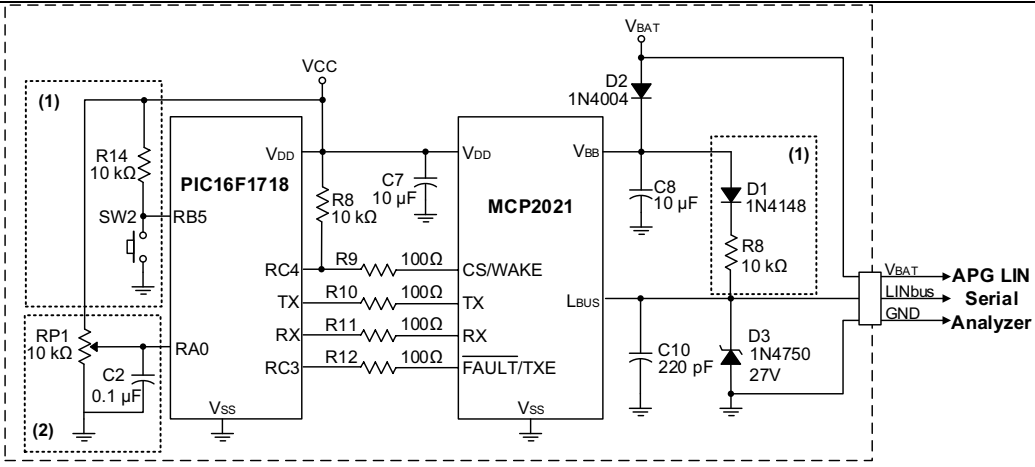


Figure 7 shows a section of the PICkit™ 28-pin LIN Demo Board implemented in the demo applications.

FIGURE 7: LIN DEMO SCHEMATIC



Note 1: Implemented in the LIN Master Demo only.

2: Implemented in the LIN Slave Demo only.

LIN MASTER STACK

Figure 8 shows the master node configuration table. Each row represents an individual frame. The content in each row shown are the default values. The user can manually edit the content and add more frames by adding rows. The schedule table in the code will be generated based on the parameters entered in this table. The columns in the table are defined in the following:

FIGURE 8: DEFAULT LIN MASTER TABLE

Signal	Type	Length	Time Out	Period
UNLOCK	Transmit	1	0	100
RSSI	Transmit	1	0	20
LFRX	Receive	8	100	0

Signal

In this column, the user provides a descriptive name to the frame ID. The top-most name (frame) will automatically be assigned with the ID 0. The MCC LIN stack converts the ID to PID for you. For example, ID 0 becomes PID 0x80. The incremental ID will be assigned as more frames are added. The scheduler code will start sending frames from top to bottom rows.

Note:	Signal in this table is referred to as a frame's name associated with a PID. This is not the same as the "signal" defined in the LIN specification. In the LIN specification, signals are referred to as the name of the bits and bytes in the Data Fields, not the PID.

Type

The frame can either be a Transmit or a Receive type. Configuring the frame as Transmit type means that the frame's data will be from the master. In other words, the response will be transmitted from the master itself following the header. Configuring the frame as Receive type means that the frame's data will be from a slave. In other words, after the master sends out the header, the master is expecting a response from a slave node.

Length

The length is the number of bytes in the data field. It should be a value between 1 to 8.

Time Out

Time out is the interval in millisecond (ms) between the end of the frame's header and the expected time when a node should finish transmitting the response. If a node takes longer than the expected time out to complete a response, the master stops transmitting the current frame and moves on to transmitting the next frame. Make sure the time out is at least longer than response space and the frame's response combined. Otherwise, the current frame will keep getting skipped and act as if it does not work.

Period

The period is the time interval in milliseconds (ms) between the start of the current frame to the start of the next frame. The period should be longer than the amount of time it takes to transmit the current frame. This allows more inter-frame time to account for the node to process the previous frame before it is ready to take in the next frame.

IMPLEMENTATION OF THE LIN MASTER STACK

This section presents a simple application of the LIN master library. A PICkit™ 28-pin LIN Demo Board is populated with a PIC16F1718 microcontroller and is configured as a master to communicate with a LIN Serial Analyzer configured as a slave. See **Section "MCC LIN Stack Demo"** for the proper hardware setup.

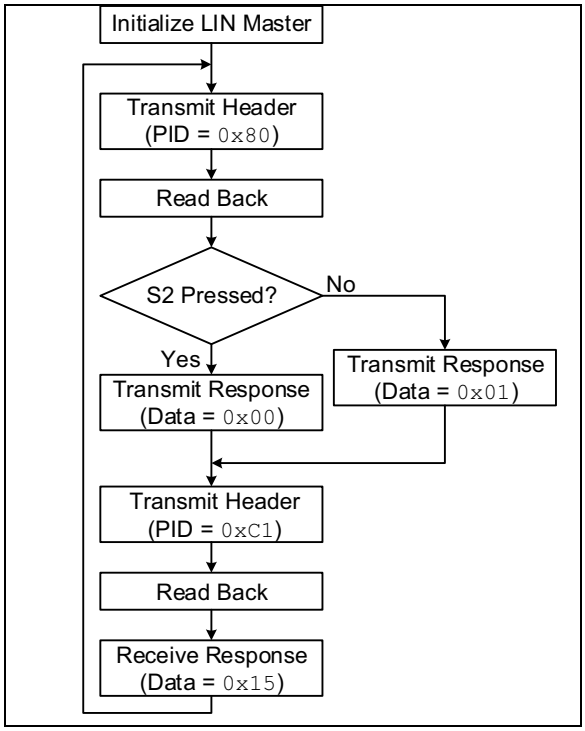
In the following demo example, only two unconditional frames will be entered on the schedule table GUI.

The first frame is a transmit type with one byte data. The master will transmit a frame with PID value 0x80 (ID = 0). The data will be from the master node itself signaling to the subscribers on the LIN cluster about the status of the master node's on-board switch, S2. When data is 0x00, S2 is pressed. When data is 0x01, S2 is unpressed.

The second frame is a receive type with one byte data. The master will transmit a frame with PID value 0xC1 (ID = 0x01). The node with the matching PID will simply transmit a response with fixed data value of 0x15.

Figure 9 illustrates a simplified demo flowchart.

FIGURE 9: MASTER DEMO FLOWCHART



MCC Configuration

The following will walk the user through the steps for setting up LIN frames, the required peripherals (system, timer, EUSART) and I/Os needed to enable the LIN transceiver. Also follow the screen captures in the **Section "MCC GUI Settings for Master Demo"**.

1. Navigate to *Tools > Embedded > MPLAB Code Configurator v3* to launch MCC version 3.
2. Under Project Resources, click **System** and select **System Module**.
3. Select the desired system clock settings. For the sample application, select Internal Oscillator (INTOSC) mode: I/O function on the CLK pin on the **Oscillator Select** menu. Set FOSC as the "System Clock" and 16 MHz_HF as the "Internal Clock". Uncheck the **PLL Enabled** checkbox. See Figure 10.
4. Disable Watchdog Timer.
5. Under Device Resources, select *Libraries > LIN > LIN Master*.
6. The LIN Master Table GUI shown in Figure 8 will appear.
7. To add a frame, click **Add**. An additional row will appear below the "Signal-Type-Length-Timeout-Period" columns.
8. Edit the fields as shown in Figure 11. Each column is defined in **Section "LIN Master Stack"**.
9. Select Timer2 as timer. Since PIC16F1718 has only one EUSART, only one option will appear in the EUSART selection tab.
10. Click the **Notifications** tab. This tab lists all the additional configurations that must be done for the LIN stack to become fully functional. Take note of the "HINT" type notification as this will be used later after the MCC settings are generated.
11. One of the notifications describes that the LIN module uses Timer2. Hence, Timer2 should be configured. To do this, navigate to *Device Resources > Peripherals > Timer > TMR2*.
12. Check both **Enable Timer** and **Enable Timer Interrupt**.
13. Set the Timer Period to 1 ms. When using a high-frequency system clock, the prescaler and postscaler ratios might need some adjustments.
14. Another notification tells that the LIN module uses EUSART. Hence, the EUSART module needs to be configured. To do this, navigate to *Device Resources > Peripherals > EUSART > EUSART*.
15. In the MCC EUSART GUI, set the mode to asynchronous. Check **Enable EUSART**, **Enable Transmit** and **Enable EUSART Interrupts**. Set both Transmission bits and Reception bits to 8-bit, and Clock Polarity to Non-Inverted.

16. Setup the baud rate. The baud rate should be a value between 1 and 20000. For the sample application, a baud rate of 9600 is used.
17. Set the corresponding EUSART TX and RX pins on the “Table View” at the lower right part of the MCC window. For this example, RC6 and RC7 are assigned as the TX and RX pins, respectively.
18. After setting up both the Timer2 and EUSART modules, configure the I/O ports to enable the connected LIN transceiver (MCP2021). Note that different transceivers may require different set up. Refer to the particular transceiver data sheet. For a 28-pin LIN Demo Board, the CS/WAKE and FAULT/TXE pins of the transceiver are connected to the RC4 and RC3 pins, respectively. CS/WAKE should be configured as an output and set high, while FAULT/TXE should be an input with weak pull-ups enabled.
19. Other peripherals can now be configured depending upon the specific application. For the sample application, only an external switch connected to RB5 is used. It is configured as a digital input pin. The pin module GUI is shown in Figure 15.
20. Click **Generate**. When generating the MCC code for the first time, a pop-up window will appear asking for the configuration settings to be saved in a .mc3 file format. Enter the desired file name then click **Save**. The configuration settings are summarized in **Section “MCC GUI Settings for Master Demo”**.
21. The configuration settings are now generated. However, the configuration process is not done yet. The “HINT” type notification, “Add `LIN_handler()` in the `main.c` while loop” should be implemented on the generated code.
22. Uncomment `INTERRUPT_GlobalInterruptEnable()` and `INTERRUPT_PeripheralInterruptEnable()` in `main.c`.
23. The LIN Master driver is now set. For the sample application, the `while (1)` loop of `main.c` is shown in Example 1.

EXAMPLE 1: MASTER DEMO CODE SNIPPET

```
while (1)
        {
            // Add your application code
            LIN_handler();

            if (PORTBbits.RB5 == HIGH){
                Master_Switch_Data[0] = 0x01;
            }
            else{
                Master_Switch_Data[0] = 0x00;
            }
        }
```

MCC GUI Settings for Master Demo

The following figures show the MCC Composer Area view of the different peripheral modules settings implemented in the MCC LIN Stack Master Demo.

FIGURE 10: SYSTEM MODULE SETTINGS

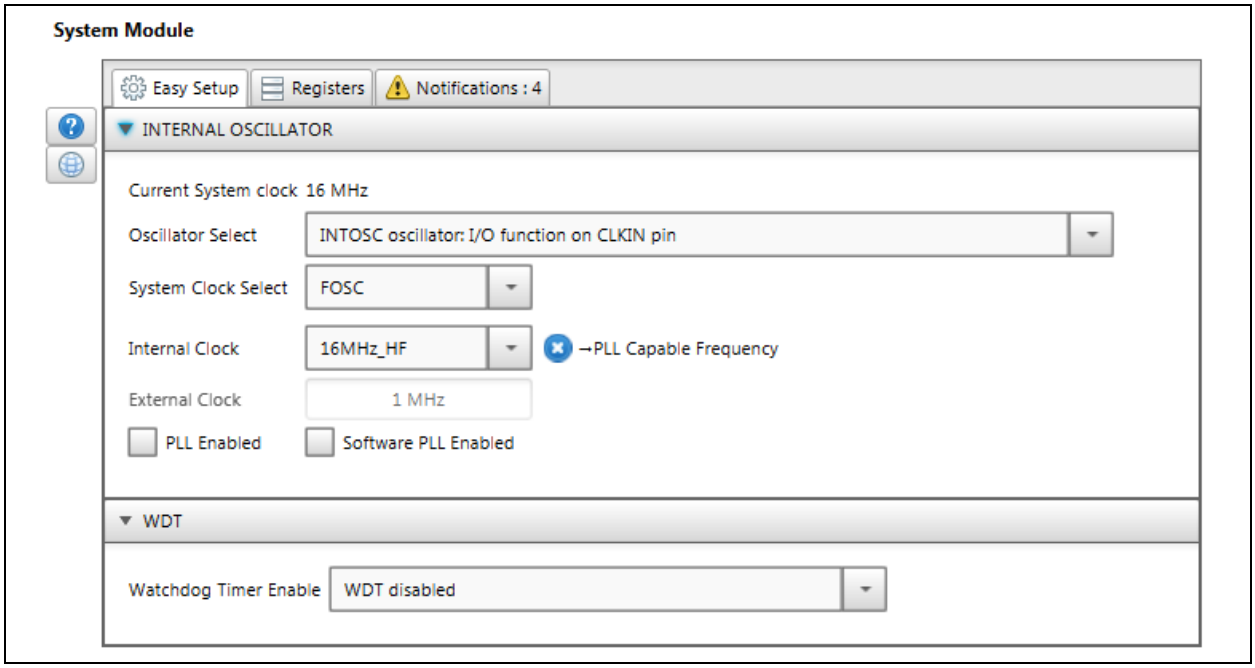


FIGURE 11: LIN MASTER SETTINGS

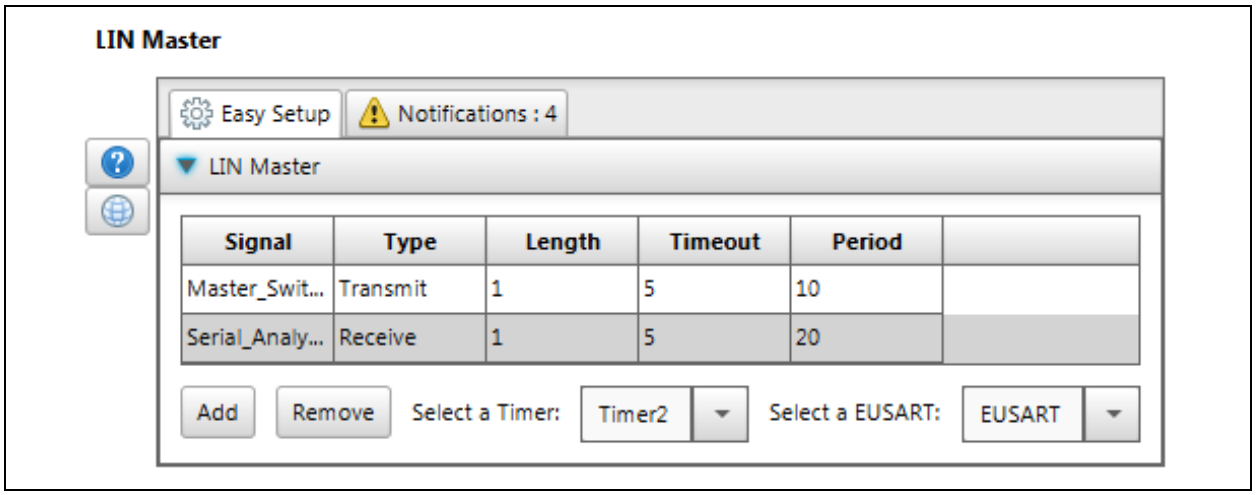


FIGURE 12: LIN MASTER NOTIFICATIONS

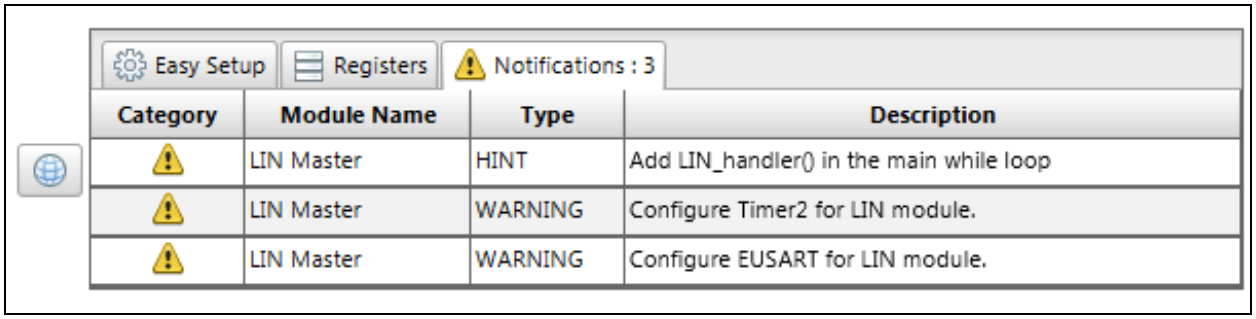


FIGURE 13: ASSOCIATED MODULE SETTINGS – EUSART

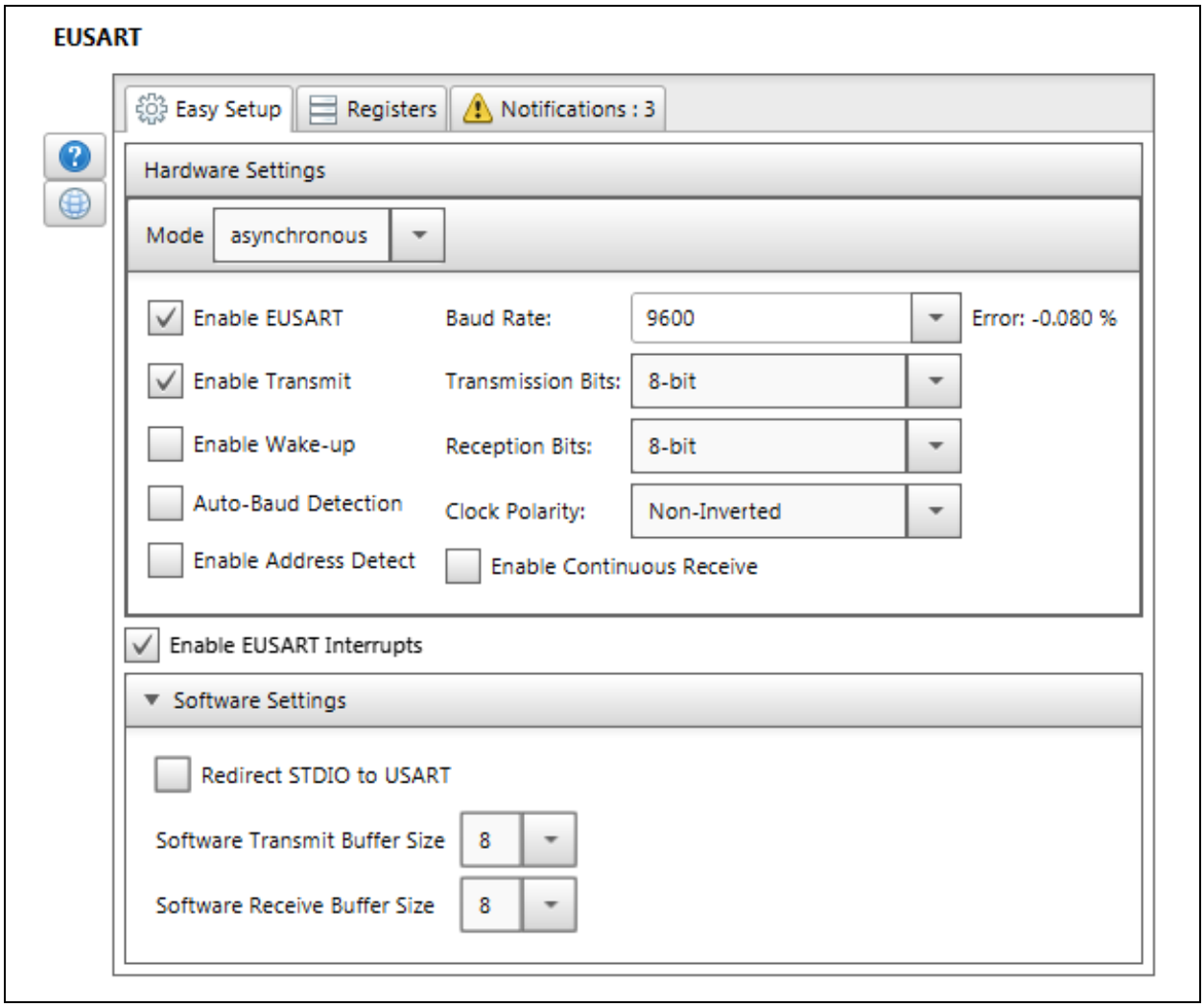


FIGURE 14: ASSOCIATED MODULE SETTINGS – TMR2

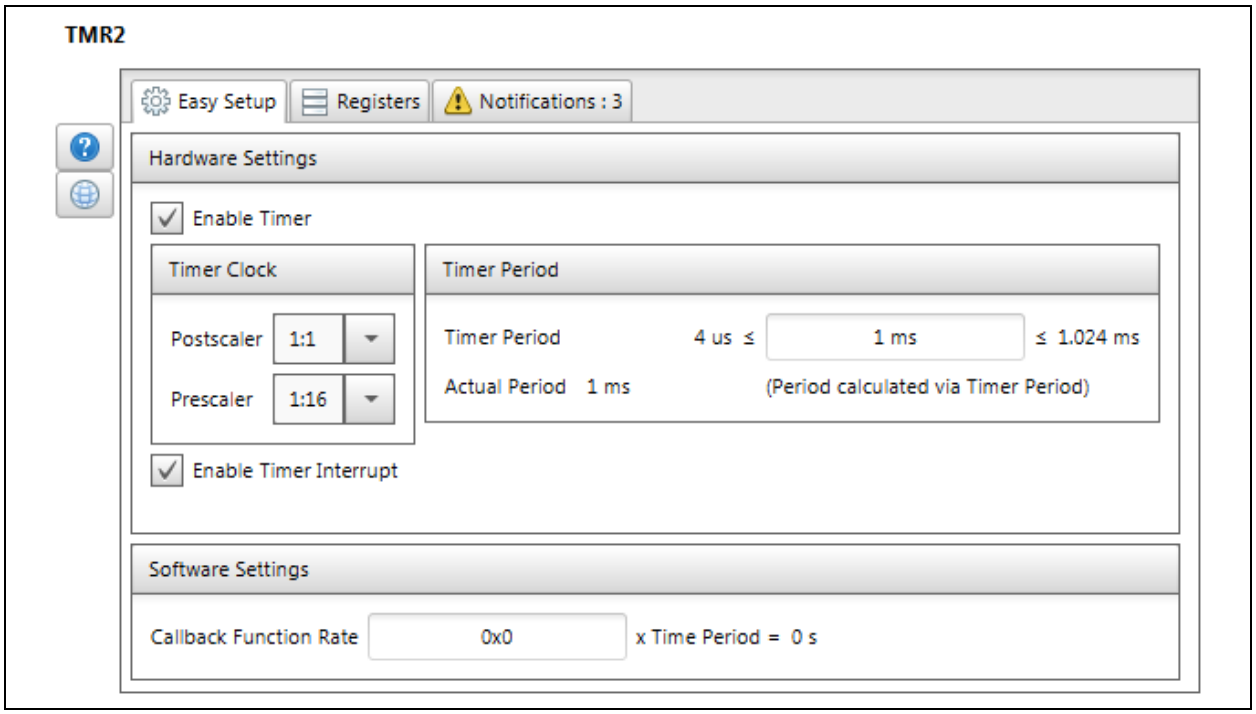
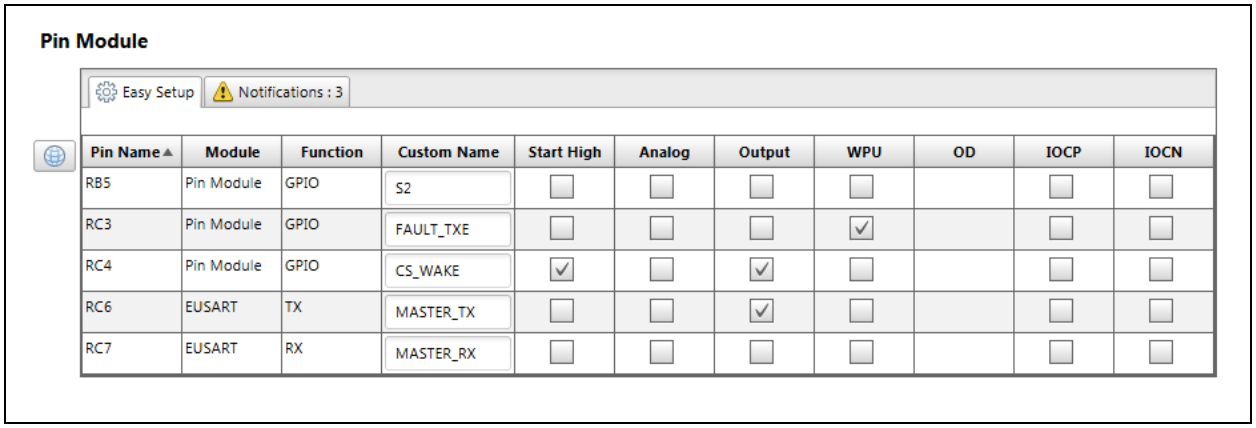


FIGURE 15: ASSOCIATED MODULE SETTINGS – PIN MODULE

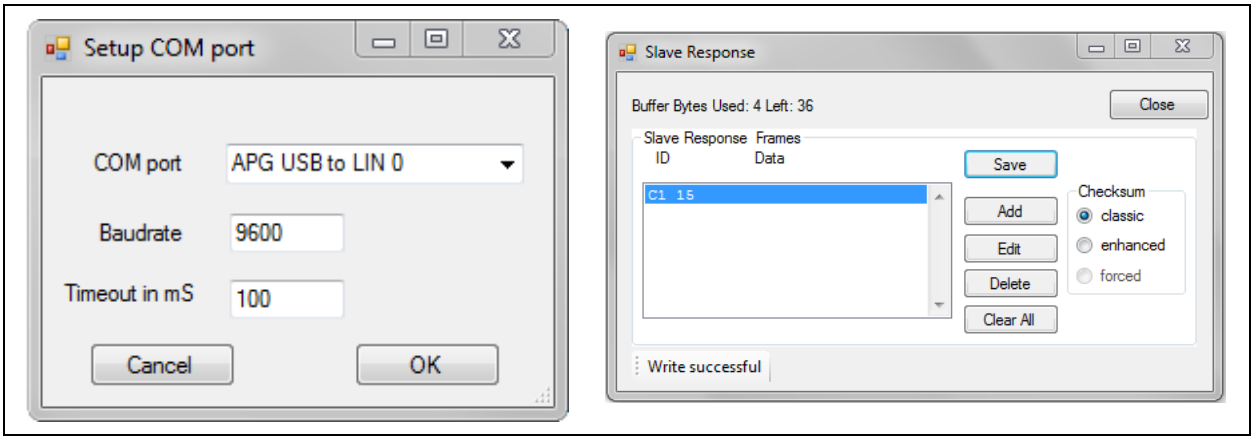


Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOCP	IOCN
RB5	Pin Module	GPIO	S2	☐	☐	☐	☐		☐	☐
RC3	Pin Module	GPIO	FAULT_TXE	☐	☐	☐	☑		☐	☐
RC4	Pin Module	GPIO	CS_WAKE	☑	☐	☑	☐		☐	☐
RC6	EUSART	TX	MASTER_TX	☐	☐	☑	☐		☐	☐
RC7	EUSART	RX	MASTER_RX	☐	☐	☐	☐		☐	☐

Slave Settings

The settings for the LIN Serial Analyzer are shown in Figure 16. The implementation of the LIN Serial Analyzer will not be elaborated in this application note. Refer to www.microchip.com for more info.

FIGURE 16: LIN SERIAL ANALYZER SETTINGS



Master Demo Results

Figure 17 and Figure 18 show the Serial Analyzer and the Logic Analyzer readings for the master demo application, respectively. When S2 is pressed (RB5 is low), the master transmits `0x00`. Conversely, when S2 is not pressed (RB5 is high), the master transmits `0x01`.

FIGURE 17: LIN SERIAL ANALYZER DISPLAY FOR MASTER DEMO

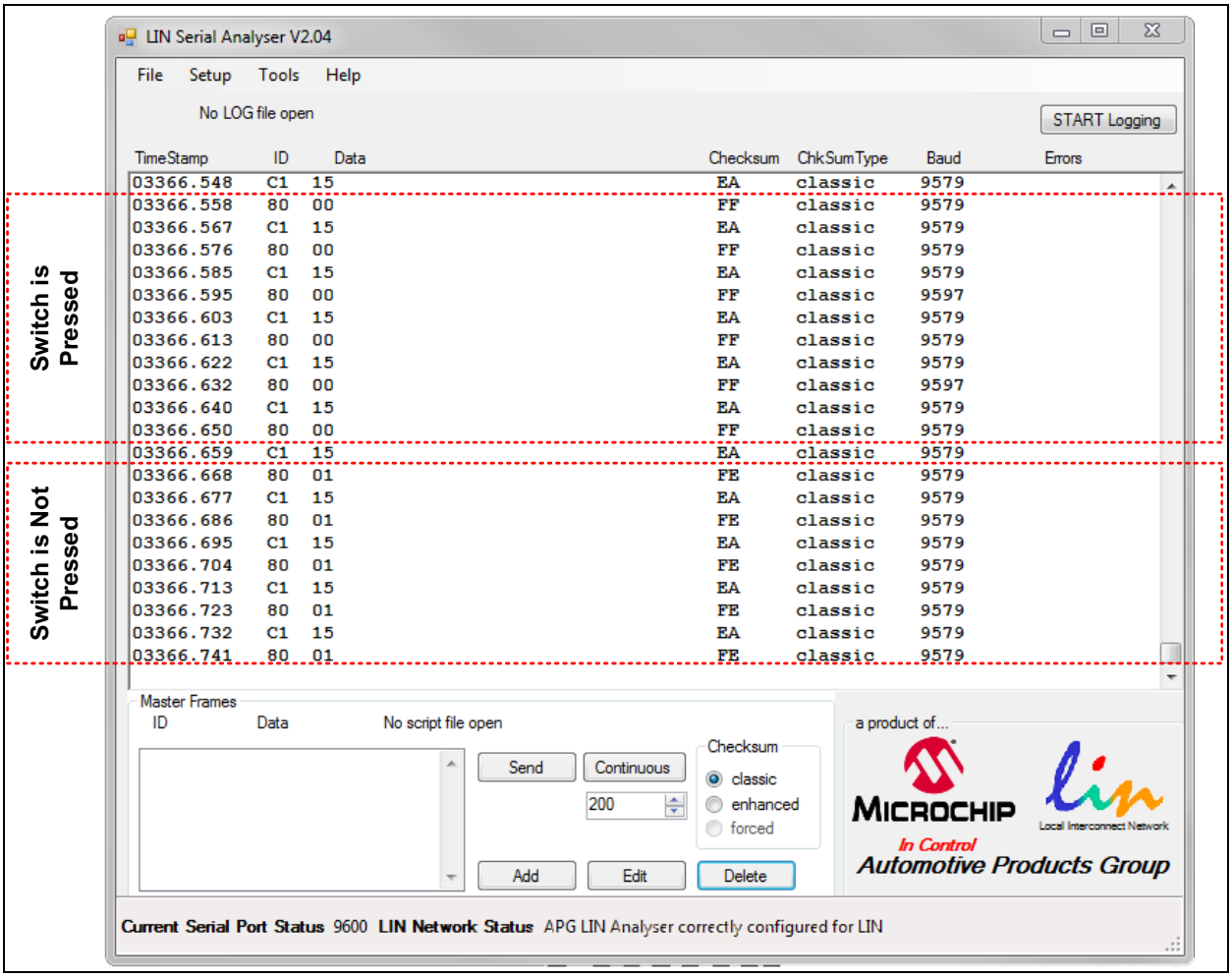
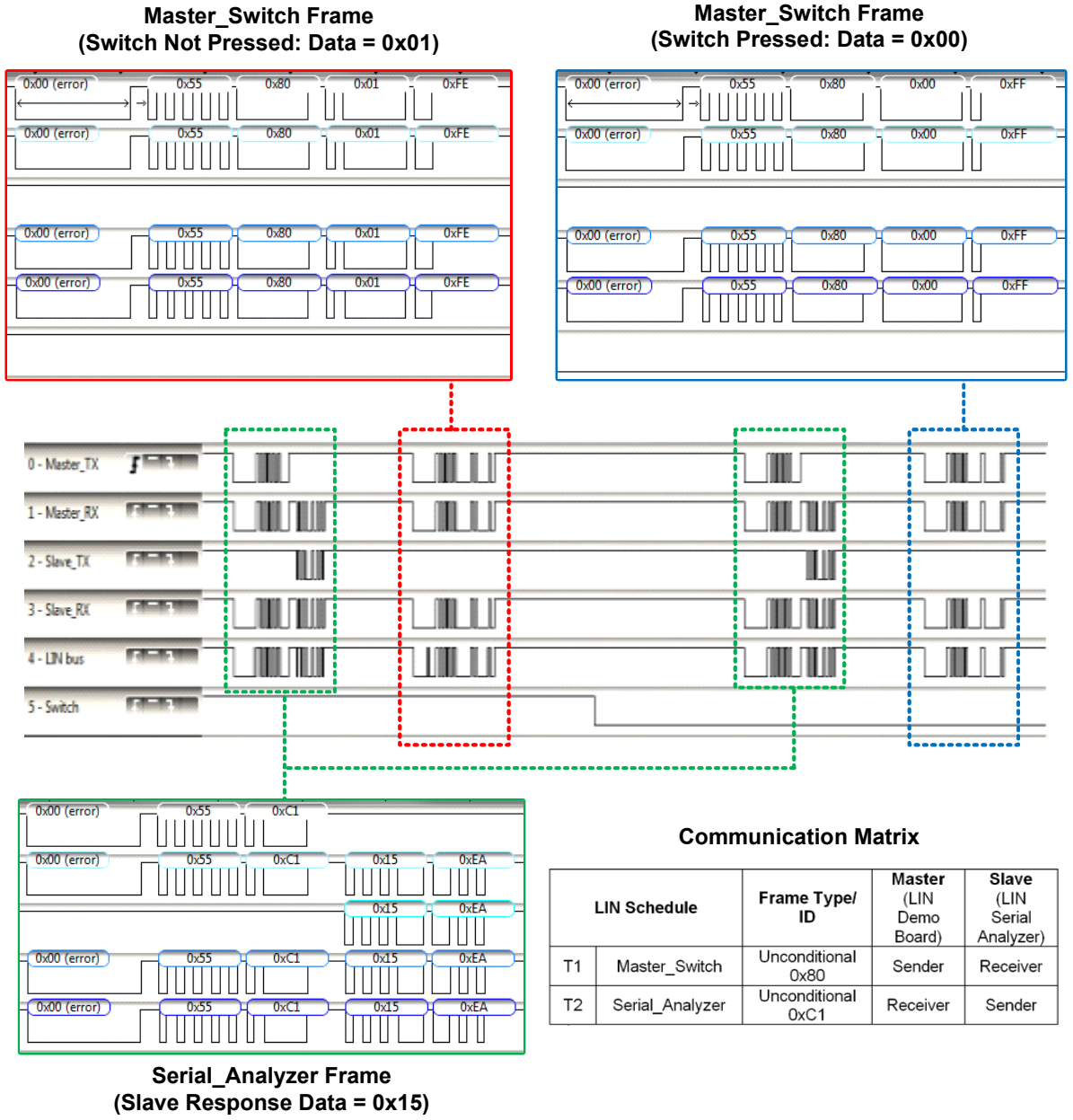


FIGURE 18: LOGIC ANALYZER DISPLAY FOR MASTER DEMO



Communication Matrix

LIN Schedule		Frame Type/ ID	Master (LIN Demo Board)	Slave (LIN Serial Analyzer)
T1	Master_Switch	Unconditional 0x80	Sender	Receiver
T2	Serial_Analyzer	Unconditional 0xC1	Receiver	Sender

LIN SLAVE STACK

The default slave node is shown in Figure 19. Unlike the master table, the slave table is only composed of three columns for signal, type and length.

FIGURE 19: DEFAULT LIN SLAVE TABLE

Signal	Type	Length
UNLOCK	Receive	1
RSSI	Receive	1
LFRX	Transmit	8

Signal

If the MCC LIN master stack is implemented on the master node connected on the same bus with this slave, all the frames listed in the master table should also be entered in the same order on the LIN slave table. This is due to the PID assignment system implemented in the MCC LIN Stack. Using a different master may not require this but the user should take note of the specific PID for the given frame (see **Section "MCC LIN Stack PID Assignment"**).

Type

Configuring the frame as Transmit type means the slave is going to transmit the response after receiving the header with the matching PID from the master. Configuring frame as Receive type means the slave is a subscriber to the data in the Data Field of the corresponding frame (PID).

Note:	In a LIN cluster, we may have multiple subscribers (receivers or listeners) to a master's frame. However, only one response to a frame header (only one publisher) is allowed at a given time. For example, if the master configures frame ID 0 to be a transmit type, it is allowed to have multiple slave nodes to have the same frame ID 0 to be as receive type. In another example, if the master configures frame ID 0 to be a receive type, only one slave node is allowed to configure as transmit type to respond to the frame ID 0.

Length

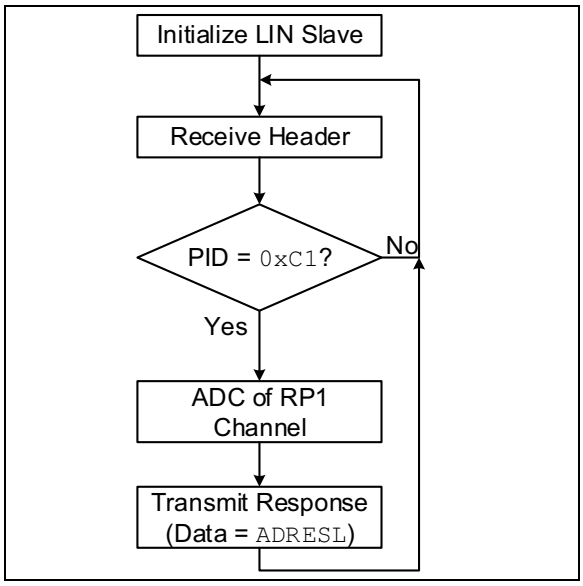
The length is the number of bytes in a frame's data field. The length should be the same on both master node or slave nodes for a corresponding frame PID. For example, if PID 0x80 frame has length of two data bytes, so should the slave.

IMPLEMENTATION OF THE LIN SLAVE STACK

This section presents a sample application of the MCC LIN slave driver using a LIN Serial Analyzer as the master and a PICkit™ 28-pin LIN Demo Board populated with a PIC16F1718 microcontroller as slave. This demo makes use of the on-board potentiometer RP1 and the on-chip Analog-to-Digital Converter (ADC) module of the PIC16F1718. See **Section "MCC LIN Stack Demo"** for the hardware setup.

A simplified flowchart for the slave demo is shown in Figure 20. The slave will only transmit a response if a PID of `0xC1` is received. It will send a data byte with a value equal to the eight LSBs of the ADC reading of the potentiometer input.

FIGURE 20: SLAVE DEMO FLOWCHART



MCC Configuration

To configure the MCC LIN slave driver, follow these steps:

1. Navigate to *Tools > Embedded > MPLAB Code Configurator v3* to launch MCC version 3.
2. Under "Project Resources", click **System** and select **System Module**.
3. Select the desired system clock settings. For the sample application, select Internal Oscillator (INTOSC) mode: I/O function on the CLK pin on the **Oscillator Select** menu. Set FOSC as the "System Clock" and 16 MHz_HF as the "Internal Clock." Uncheck the **PLL Enabled** checkbox. See Figure 10.
4. Disable Watchdog Timer.
5. Under "Device Resources", select *Libraries > LIN > LIN Slave*.
6. The LIN Slave Table GUI shown in Figure 19 will appear.
7. To add a frame, click the **Easy Setup** tab then **Add**. An additional row will appear below the "Signal-Type-Length" columns.
8. Edit the fields as shown in Figure 21. Each column is defined in **Section "LIN Slave Stack"**.
9. Select Timer2 as timer. Since PIC16F1718 has only one EUSART, only one option will appear in the EUSART selection tab.

10. Click the **Notifications** tab. This tab lists all the additional configurations that must be done for the LIN Stack to become fully functional. Take note of the “HINT” type notification, as this will be used later after the MCC settings are generated.
11. One of the notifications describes that the LIN module uses Timer2. Hence, Timer2 should be configured. To do this, navigate to *Device Resources > Peripherals > Timer > TMR2*.
12. Check both **Enable Timer** and **Enable Timer Interrupt**.
13. Set the Timer Period to 1 ms. When using high frequency system clock, the prescaler and postscaler ratios might need some adjustments.
14. Another notification tells that the LIN module uses EUSART. Hence, the EUSART module needs to be configured. To do this, navigate to *Device Resources > Peripherals > EUSART > EUSART*.
15. In the MCC EUSART GUI, set the mode to asynchronous. Check **Enable EUSART**, **Enable Transmit**, **Enable Continuous Receive** and **Enable EUSART Interrupts**. Set both Transmission bits and Reception bits to 8-bit, and Clock Polarity to Non-Inverted.
16. Setup the baud rate. The baud rate should be a value between 1 and 20000. For the sample application, a baud rate of 9600 is used.
17. Set the corresponding EUSART TX and RX pins on the “Table View” at the lower right part of the MCC window. For this example, RC6 and RC7 are assigned as the TX and RX pins, respectively.
18. After setting up both the Timer2 and EUSART modules, configure the I/O ports to enable the connected LIN transceiver (MCP2021). Note that different transceivers may require different setup. Refer to the particular transceiver data sheet. For a 28-pin LIN Demo Board, the CS/WAKE and FAULT/TXE pins of the transceiver are connected to RC4 and RC3, respectively. CS/WAKE should be configured as an output and set high, while FAULT/TXE should be an input with weak pull-ups enabled. The pin module GUI is shown in Figure 25.
19. Other peripherals can now be configured depending upon the specific application. The sample application uses an analog input; therefore, the ADC module must be setup.
20. To configure the ADC, go to *Device Resources > Peripherals > ADC > ADC*. In “Table View”, select RA0 as an ADC Input pin. AN0 will then be added automatically to the “Selected Channels” list in the “Composer Area”. Use “RP1” as a custom name for this channel. Figure 24 details the complete ADC settings.
21. Click **Generate**. When generating the MCC code for the first time, a pop-up window will appear asking for the configuration settings to be saved in a .mc3 file format. Enter the desired file name then click **Save**. The configuration settings are summarized in **Section “MCC GUI Settings for Slave Demo”**.
22. The configuration settings are now generated. However, the configuration process is not done yet. The “HINT” type notification, “Add `LIN_handler()` in the `main.c` while loop”, should be implemented on the generated code.
23. Uncomment `INTERRUPT_GlobalInterruptEnable()` and `INTERRUPT_PeripheralInterruptEnable()` in `main.c`.
24. The LIN Slave driver is now set. For the sample application, `../adc.h` should be included in `lin_app.c`, and `processLIN()` is modified as shown in Example 2.

EXAMPLE 2: SLAVE DEMO CODE SNIPPET

```
void processLIN(void){
    uint8_t tempRxData[8];
    uint8_t cmd;
    cmd = LIN_getPacket(tempRxData);
    switch(cmd){
        case Serial_Analyzer:
            break;
        case Slave_Pot:
            Slave_Pot_Data[0] = ADC_GetConversion(RP1);
            break;
        default:
            break;
   }
```

MCC GUI Settings for Slave Demo

The following figures show the MCC Composer Area view of the different peripheral modules settings implemented in the MCC LIN Stack Slave Demo.

Note: System Module and TMR2 Settings are the same as shown in Figure 10 and Figure 14.

FIGURE 21: LIN SLAVE SETTINGS

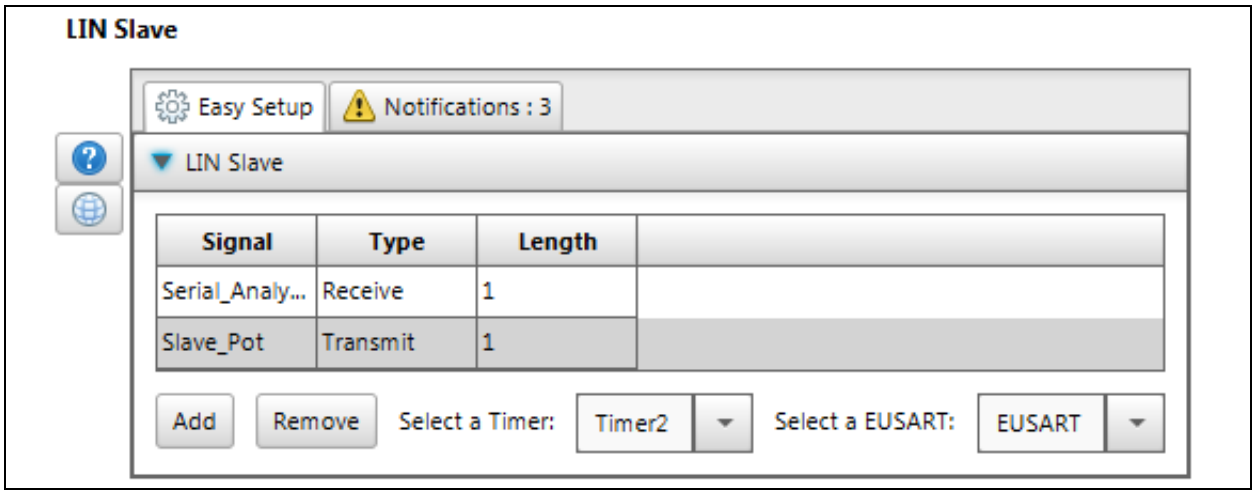


FIGURE 22: LIN SLAVE NOTIFICATIONS

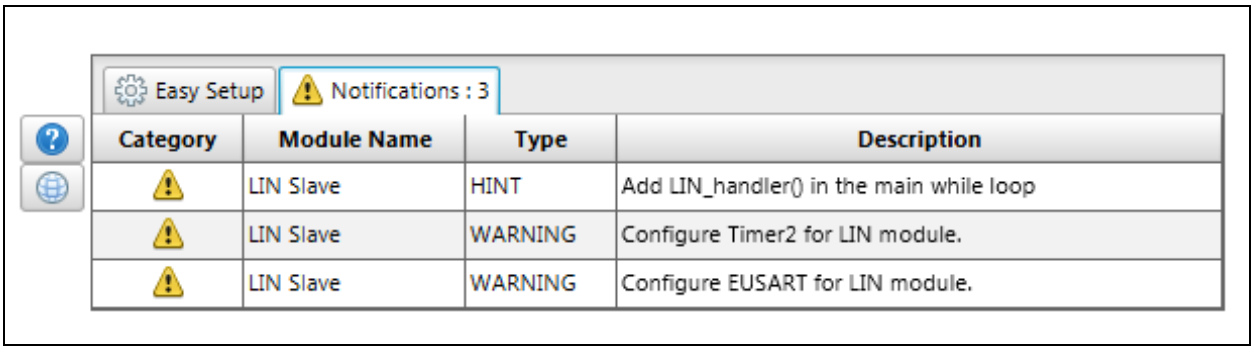


FIGURE 23: ASSOCIATED MODULE SETTINGS – EUSART

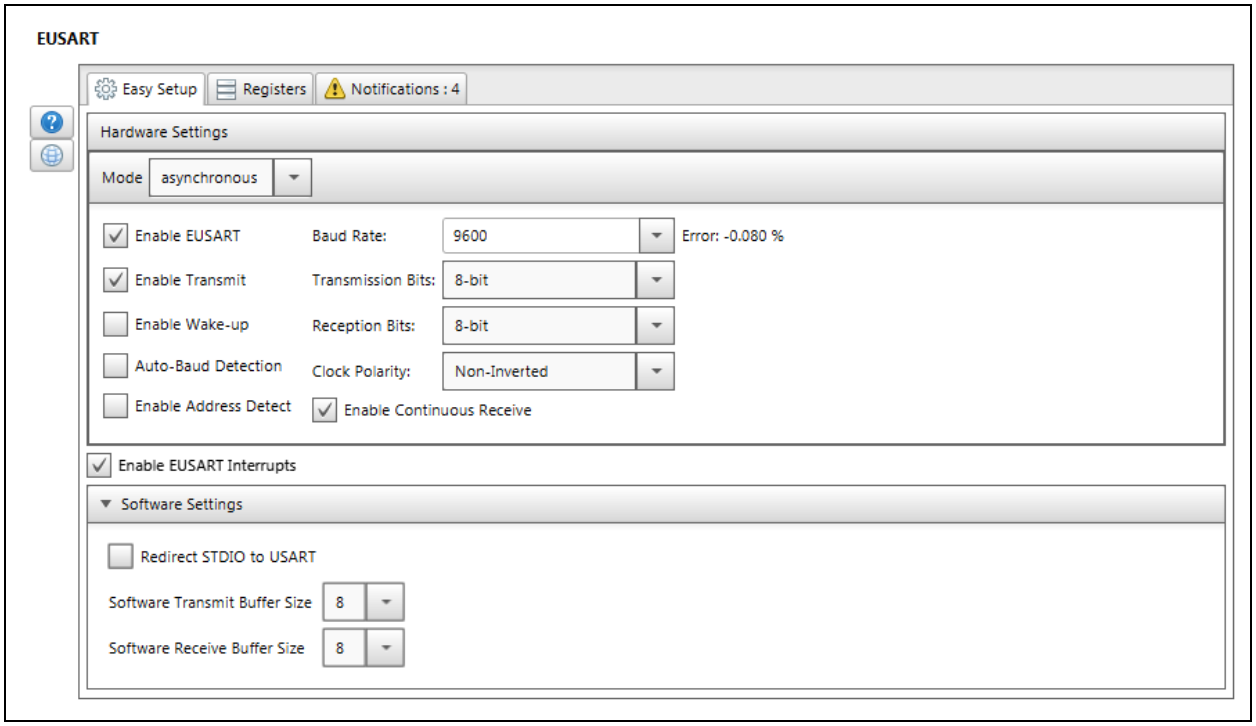


FIGURE 24: ASSOCIATED MODULE SETTINGS – ADC

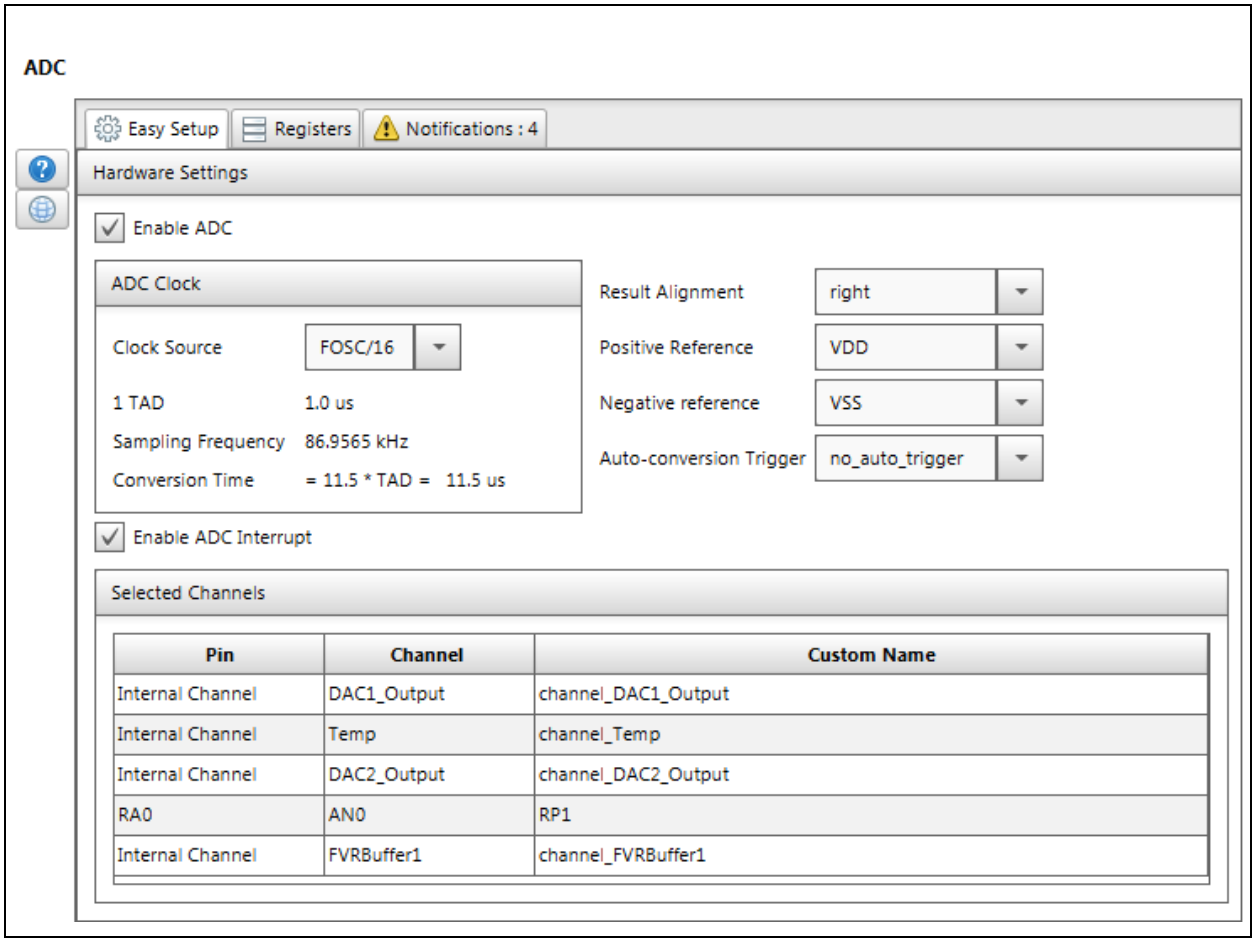
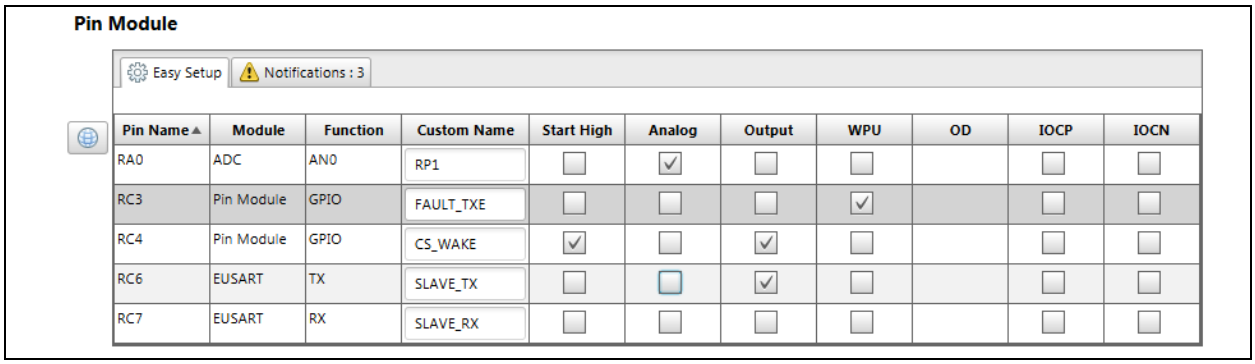


FIGURE 25: ASSOCIATED MODULE SETTINGS – PIN MODULE



Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOCP	IOCN
RA0	ADC	AN0	RP1	☐	☑	☐	☐		☐	☐
RC3	Pin Module	GPIO	FAULT_TXE	☐	☐	☐	☑		☐	☐
RC4	Pin Module	GPIO	CS_WAKE	☑	☐	☑	☐		☐	☐
RC6	EUSART	TX	SLAVE_TX	☐	☐	☑	☐		☐	☐
RC7	EUSART	RX	SLAVE_RX	☐	☐	☐	☐		☐	☐

Slave Demo Results

Figure 26 and Figure 27 show the Serial Analyzer and the Logic Analyzer readings for the slave demo, respectively. The master periodically sends data with a hex value of `0x07`, while the slave responds with the eight LSBs of the ADC reading from the potentiometer input every time the PID of `0xC1` is broadcasted by the master.

FIGURE 26: LIN SERIAL ANALYZER DISPLAY FOR SLAVE DEMO

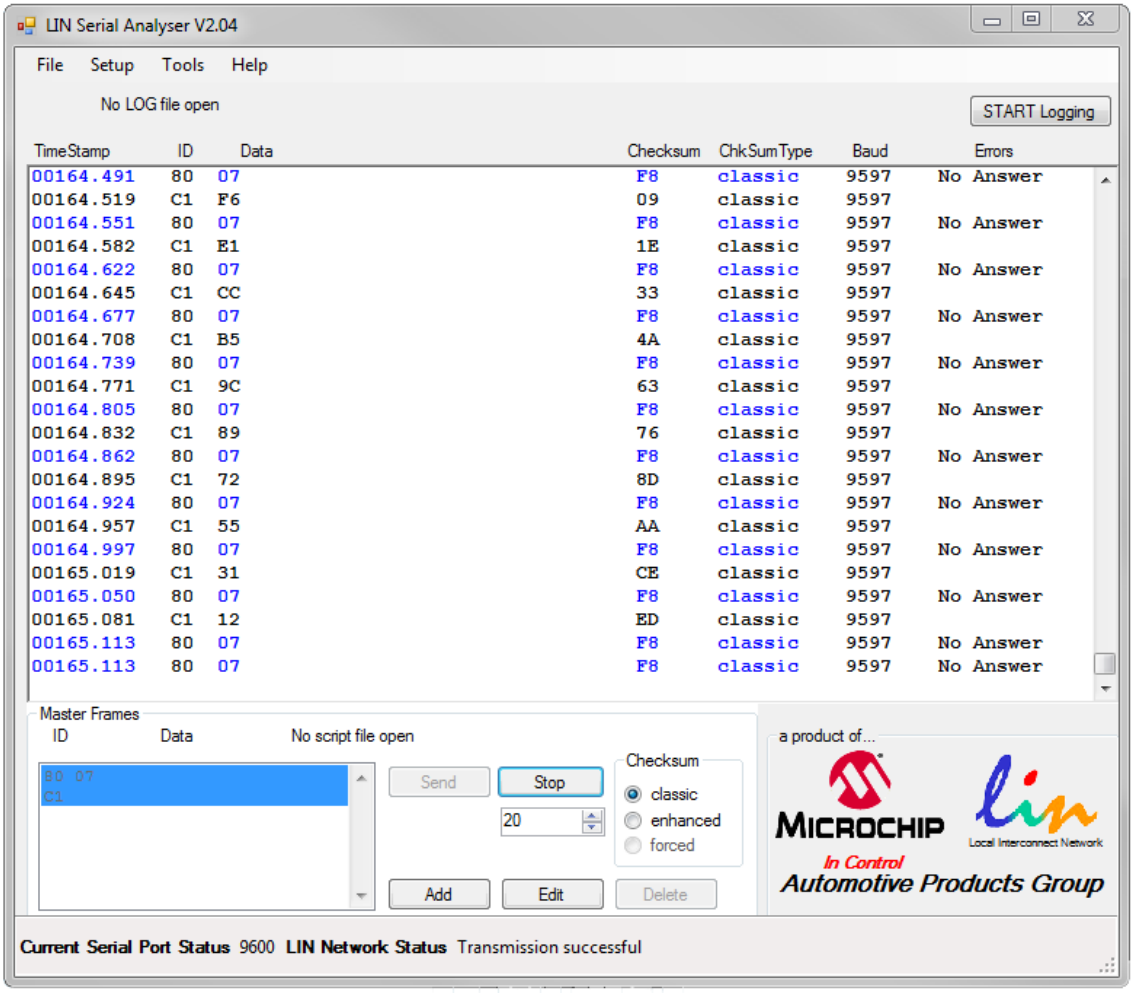
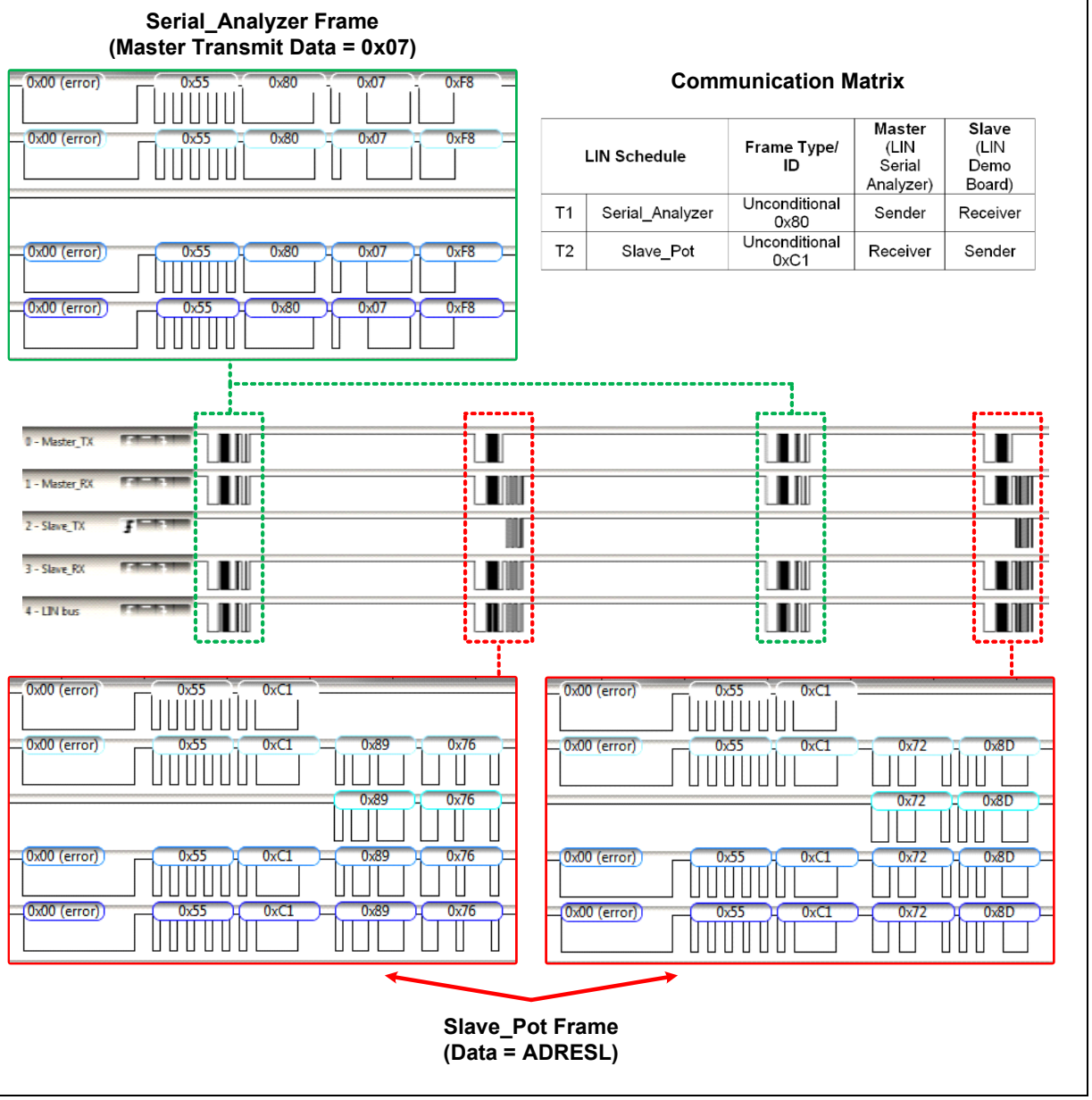


FIGURE 27: LOGIC ANALYZER DISPLAY FOR SLAVE DEMO



CONCLUSION

This application note demonstrates how to rapid prototype LIN-related embedded applications using Microchip's MPLAB Code Configurator (MCC). LIN code was generated for master and slave nodes, and the user learned how to configure peripherals such as I/O, EUSART, ADC and the system clock. Also demonstrated are actual LIN transmissions, which should provide the user with a basic understanding of the protocol. The next step is to move on to the production level by further understanding the LIN specification. Again, the MCC LIN stack is designed for prototyping purposes only and does not support the full LIN specification. When full automotive or production drivers are needed, the designer should consult with Microchip's design or third-party partners such as IHR.

REFERENCES

1. LIN Specification Package Revision 2.2A.

APPENDIX A: FRAME PID LOOK-UP TABLE

TABLE A-1: PID LOOK-UP TABLE FOR SIGNAL-CARRYING FRAMES

P1	P0	ID[5…0] PID<5:0>		PID Field PID<7:0>	
PID<7>	PID<6>				
¬(ID1⊕ID3⊕ID4⊕ID5)	ID0⊕ID2⊕ID3⊕ID4	Dec	Hex	Dec	Hex
1	0	**0**	0x00	128	0x80
1	1	**1**	0x01	193	0xC1
0	1	**2**	0x02	66	0x42
0	0	**3**	0x03	3	0x03
1	1	**4**	0x04	196	0xC4
1	1	**5**	0x05	133	0x85
0	0	**6**	0x06	6	0x06
0	0	**7**	0x07	71	0x47
0	0	**8**	0x08	8	0x08
0	0	**9**	0x09	73	0x49
1	1	**10**	0x0A	202	0xCA
1	1	**11**	0x0B	139	0x8B
0	0	**12**	0x0C	76	0x4C
0	0	**13**	0x0D	13	0x0D
1	1	**14**	0x0E	142	0x8E
1	1	**15**	0x0F	207	0xCF
0	0	**16**	0x10	80	0x50
0	0	**17**	0x11	17	0x11
1	1	**18**	0x12	146	0x92
1	1	**19**	0x13	211	0xD3
0	0	**20**	0x14	20	0x14
0	0	**21**	0x15	85	0x55
1	1	**22**	0x16	214	0xD6
1	1	**23**	0x17	151	0x97
1	1	**24**	0x18	216	0xD8
1	1	**25**	0x19	153	0x99
0	0	**26**	0x1A	26	0x1A
0	0	**27**	0x1B	91	0x5B
1	1	**28**	0x1C	156	0x9C
1	1	**29**	0x1D	221	0xDD
0	0	**30**	0x1E	94	0x5E
0	0	**31**	0x1F	31	0x1F
0	0	**32**	0x20	32	0x20
0	0	**33**	0x21	97	0x61
1	1	**34**	0x22	226	0xE2
1	1	**35**	0x23	163	0xA3
0	0	**36**	0x24	100	0x64

TABLE A-1: PID LOOK-UP TABLE FOR SIGNAL-CARRYING FRAMES (CONTINUED)

P1	P0	ID[5…0] PID<5:0>		PID Field PID<7:0>	
PID<7>	PID<6>				
⌐(ID1⊕ID3⊕ID4⊕ID5)	ID0⊕ID2⊕ID3⊕ID4	Dec	Hex	Dec	Hex
0	0	**37**	0x25	37	0x25
1	0	**38**	0x26	166	0xA6
1	1	**39**	0x27	231	0xE7
1	0	**40**	0x28	168	0xA8
1	1	**41**	0x29	233	0xE9
0	1	**42**	0x2A	106	0x6A
0	0	**43**	0x2B	43	0x2B
1	1	**44**	0x2C	236	0xEC
1	0	**45**	0x2D	173	0xAD
0	0	**46**	0x2E	46	0x2E
0	1	**47**	0x2F	111	0x6F
1	1	**48**	0x30	240	0xF0
1	0	**49**	0x31	177	0xB1
0	1	**50**	0x32	50	0x32
0	1	**51**	0x33	115	0x73
1	1	**52**	0x34	180	0xB4
1	0	**53**	0x35	245	0xF5
0	1	**54**	0x36	118	0x76
0	0	**55**	0x37	55	0x37
0	1	**56**	0x38	120	0x78
0	0	**57**	0x39	57	0x39
1	0	**58**	0x3A	186	0xBA
1	1	**59**	0x3B	251	0xFB

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip 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."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

ISBN: 978-1-5224-0321-0

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110

Canada - Toronto
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7828

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Dusseldorf
Tel: 49-2129-3766400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Venice
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Poland - Warsaw
Tel: 48-22-3325737

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

07/14/15