Freescale Semiconductor
Application Note

Document Number: AN2961
Rev. 1, 05/2007

Software Drivers for MC33696

by: Petr Gargulak
John Logan
Graham Williamson
TSPG, Freescale Semiconductor, Inc.

1 Introduction

This application note describes the software device driver written for the MC33696 transceiver IC (Echo) and how it can be integrated and used in an application on the HCS08 family of MCUs. The device driver allows customer applications to use all features of the Echo platform while remaining easily configurable.

The driver may be configured to run on any HCS08 MCU. You can specify different timer channels and I/O pins on the MCU, allowing flexibility in system design, modulation techniques, data rates, and clock sources. All other Echo features are fully configurable. Full source code and example applications are available on www.freescale.com.

The driver is designed to allow communication in Echo-only networks and also with the Tango3 transmitter and Romeo2 receiver devices. Device drivers for Tango and Romeo are available on www.freescale.com and documented in application note

Contents

AN2707. For more information on configuring the device drivers to allow communication between Tango, Romeo, and Echo devices, see Section 14, “Communicating with Romeo2 and Tango3 Devices.”

The driver is supplied as two software files—Echo.C and Echo.H. Configure the driver by making selections in the Echo.H file.

Features of the driver include:

- Configurable statically for modulation type, frequency band, data rate, device ID, and header
- Supports bank switching—communicating on two RF channels
- Transmits and receives variable length messages with up to 127 bytes data
- Automatic checksum based error detection for each message
- Controls receiver on/off cycling using Echo’s internal strobe oscillator or directly via an I/O pin
- Provides dynamic control of the precise frequencies used in the selected band
- Provides dynamic control of Tx power and Rx sensitivity
- Supports received signal strength indicator (RSSI) measurement during message reception
- Compatible with Tango3/Romeo2 communications
- Compatible with 8-bit Freescale MCU families HC08 and HCS08

For more information, refer to the Echo device data sheet (available at http://www.freescale.com).

2 Communication Concept

Echo can support communication in the 304 MHz, 315 MHz, 434 MHz, 868 MHz, and 916 MHz bands with data rates up to 22 kbps. Within each frequency band, you can choose the local oscillator and carrier frequencies used.

Multiple transmitters (Tango3), receivers (Romeo2), and transceivers (Echo) may be present in a network and can be individually selected by assigning each receiving device a unique ID. Groups of devices may be formed by assigning a number of devices the same ID, supporting multicast or broadcast communication.

Echo devices can be used in applications requiring two-way communication or flexible allocation of transmitter and receiver roles, which can be changed dynamically. Public key and certification authentication schemes are examples of applications that demand two-way communication.

3 Message Format

The driver supports sending messages in two formats that differ in the way that the ID is sent: spaced preamble/ID sequences followed by the message or repeated consecutive IDs, with no spacing, followed by the message. The driver also supports tone signaling: the application may use the ID repeat mode with an ID of all 1 or all 0. These formats are shown in Figure 1.

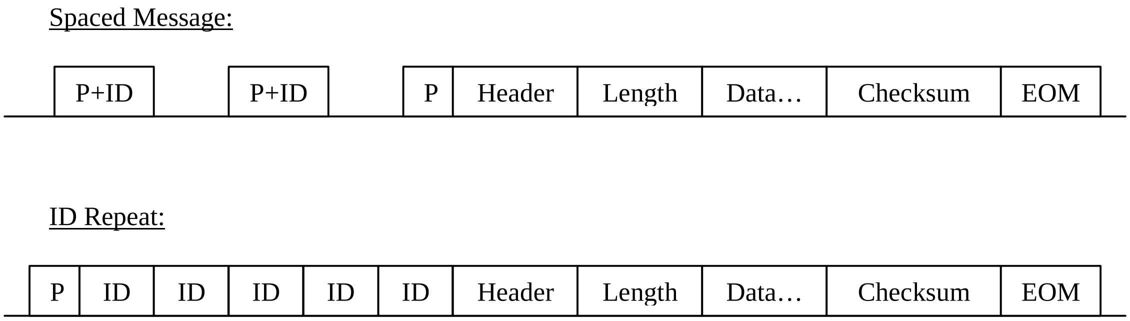


Figure 1. Message Formats

In both cases, the data manager hardware on Echo is employed to process received signals and detect the ID and header portions. The length, data, and checksum fields only are returned to the MCU on the SPI.

Table 1 provides a description of each field of a message.

Table 1. Message Format Field Description

Field	Description
Preamble	This sequence is required before the ID and header fields and allows Echo to synchronize with the clock encoded in the received signal and prepare for receiving the message.
ID	Determines which device or devices should receive the message. Only messages whose ID field matches the device ID stored in Echo's ID register are successfully received and sent to the MCU. Messages may be sent with any ID.
Header	The header signals the beginning of the data in a message. After the header has been recognized, Echo returns all following bytes to the MCU on the SPI lines until it encounters the end of message (EOM) sequence. The header is chosen via a `#define in Echo.h`. **Note:** There are some restrictions on the ID and header combinations that can be used. The header must not be contained within the ID or, if using ID repeat mode, it must not be contained in a sequence of IDs. **Note:** A further consideration is that the preamble must contain either one or four 0 bits, which form a prefix on the ID. The header must not be found within the prefixed ID. Device IDs must be unique even when prefixed with one or more 0 bits.
Length	This specifies the number of bytes in the immediately following data field. This may be a value up to 127.
Data	Contains the actual message sent: up to 127 data bytes.
Checksum	The checksum is the sum of the ID, length, and data fields (mod 256). This is generated automatically by the driver on transmission and checked on reception to provide a basic error detection facility.
EOM	Indicates the end of the current message.

4 Message Encoding

The application sets up transmit message buffers with the parameters and data required for the transmission. The driver reads and encodes the information before it is sent to Echo for transmission.

Individual bits are encoded by the driver using Manchester encoding. Sending a single bit with Manchester encoding requires that the value be sent during the first half bit-time and its complement sent during the second half bit-time. For example:

1 → ‾‾|__
0 → __|‾‾

The pulse-width modulation (PWM) mode of the MCU's timer module is used to produce this waveform:

- Set the timer modulus to the pulse width (this depends on the data rate)
- Set the output compare value to 0.5 modulus (50% duty cycle)

An interrupt occurs at output compare and we can set up for the next bit at this point.

The actual set up to send a single bit differs only slightly. The difference is to set high-true pulses or low-true pulses for a 1 or a 0, respectively.

NOTE

The driver requires close control of the timer channel it is using, along with the associated timer channel interrupt vector and pin. It also requires control of the timer modulus to set it up for a single bit-time. However, after it is set, the modulus is not modified further and can be used reliably on another timer channel. The driver can be configured via the header file `#define ECHO_TIMER_DISABLE` to leave the timer turned on even when the driver is not using it.

When the Manchester encoded bit-stream is presented to Echo, it is modulated onto the carrier using on-off keying (OOK) or frequency shift keying (FSK) for RF transmission.

Upon reception, Echo demodulates the RF signal and the data manager then removes the Manchester encoding, reducing the processing required on the MCU.

Figure 2 shows the sequence of events in an application where a message transmission is triggered by pressing a switch on one board, resulting in a LED flashing on another board.

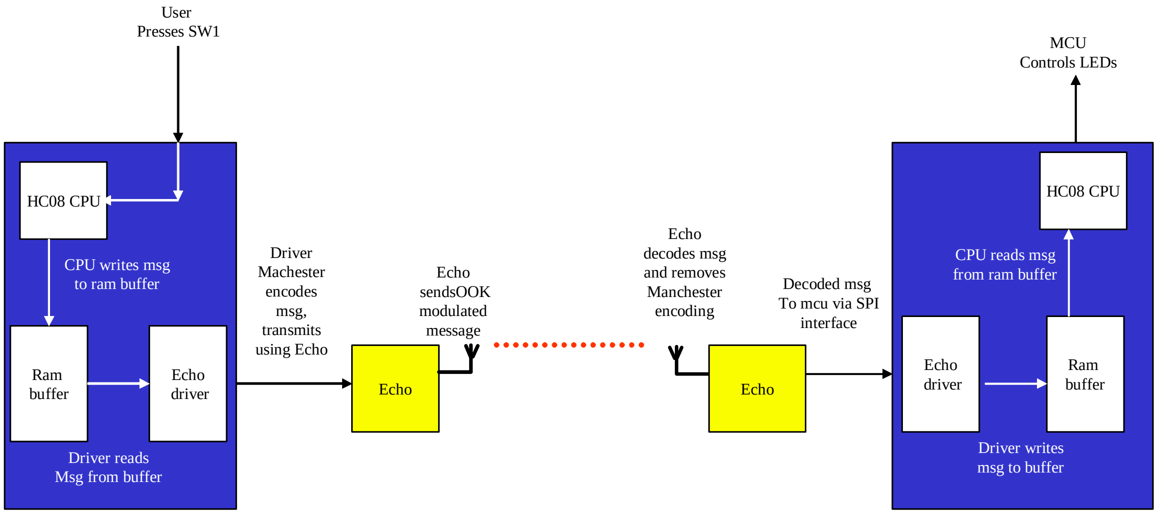


Figure 2. Data Flow and Message Encoding Steps in Message Transfer

5 Reducing Power Consumption

Echo can be placed into a standby mode to reduce power consumption. Controlling how Echo enters and wakes from standby mode can be achieved in two ways: using the internal strobe oscillator or external control via an I/O pin.

When Echo is in receive mode and the strobe oscillator is enabled (`#define ECHO_SOE_VALUE 1` in the driver's header file), the internal strobe oscillator uses the values `RON` and `ROFF` in the `RXONOFF` register to control how long Echo is awake and asleep, respectively. These values may be set in the Echo header file.

If Echo's strobe pin is connected to the MCU, it can control on/off periods directly by toggling this pin. When strobe is high, Echo will turn on and stay on. When strobe is low, Echo will turn off. If Echo recognizes an ID during an on period, it will stay on regardless of the strobe pin and internal strobe oscillator.

NOTE

> Echo takes little time to wake up out of standby before it is possible to receive a message. This time is typically 1.2 ms.

More detailed information on the operation of the strobe oscillator and strobe pin can be found in the Echo data sheet. In particular, the section titled 'Receiver On/Off Control' explains detailed timing information. The state machine descriptions show the interaction of the strobe oscillator and strobe pin on Echo's state.

5.1 Consequences on Message Format

If the power-saving features of Echo are used when receiving a message, a message can begin while Echo is in standby mode. To avoid this, the ID must be repeated long enough to cover at least two of Echo's ON periods. Both message formats described in Section 3, "Message Format," can repeat the ID over a period of time to allow Echo time to wake up. Figure 3 illustrates the required formats.

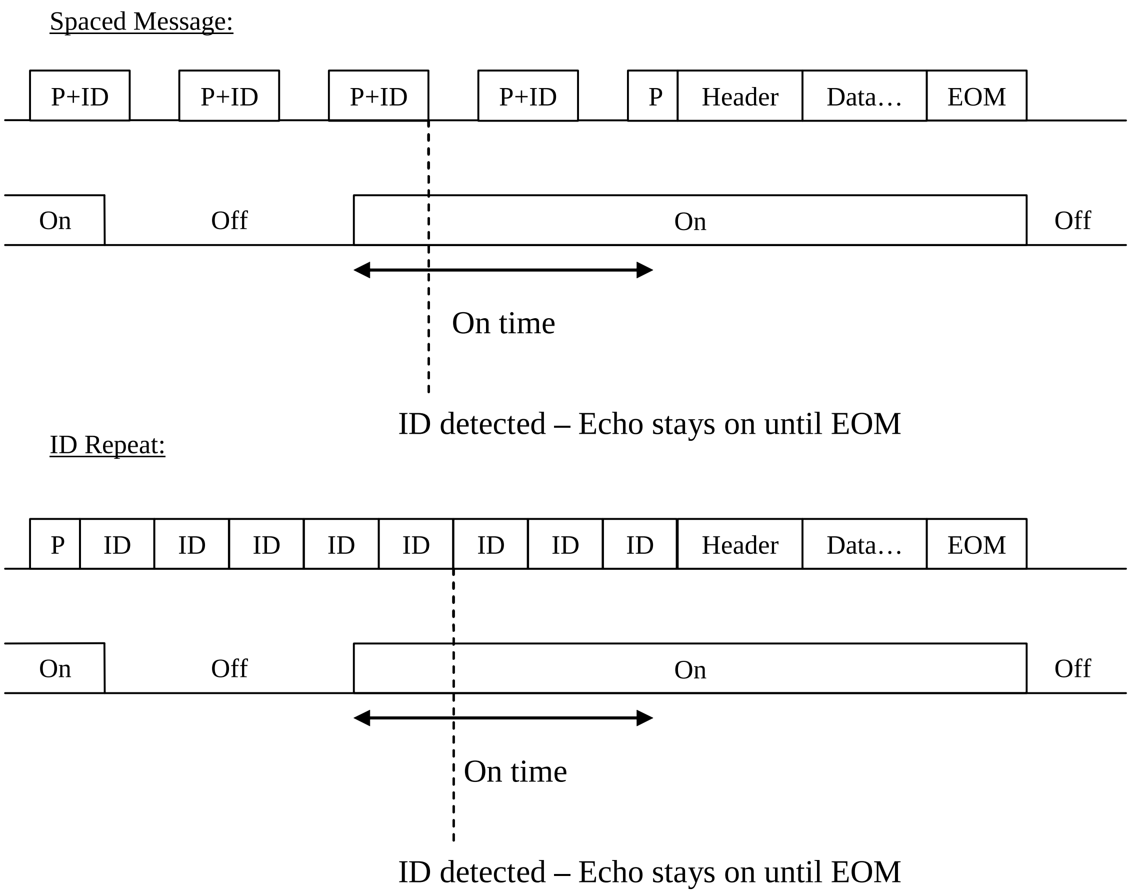


Figure 3. On/Off Time Cycling

6 Hardware Connections

Figure 4 shows the connection of Echo to the MCU, highlighting the required and optional connections. The MCU pin used for each connection is defined in the Echo.H header file.

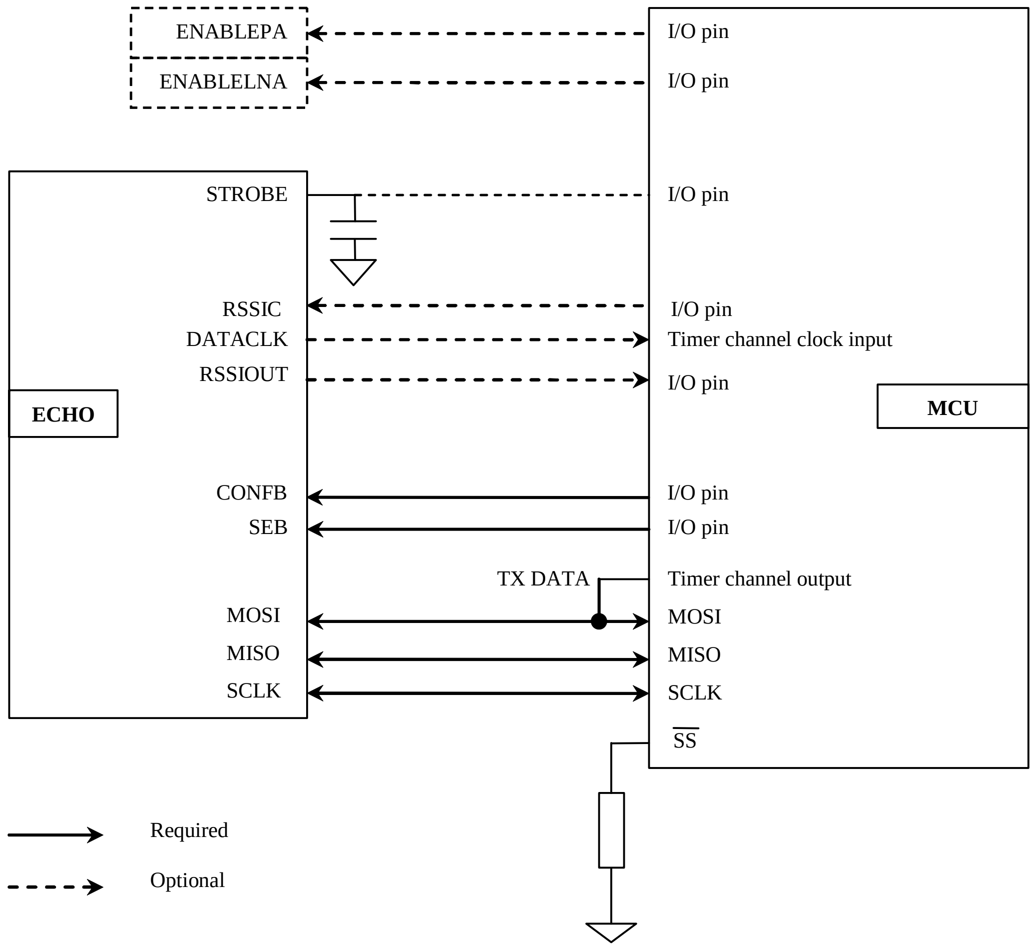


Figure 4. Echo Hardware Connections

A brief description of each connection follows:

- MOSI, MISO, SCLK, and SS—SPI connections allowing configuration information to be sent to Echo and received data to be sent back to the MCU. SS must be held low when the MCU is an SPI slave and awaiting data from Echo. In many systems this can be grounded with a pulldown resistor.
- TX DATA—The MOSI pin on Echo has dual functionality. When Echo is in receive mode, received data is output on the SPI to the MCU. When in transmit mode, the MOSI pin directly receives the waveform to be modulated. The driver generates the Manchester encoded waveform using a timer channel, and the corresponding channel pin is connected to MOSI when in transmit mode.
- SEB—Enables the digital interface on Echo. When SEB is high, the MOSI, MISO, and SCLK pins on Echo are high impedance, allowing other devices to be used on the SPI.
- CONFB—Resets Echo into configuration mode to allow access to Echo’s internal registers.

NOTE

The RSSIE bit in Echo register COMMAND must be set to 1 to enable the entire RSSI module before RSSI measurements can be made.

- RSSIC—Outputs an analog indication of the RSSI. This can be sampled with an MCU ADC to obtain greater accuracy or faster sampling than possible with Echo's internal conversion. The driver can be configured to use an MCU ATD module to read analogue RSSI measurements.

NOTE

Applications can enable the RSSI module, set RSSIC high of their own accord, and use an ADC directly without driver intervention.

- DATACLK—Can be used to provide the MCU with an accurate clock source for generating the output waveforms for transmission. This is useful on MCUs that use a low-accuracy clock, such as an RC oscillator.

  The driver can be configured to allow Echo to output DATACLK and to use this signal as an external clock input for the timer.
- STROBE—Can be used to control power by turning Echo on and off when in receive mode. See Section 5, "Reducing Power Consumption," for more information.
- ENABLEPA—This signal can be used on Freescale's Echo RF modules to control an additional external amplification stage on the output transmission. When ENABLEPA is 1, the power amplifier is enabled. When ENABLEPA is 0, the power amplifier is disabled. The driver can be configured to ignore ENABLEPA.
- ENABLELNA—This signal can be used on Freescale's Echo RF modules to control an additional external amplification stage on the received signal. When ENABLELNA is 1, the low noise amplifier is enabled. When ENABLELNA is 0, the low noise amplifier is disabled. The driver can be configured to ignore ENABLELNA.

6.1 MCU Resources Required

The hardware connection diagram shown in Figure 4 illustrates the required and optional resources that the driver can be configured to use.

- Required resources:
  - SPI—SPI connections for communication with and configuration of Echo.
  - Timer channel—A single timer channel must be completely under driver control, including the associated channel I/O pin. The driver must be linked to the channel interrupt vector as well.

NOTE

The driver must be able to initialize the timer modulus for the timer containing the channel; it cannot then be altered.

  - Three I/O pins—General-purpose I/O pins are required for configuration of Echo. SEB and CONFB are required.

- Optional resources:
  - — Timer channel—A second timer channel may be used to allow DATACLK to be input as a clock source for the timer.
  - — I/O pin—For STROBE line in order to allow the application direct control of Echo on/off cycling.
  - — I/O pin—Connects RSSIC to the signal strength measurement and allows Echo to begin an internal conversion of the signal strength to be placed in the RSSI register. While RSSIC is held high, conversions continue as long as CONFB is high.
  - — I/O pin—For ENABLEPA to allow the driver to control an additional external amplifier for transmission.
  - — I/O pin—For ENABLELNA to allow the driver to control an additional external amplifier for reception.

7 Driver Overview

The driver is supplied in two C code files – Echo.C and Echo.H.

The application communicates with the driver through the services and globally accessible storage shown in Table 2 and Table 3.

Table 2. Driver Services

Driver	Function
Echo_Initialize	Sets the initial configuration for Echo and performs essential driver set up. Must be called before using any other driver service.
Echo_Enable	Turns Echo on; goes into receive mode after 2 ms startup delay.
Echo_Disable	Turns Echo off; no messages can be received or transmitted.
Echo_DriverStatus	Returns Echo’s current status.
Echo_ClearError	Clears the error and timeout flags.
Echo_SendPreambleID	Sends a preamble/ID sequence.
Echo_SendData	Sends the body of a message.
Echo_SendMessage	Sends an entire message telegram.
Echo_SendIDRepeat	Sends a message in ID repeat format.
Echo_StrobeHigh	Sets the STROBE line to 1.
Echo_StrobeLow	Sets the STROBE line to 0.
Echo_StrobeTriState	Tristates the STROBE line.
Echo_TxTimer_Interrupt	Used by the driver to generate the transmitted waveform.
Echo_RxSPI_Interrupt	Used by the driver to receive and process messages.
Echo_SetFreq	Sets Echo frequencies using user friendly-access mode.
Echo_SetFreqNoFRM	Sets Echo frequencies directly.
Echo_Set_RxOnOff	Sets up RXONOFF register.

Table 2. Driver Services (continued)

Driver	Function
Echo_EnableRSSI	Enables the RSSI module and read RSSI on incoming messages automatically.
Echo_DisableRSSI	Disables the RSSI module.
Echo_SetRxSensitivity	Sets receive sensitivity: 1 [sensitive] – 4 [less sensitive].
Echo_SetTxPower	Sets output power: 1 [high power] – 4 [low power].
Echo_RagcHigh	Resets the automatic gain control to maximum level.
Echo_RagcLow	Allows automatic gain control to take place.
Echo_FagcHigh	Freezes the automatic gain control level.
Echo_FagcLow	Unfreezes the automatic gain control level.
Echo_ChangeConfig	Reads or writes directly to Echos internal registers.
Echo_ChangeBank	Changes active bank of Echos registers.

Table 3. Driver Storage

Driver	Function
unsigned char echoTransmitBuffer[ECHO_MAX_DATA_SIZE+4]	Buffer that the application must fill with transmit settings and data. For buffer format, see Section 9.5, "Buffer Formats."
unsigned char *echoNextMessage	Pointer to the next received message; updates after Echo_DriverStatus()

8 Status Word

The status word returned by `Echo_DriverStatus()` is a 16-bit unsigned int. Figure 5 shows the format of the word.

Bit	15	14	13	12	11	10	9	8
Name	SwitchMode	Timeout	LVD	Error	RSSI_InPROGRESS	RSSI_Enabled	ModeSwitchDelay	EnableDelay

Bit	7	6	5	4	3	2	1	0
Name	Mode	Overrun	Checksum	Rx	Tx	MSGReady	Busy	Enabled

Figure 5. Status Word Bit Allocation

The function of each bit is described in Table 4. Multiple bits may be set at any one time to reflect the state of the device driver at that time.

Table 4. Status Word Field Description

Bit	Name	Description	0	1
0	Enabled	Indicates when the Echo module is enabled.	Disabled	Enabled
1	Busy	Echo driver is busy.	—	Busy
2	MsgReady	There is a message ready in an Rx buffer. The pointer '`echoNextMessage`' indicates the start of the buffer. The `Checksum` bit indicates if this buffer had a checksum error.	No message	Message ready in buffer
3	Tx	The driver is currently transmitting information via Echo.	Not transmitting	Transmission in progress
4	Rx	A message is being received and processed by the driver. Check this flag before sending a message to avoid losing a message.	Not currently receiving	Receive is in progress
5	Checksum	The message at `echoNextMessage` has a checksum error.	Checksum ok	Checksum error
6	Overrun	All Rx buffers are full and another message has arrived with nowhere to be stored.	—	Buffer overrun
7	Mode	Indicates whether Echo is in transmit or receive mode.	Receive Mode	Transmit Mode
8	EnableDelay	Echo is powering up out of standby; this takes 2 ms.	—	Echo powering up
9	ModeSwitchDelay	Echo is switching between modes; delay is approximately 500 μs	—	Echo switching modes
10	RSSI_Enabled	Indicates if the RSSI module is on or off	RSSI disabled	RSSI enabled
11	RSSI_InProgress	Indicates if the MCU ADC is currently in use by the driver. When an analog RSSI is requested, the ADC must be free for the driver to use. This flag indicates when the application can take control of the ADC again, if necessary.	ADC free	ADC busy
12	Error	Indicates that an error has occurred with a request, e.g. RSSI.	—	Error
13	LVD	Indicates Echo has detected its supply voltage is < 1.8 V.	—	Low Voltage Detected
14	Timeout	A timeout has occurred when waiting on a byte to be received.	—	Timeout occurred
15	SwitchMode	Indicates whether Echo is in bank switching mode.	Only one bank active	Two banks active

9 Using the Driver

9.1 General Conventions

While the driver is performing a service, the busy status flag is set. After any request by the application for the driver to perform a service, the application must verify the busy flag is cleared to determine when the service has completed. This is illustrated in Figure 6.

If the driver is busy, calls to any service (excluding `Echo_DriverStatus()`) may corrupt the request and result in unspecified behavior. Exceptions to this rule, such as the ability to double buffer transmitted messages using overlapped calls to `Echo_SendMessage()`, are detailed in the documentation for that service (see Section 9.3, "Transmitting Messages").

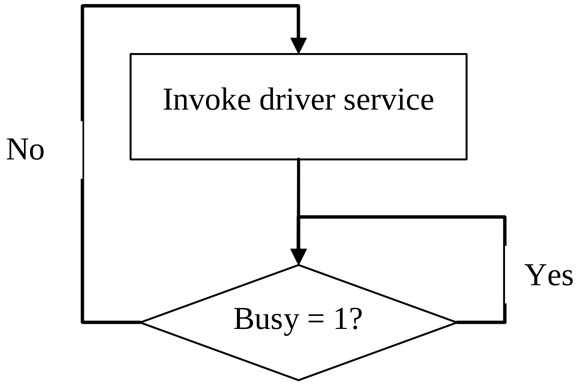


Figure 6. Driver Busy Flag Paradigm

The error flag indicates if any error occurred during the processing of a request. Sources of error include:

- timeout from the receive states
- attempting to read RSSI without the module enabled
- attempting to call `Echo_SendData()` when not in transmit mode
- or calling `Echo_ChangeConfig()` with illegal parameters

Errors may also occur when the Tx or Rx state machines are corrupted, indicating some problem outside the driver's control. To clear an error or timeout, call the `Echo_ClearError()` service.

9.2 Setup and Initialization

Before the driver can be used, some initial set up is necessary. This is performed in the `Echo_Initialize()` service. Consequently, `Echo_Initialize()` must be called before using any other driver service.

After the driver has been initialized, Echo must be enabled before any messages can be sent or received or before any function on Echo can be used. Echo can be configured while disabled. A typical start up sequence is shown in Figure 7.

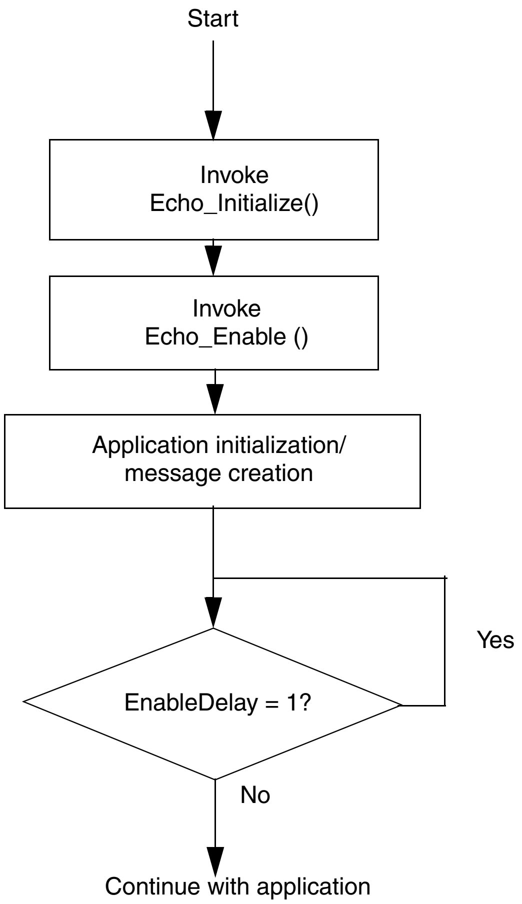


Figure 7. Echo Driver Start Up

After invoking `Echo_Enable()`, there is a 2 ms start up delay while Echo is brought out of standby mode. At this point, other application initialization may occur or messages for transmission may be constructed to be sent after start up.

9.3 Transmitting Messages

There are two different formats for message transmission as detailed in section Section 3, "Message Format." The driver also provides support for double buffering of transmitted messages and explicit control of the timing and number of Preamble/ID sequences. To support transmission in both formats and allow explicit control of Preamble/ID repeats, there are a total of four services provided by the driver:

- `Echo_SendPreambleID`—Send a preamble/ID sequence
- `Echo_SendData`—Send the body of a message
- `Echo_SendMessage`—Send an entire message telegram
- `Echo_SendIDRepeat`—Send a message in ID repeat format

9.3.1 Manual Preamble/ID Sending

The first two services allow the application to transmit Preamble/ID sequences and the data sequence manually. This may be necessary if the timing control provided by the driver – in the `Echo_SendMessage()` service – does not suit the application requirements.

The sequence for sending messages using `Echo_SendPreambleID()` and `Echo_SendData()` is similar to the equivalent Tango services. This is summarized in Figure 8.

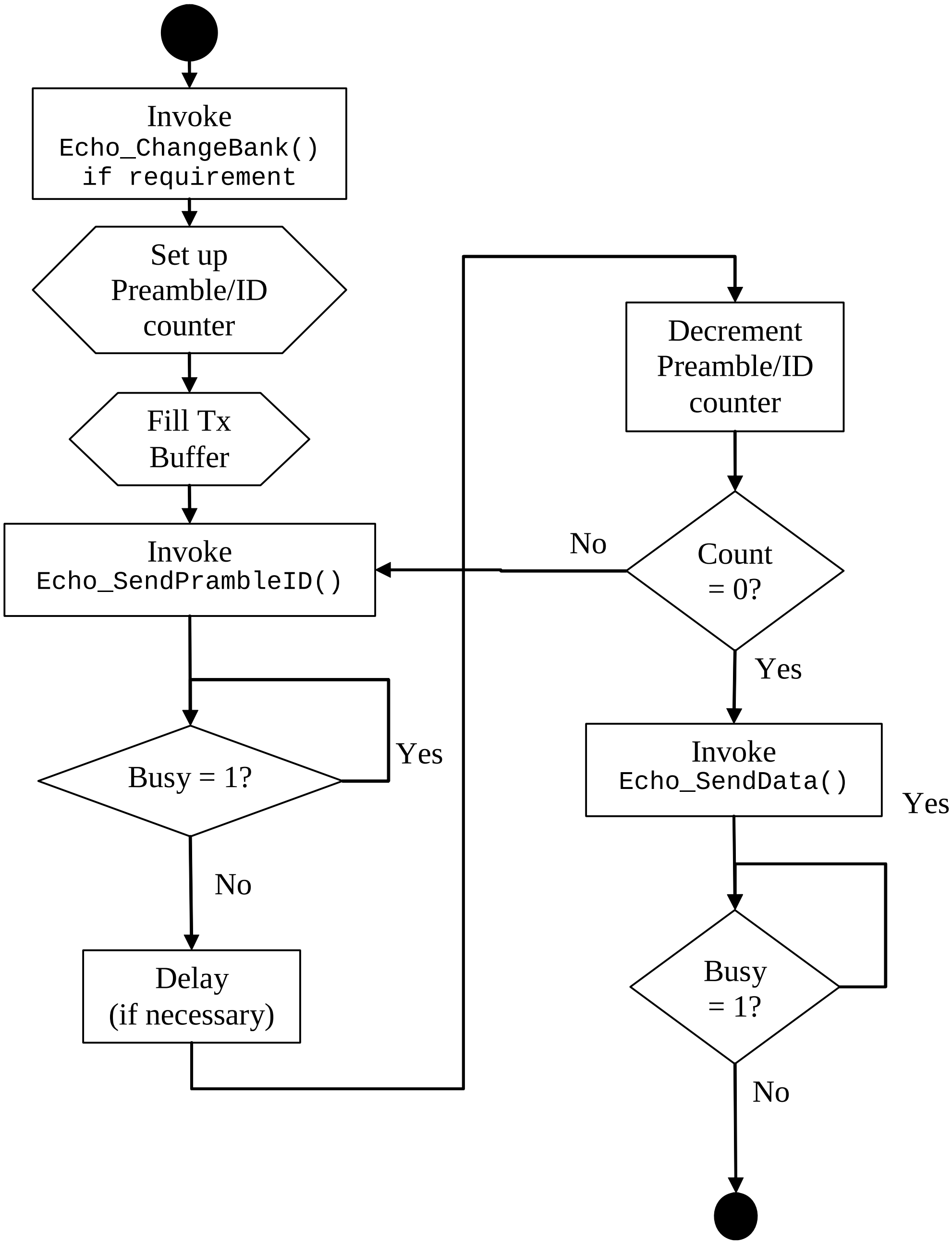
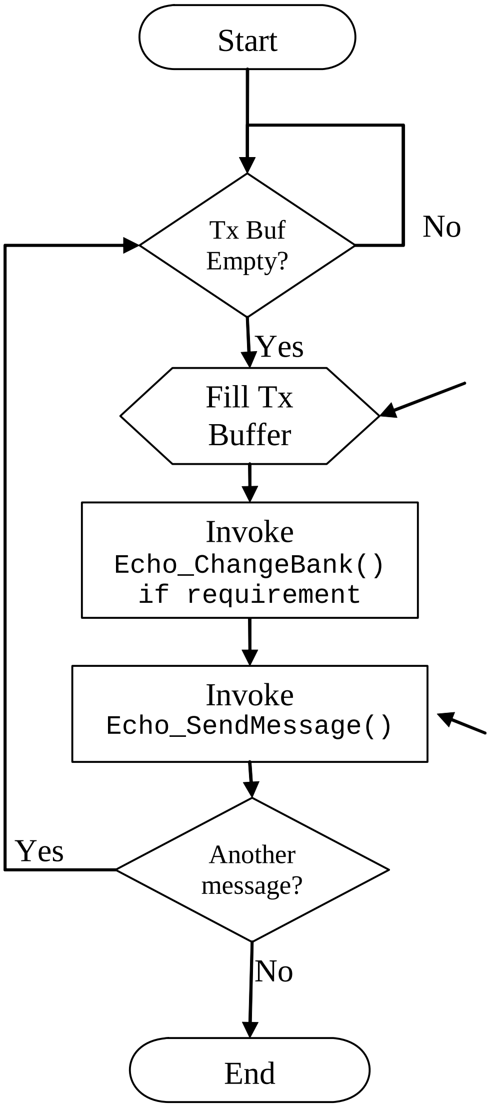


Figure 8. Manual Preamble/ID Sending

9.3.2 Automatic Preamble/ID Sending and Message Queuing

Using the `Echo_SendMessage()` service, it is possible to queue messages for transmission. To queue messages, you must specify the number of Preamble/ID sequences to preface the message with and, of course, the spacing between each sequence. This information is stored in the transmit buffer (see Section 9.5, “Buffer Formats”). To simplify timing, the spacing between messages is measured in bit times. With this information stored alongside the message, a complete telegram can be sent without the intervention of the application software.

When queueing messages with `Echo_SendMessage()`, the sequence of events is shown in Figure 9


Start
Tx Buf Empty?
No
Yes
Fill Tx Buffer
Invoke Echo_ChangeBank() if requirement
Invoke Echo_SendMessage()
Yes
Another message?
No
End

Application must clear the full flag. `Echo_SendMessage()` will set the full flag to indicate the Tx buffer is busy.

The Echo driver initiates the sending. When possible, the message is copied into the internal buffer, transmission begins and the buffer full flag is cleared to allow another message to be queued.

Figure 9. Querying Complete Telegrams

9.3.3 ID Repeat Message Format

Sending messages using ID repeat format is simple. Fill the transmit buffer and, when the driver is not busy, invoke the `Echo_SendIDRepeat` service with the desired number of IDs to be repeated as a parameter.

9.4 Receiving Messages

For receiving messages, the driver is flexible enough to accommodate a number of buffers defined in the header file. The scheme employed provides a globally accessible pointer for the next full receive data buffer.

The pointer is updated after calling `Echo_DriverStatus()`. If a message is available in the buffers, the status bit `MsgReady` is set and the pointer `echoNextMessage` is updated. If that message has a checksum error, the `Checksum` status bit is also set. If no messages are waiting, `echoNextMessage` is undefined and the status bit `MsgReady` is cleared.

Internally, there is a temporary receive buffer that the Echo driver uses for partially received messages and associated processing. This is copied to a free buffer when the reception is complete. If no free buffers exist, the message will be discarded and the `Overrun` status bit is set.

When the application has finished using the buffer at `echoNextMessage`, it must clear the buffer full bit to return it to the pool before the next message will be released to it. If the buffer full bit is not cleared, subsequent calls to `Echo_DriverStatus()` will not update the `echoNextMessage` pointer.

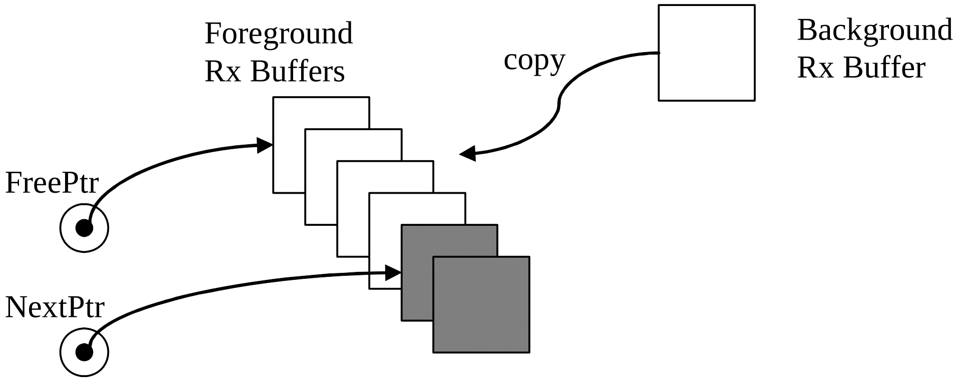


Figure 10. Receive Buffering

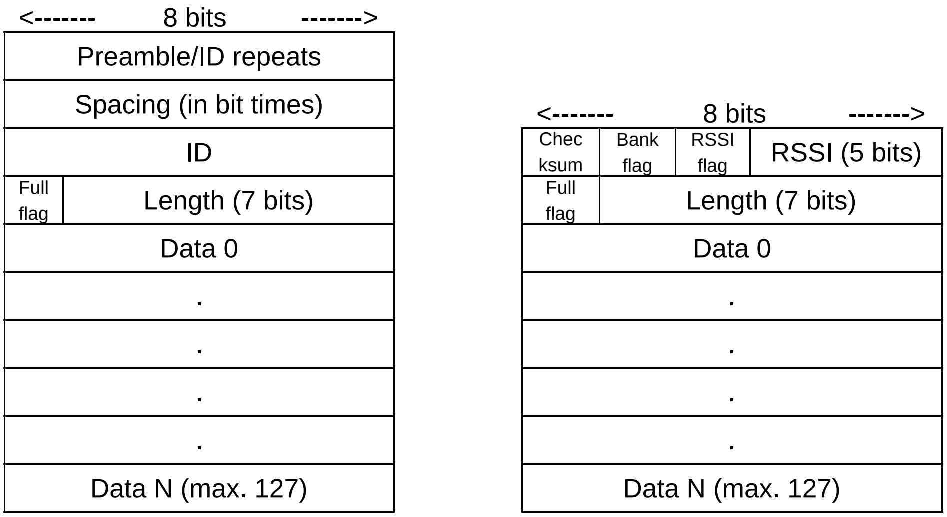
9.4.1 Timeouts

If the length is corrupted or part of a message is lost, the driver can remain in a waiting state for the next byte of a message to arrive. Consequently, parts of future messages can be mistakenly read as part of this message. To avoid this, the timer provides a timeout mechanism while receiving.

The timeout is measured in bit-times and must clearly be greater than 8 bits (to allow a full byte to arrive and be transferred out of Echo onto the SPI). This is defined in the driver header file: `ECHO_RX_TIMEOUT_BITS`.

If a timeout occurs, both the `Timeout` and `Error` status bits are set.

9.5 Buffer Formats



Tx Buffer **Rx Buffer**

Figure 11. Buffer Formats

9.5.1 Preamble/ID Repeats

When using the `Echo_SendMessage` service, this byte defines the number of times to repeat the preamble/ID sequence before sending the data.

9.5.2 Spacing

When using the `Echo_SendMessage` service, this byte defines the number of bit-times to leave between repetitions of the preamble/ID sequence.

9.5.3 ID

The message ID. The actual number of bits sent for the ID depends on the configuration option `ECHO_ID_LENGTH` in the driver header file. If the length is less than eight, the bits used correspond to the least significant bits in this byte.

9.5.4 Full Flag

The full flag indicates when the buffer is in use, either by the driver (for the Tx buffer) or by the application (for the Rx buffer). Using this bit limits the maximum data length to 127 bytes.

When any of the message sending services are called, the buffer is marked full for the duration of its use by the driver. Only when the driver clears the full flag can the transmit buffer be modified again. Modifying it while this bit is set can lead to corruption of the messages sent or other undesirable and unpredictable effects.

When the driver fills an Rx buffer, it sets the full flag to signify the buffer contains a message. The application must clear the full flag to return the buffer to the pool and be available to receive more messages. The next message will not be returned to the application until the previous message is released.

9.5.5 Checksum

Although not included in the buffer, the checksum is calculated and automatically sent with messages by the driver software. To maintain compatibility with the Tango and Romeo device drivers, the checksums sent and received are calculated over the ID, length, and data bytes.

The checksum provides a minimal error detection mechanism. The checksum bit in the Rx buffer indicates that a checksum error has been detected. This is set to 1 on error and cleared otherwise.

9.5.6 Bank Flag

If both banks are active, this bit indicates which bank received the message.

9.5.7 RSSI

If the automatic RSSI reading is enabled (by calling `Echo_EnableRSSI()`), the RSSI of incoming messages is automatically measured and stored with the messages. The 5 bit RSSI value is the sum of the upper and lower 4-bit portions of the Echo RSSI register value. If necessary, this can be used to calculate the signal strength in dBm.

If the automatic RSSI reading is disabled, the RSSI value will not be present with the message. The RSSI flag bit indicates whether there is a valid RSSI measurement for this message; the RSSI flag is set to 1 if the RSSI bits contain a valid measurement.

10 Function Headers and Description

10.1 Initialization and Startup Functions

10.1.1 Echo_Initialize

Syntax:	`void Echo_Initialize(void)`
Parameters:	None
Return:	None

Description: This service initializes the Echo module and the device driver software. The Echo module is configured according to the settings provided in the Echo.H header file via the SPI; the driver status is disabled; both necessary and optional port pins are configured if used. To reduce power consumption, neither the Echo module nor the external amplifiers (if present) are enabled.

Notes: This service must be called before using any others provided by the Echo driver. Failure to do so produces unpredictable results.

10.1.2 Echo_Enable

Syntax: `void Echo_Enable(void);`

Parameters: None

Return: None

Description: This service powers Echo into receive mode and waits for arriving messages. It also enables the external LNA (if present) and schedules a 2 ms start-up delay for Echo using the timer. If the strobe pin is under driver control, the strobe is taken high to turn Echo on. The RSSI module is disabled by default, to conserve power. Status bit `Enabled` is set.

Notes: While the driver is in the enable delay state (status bit `EnableDelay` set), no further services from the Echo driver may be invoked. However, a message may be constructed into the Tx buffer in preparation to send when the driver state returns to idle (status bit `Busy` cleared).

10.1.3 Echo_Disable

Syntax: `void Echo_Disable(void);`

Parameters: None

Return: None

Description: This service powers down Echo into standby mode. The PA, LNA, and strobe are set low. The timer may be turned off depending on the `ECHO_TIMER_DISABLE` configuration option. Status bit `Enabled` is cleared.

Notes: This service immediately initiates a shutdown of Echo. Any transmission or reception in progress will be abandoned.

10.2 Status Functions

10.2.1 Echo_DriverStatus

Syntax: `tECHO_STATUS Echo_DriverStatus(void);`

Parameters: None

Return: 16-bit flag word indicating the status of the driver.

Description: Indicates the status of the device driver via the bit flags returned. The format of the status word is detailed in Section 8, "Status Word."

Notes: This service must be called before attempting to read from the Rx buffer to ensure there is valid data in the buffer at `echoNextMessage` (if a message has been received).

10.2.2 Echo_ClearError

Syntax: `void Echo_ClearError(void);`

Parameters: None

Return: None

Description: Clears the error and timeout bits.

10.3 Message Sending Functions

10.3.1 Echo_SendPreambleID

Syntax: `void Echo_SendPreambleID(void);`

Parameters: None

Return: None

Description: Switches to transmit mode and initiates sending a message consisting of a preamble sequence followed by the target device ID. The ID is read from the ID byte in the Tx buffer, with the number of bits defined by the `ECHO_ID_LENGTH` configuration option. During transmission, the status flag `BusyTx` is set.

Notes: If `ECHO_ID_LENGTH` is less than eight, the ID should reside in the bottom `ECHO_ID_LENGTH` bits of the ID byte. This service does not switch back to receive mode and should be used with the `Echo_SendData()` service to send complete messages and return to receive mode (see Figure 8). Before using this function to switch driver mode (both banks are enabled), the right bank of registers must be set by `Echo_ChangeBank(ECHO_BANK_x)`.

10.3.2 Echo_SendData

Syntax: `void Echo_SendData(void);`

Parameters: None

Return: None

Description: Initiates sending a message consisting of a preamble sequence, header, length, data, and checksum. Length and data are read from the Tx buffer; the header is statically defined in the `Echo.h` configuration file; and checksum is calculated (and confirmed on reception) by the driver. During transmission, the status flag `BusyTx` is set. Switches back to Rx mode on completion of the transmission.

Notes: This service requires the driver to be in transmit mode before entry and should be used in conjunction with the `Echo_SendPreambleID()` service to send complete messages (see Figure 8). Before using this function to switch driver mode (both banks are enabled), the right bank of registers must be set by `Echo_ChangeBank(ECHO_BANK_x)`.

10.3.3 Echo_SendMessage

Syntax: `void Echo_SendMessage(void);`

Parameters: None

Return: None

Description: Switches to Tx mode and initiates the buffered sending of a complete message telegram sequence consisting of a number of preamble/ID sequences followed the message in the same format as `Echo_SendData()`. The number of preamble/ID sequences to repeat and the spacing between them are defined by parameters in the Tx buffer. During transmission, the status flag `BusyTx` is set.

Notes: The message being sent is copied into an internal buffer, thus freeing the external buffer to queue another message for transmission. To queue another message, verify the buffer is empty (using the buffer full bit) and if so, load in the next message. Call `Echo_SendMessage()` again, which will set the buffer full flag until the message is copied into the internal buffer to be sent. At that point another message can be queued. If no new message is queued at the end of transmission of a message, the mode is switched back to Rx (incurring a mode switch delay of 500 µs). Before using this function to switch driver mode (both banks are enabled), the right bank of registers must be set by `Echo_ChangeBank(ECHO_BANK_x)`.

10.3.4 Echo_SendIDRepeat

Syntax: `void Echo_SendIDRepeat(unsigned char repeatCount);`

Parameters: `repeatCount`—The number of additional IDs to send.

Return: None

Description: Switches to Tx mode and initiates the sending of a complete ID repeat message telegram sequence. This consists of a preamble and number of consecutive IDs followed immediately by a header and the message in the same format as `Echo_SendData()`. During transmission, the status flag `BusyTx` is set. Before using this function to switch driver mode (both banks are enabled), the right bank of registers must be set by `Echo_ChangeBank(ECHO_BANK_x)`.

10.4 Strobe Power Control Functions

10.4.1 Echo_StrobeHigh

Syntax: `void Echo_StrobeHigh(void);`

Parameters: None

Return: None

Description: Brings the strobe high if the strobe pin is under driver control. This allows control of Echo on/off time in receive mode to manage power consumption. Strobe high turns the receive circuit on.

Notes: See Section 5, “Reducing Power Consumption,” for more information.

10.4.2 Echo_StrobeLow

Syntax: `void Echo_StrobeLow(void);`

Parameters: None

Return: None

Description: If the strobe pin is under driver control, this function takes the strobe low. This allows control of Echo on/off time in receive mode to manage power consumption. Strobe low stops the off counter, which maintains the receive circuit off.

Notes: See Section 5, “Reducing Power Consumption,” for more information.

10.4.3 Echo_StrobeTriState

Syntax: `void Echo_StrobeTriState(void);`

Parameters: None

Return: None

Description: If the strobe pin is under driver control, this function sets the strobe pin to high impedance. When the strobe is in a high impedance state, Echo on/off time can be controlled by Echo's internal timer. This can be set up for a range of on and off times via the `ECHO_RXON_VALUE` and `ECHO_RXOFF_VALUE` configuration options.

Notes: See section Section 5, "Reducing Power Consumption," for more information.

10.4.4 Echo_Set_RxOnOff

Syntax: `void Echo_Set_RxOnOff(byte on, byte off);`

Parameters: on—the value to set the Rx on timer
Off—time bits in RXONOFF register

Description: When the Echo on/off time is controlled by Echo's internal time, this function can change in runtime on and off times in the `RXONOFF` configuration registers.

Notes: See Section 5, "Reducing Power Consumption," for more information. The driver sets up the working bank. If you have selected both banks, the driver sets up the register in both banks.

10.5 Driver Internal Processing

10.5.1 Echo_TxTimer_Interrupt

Syntax: `interrupt void Echo_TxTimer_Interrupt(void);`

Parameters: None

Return: None

Description: This function is used primarily for generating the waveforms for transmission, and other timing requirements for the Echo driver. This function must be linked to the interrupt vector for the timer channel the Echo driver is configured to use.

10.5.2 Echo_RxSPI_Interrupt

Syntax: `interrupt void Echo_RxSPI_Interrupt(void);`

Parameters: None

Return: None

Description: This function is used by the driver to receive information from the Echo module. This function must be linked to the interrupt vector for the SPI module in use.

10.6 Programmable Frequency

10.6.1 Echo_SetFreqOok, Echo_SetFreqFsk (Echo_SetFreq only for compatibility)

Syntax: [OOK modulation]—`Echo_SetFreqOok(unsigned int carrier);`

[FSK Modulation]—`Echo_SetFreqFsk(unsigned int carrier, unsigned char deltaF);`

Parameters: `carrier`—the value to set the carrier register bits to
`deltaF`—the value to set the frequency deviation to

Return: None

Description: Allows the frequency within the (statically) selected transmission band to be controlled. The values to place in each register may be calculated using the formula from the Echo data sheet.

Notes: Requires Echo to be reset into configuration mode and will abort any transmission or reception currently in progress. Status bit `Busy` is set while the configuration takes place. The driver sets up the working bank. If you have selected both banks, the driver sets the register up in both banks.

10.6.2 Echo_SetFreqOokNoFrm, Echo_SetFreqFskNoFrm (Echo_SetFreqNoFRM only for compatibility)

Syntax: [OOK modulation]—`Echo_SetFreqOokNoFRM(`

```
unsigned intlocalOscillator,
unsignedintcarrier0);
```

[FSKModulation]—`Echo_SetFreqFskNoFRM(`

```
unsignedintlocalOscillator,unsiged int carrier0,
unsigned int carrier1|);
```

Parameters: `localOscillator`—the value to set the LO to:

`carrier0`— the value to set the carrier0 register bits to.

`carrier1`—the value to set the carrier1 register bits to.

Return: None

Description: Allows the frequency within the (statically) selected transmission band to be controlled. The values to place in each register may be calculated using the formula from the Echo data sheet.

Notes: Requires Echo to be reset into configuration mode and thus will abort any transmission or reception currently in progress. Status bit `Busy` is set while the configuration takes place. The driver sets up the working bank. If you have selected both banks, the driver sets the register up in both banks.

10.7 RSSI

10.7.1 Echo_EnableRSSI

Syntax: `void Echo_EnableRSSI(void);`

Parameters: None

Return: None

Description: Enables the RSSI module and automatic reading of the RSSI during the reception of a message. If this is enabled, the RSSI will be measured during reception of any messages and stored in the Rx buffer along with the received data. The RSSI flag in the Rx buffer will indicate the presence of a valid RSSI reading in the buffer.

Notes: Requires Echo to be reset into configuration mode and thus will abort any transmission or reception currently in progress. Status bit `Busy` is set during configuration.

10.7.2 Echo_DisableRSSI

Syntax: `void Echo_DisableRSSI(void);`

Parameters: None

Return: None

Description: Turns off the RSSI module and disables the automatic reading of the RSSI during the reception of a message.

Notes: Requires Echo to be reset into configuration mode and thus will abort any transmission or reception currently in progress. Status bit `Busy` is set during configuration.

10.8 Power and Sensitivity

10.8.1 Echo_SetRxSensitivity

Syntax: `void Echo_SetRxSensitivity(unsigned char level);`

Parameters: `level`—sets the sensitivity to level in the range `0` to `3`

Return: None

Description: Sets the sensitivity of input stage where `0` is the most sensitive and `3` is the least sensitive.

Notes: Requires Echo to be reset into configuration mode and thus aborts any transmission or reception currently in progress. Status bit `Busy` is set during the configuration. The driver sets up the working bank. If you have selected both banks, the driver sets up the register in both banks.

10.8.2 Echo_SetTxPower

Syntax: `void Echo_SetTxPower(unsigned char level);`

Parameters: `level`—sets the output power to level in the range `0` to `3`

Return: None

Description: Sets the power of output stage where `0` is the highest power to `3` the lowest power.

Notes: Requires Echo to be reset into configuration mode and thus aborts any transmission or reception currently in progress. Status bit `Busy` is set during the configuration. The driver sets up the working bank. If you have selected both banks, the driver sets up the register in both banks.

10.8.3 Echo_RagcHigh

Syntax: `void Echo_RagcHigh();`

Parameters: None

Return: None

Description: Resets the automatic gain control (AGC) to the maximum level. While this is high, the gain will stay at the maximum level. AGC can be re-enabled using `Echo_RagcLow()`.

Notes: Requires Echo to be reset into configuration mode and thus will abort any transmission or reception currently in progress. Status bit `Busy` is set during the configuration. The driver sets up the working bank. If you have selected both banks, the driver sets up the register in both banks.

10.8.4 Echo_RagcLow

Syntax: `void Echo_RagcLow();`

Parameters: None

Return: None

Description: Stops holding the AGC at the maximum level. This allows AGC to modify the gain as necessary.

Notes: Requires Echo to be reset into configuration mode and thus will abort any transmission or reception currently in progress. Status bit `Busy` is set during the configuration. The driver sets up the working bank. If you have selected both banks, the driver sets up the register in both banks.

10.8.5 Echo_FagcHigh

Syntax: `void Echo_FagcHigh();`

Parameters: None

Return: None

Description: Freezes the AGC level. While high, the gain stays frozen.

Notes: Requires Echo to be reset into configuration mode and thus will abort any transmission or reception currently in progress. Status bit `Busy` is set during the configuration. The driver sets up the working bank. If you have selected both banks, the driver sets up the register in both banks.

10.8.6 Echo_FagcLow

Syntax: `void Echo_FagcLow();`

Parameters: None

Return: None

Description: Unfreeze the AGC level.

Notes: Requires Echo to be reset into configuration mode and thus will abort any transmission or reception currently in progress. Status bit `Busy` is set during the configuration. The driver sets up the working bank. If you have selected both banks, the driver sets up the register in both banks.

10.9 Direct Register Access

10.9.1 Echo_ChangeConfig

Syntax:

```
void Echo_ChangeConfig(unsigned char numReg,
unsigned char startReg,
unsigned char *regData,
unsigned char readWrite);
```

Parameters: `numReg`—number of registers to read or write [1, 2, 4, or 8]

`startReg`—register number to start at [0–15]

`regData`—buffer to read the bytes from or place the results into

`readWrite`— reads Echo or writes Echo [0 = read | 1 = write]

Return: None

Description: Allows direct control of the Echo status registers. Use caution when using this function. Modifying some Echo registers may leave the driver in an inconsistent state.

The number of registers must be 1, 2, 4, or 8. Multiple calls must be used if other numbers are required.

Notes: Requires Echo to be reset into configuration mode and thus will abort any transmission or reception currently in progress. Status bit `Busy` is set during the configuration.

10.9.2 Echo_Changebank

Syntax: `void Echo_Changebank(unsigned char bank);`

Parameters: `bank`—this parameter determines which bank will be active. [`ECHO_BANK_A, ECHO_BANK_B, ECHO_BANK_BOTH`]

Return: None

Description: Allows control bank switching in Echo.

Notes: Requires Echo to be reset into configuration mode and thus will abort any transmission or reception currently in progress. Status bit `Busy` is set during the configuration. If the required bank is unavailable, status bit `Error` is set.

11 Echo Driver Configuration

Many of the driver options are statically configured at compile time. These configuration options are defined in the Echo.h header file. Using these options it is possible to configure the driver to run on any HCS08 MCU.

The Echo.h header file contains a number of #define configuration options. These are described below:

11.1 Common Properties for Both Banks

11.1.1 Required I/O Pins

ECHO_CONFB

Description: This defines the I/O pin used to control Echo's CONFB pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior™ header file.

Example:

```
#define ECHO_CONFB PTED_PTED3 /* Define pin used for CONFB */
```

ECHO_CONFB_DDR

Description: This defines the data direction bit for the I/O pin used to control Echo's CONFB pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_CONFB_DDR PTEDD_PTEDD3
```

ECHO_SEB

Description: This defines the I/O pin used to control Echo's SEB pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_SEB PTED_PTED0 /* Define pin used for SEB */
```

ECHO_SEB_DDR

Description: This defines the data direction bit for the I/O pin used to control Echo’s SEB pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_SEB_DDR PTEDD_PTEDD0
```

11.1.2 SPI Setup

ECHO_SPI_ADDRESS

Description: This defines the start address of the SPI control registers in the MCU’s memory map.

Values: 0x0000–0xFFFF (see MCU data sheet for actual address)

Example:

```
#define ECHO_SPI_ADDRESS 0x40 /* Location of SPI registers */
```

ECHO_SPI_CLOCK_SPEED

Description: Defines the speed of the SPI clock.

Values: 0–20000000

Example:

```
#define ECHO_SPI_CLOCK_SPEED 8000000 /* SPI clock speed 8MHz */
```

11.1.3 Timer set up

ECHO_TIMER_ADDRESS

Description: This defines the address of the timer status and control register in the MCU’s memory map. The timers on all HCS08 MCUs have the same layout of control registers. The driver uses this base address to access the timer and control registers.

Values: Address in the range 0x0000 – 0xFFFF

Example:

```
#define ECHO_TIMER_ADDRESS 0x30 /* Location of 1st timer register */
```

ECHO_TIMER_CHANNEL

Description: This defines the timer channel used to output the Manchester encoded waveform to Echo.

Values: Channel number in the range 0–15

Example:

```
#define ECHO_TIMER_CHANNEL 1 /* Define which timer channel to use */
```

ECHO_TIMER_CLOCK_SOURCE

Description: This defines the clock used to control the timer.

Values:
1 = BUSCLK
2 = XCLK
3 = External clock source

Example:

```
#define ECHO_TIMER_CLOCK_SOURCE 1 /* Select clock source for timer */
```

ECHO_USE_DATACLK

Description: This defines whether DATACLK should be used by the driver. If this function is enabled and an external clock source is also selected, the external clock source is assumed to be DATACLK and `ECHO_TIMER_CLOCK_SPEED` is overwritten with the value calculated for the speed of DATACLK.

Values:
0 = disable DATACLK
1 = enable DATACLK

Example:

```
#define ECHO_USE_DATACLK 0 /* Indicate if Ext clock is DATACLK */
```

ECHO_TIMER_CLOCK_SPEED

Description: This defines the clock speed (in Hz) of the timer if an internal clock is chosen or if DATACLK is not enabled.

Values: Integer from 0 to MCU bus speed divided by four for an external clock

Example:

```
#define ECHO_TIMER_CLOCK_SPEED 8000000 timer clock speed in Hz */
```

ECHO_TIMER_PRESCALE]

Description: This defines the prescaler value of the timer used to send data to Echo. This will generally be set to 1. If the application sets the timer prescaler, this configuration option should be changed to reflect the selected prescaler.

Values: See data sheet for S08 MCU.

Example:

```
#define ECHO_TIMER_PRESCALE 1 /* Specify timer prescaler value */
```

ECHO_TIMER_DISABLE

Description: This allows the driver to switch off the MCU timer when it is not required by the driver. This can reduce power consumption; however, some applications may require that the timer continues running. The driver will always start the timer when required.

Values:
0 = Timer remains running when not required by the driver
1= Timer is disabled when not required by the driver

Example:

```
#define ECHO_TIMER_DISABLE 1 /* Allows driver to turn off timer */
```

ECHO_CRYSTAL_FREQUENCY

Description: This defines the speed (in Hz) of the crystal used by Echo. Typical values at supported RF frequencies are:
315 MHz—17581400
434 MHz—24190660
868 MHz—24161390

Values: Integer in the range 0–100000000

Example:

```
#define ECHO_CRYSTAL_FREQUENCY 24190660 /* Crystal frequency (in Hz) */
```

11.1.4 Buffering Parameters

ECHO_MAX_DATA_SIZE

Description: This defines the maximum number of data bytes that can be transferred. This is used to allocate buffer space for both transmitting and receiving messages.

Values: 1–127

Example:

```
#define ECHO_MAX_DATA_SIZE 127 /* Max length of data field in msg */
```

ECHO_RX_BUF_COUNT

Description: This defines the number of receive buffers that are allocated to store received messages in. Storage space allocated to external receive buffers is:
`(ECHO_MAX_DATA_SIZE+2)*ECHO_RX_BUF_COUNT.`

Values: Minimum of 1; maximum limited by size of RAM available.

Example:

```
#define ECHO_RX_BUF_COUNT 4 /* Number of Rx buffers to allocate */
```

11.1.5 Miscellaneous

ECHO_SWITCH_LEVEL

Description: Specifies the value to place Echo’s SL bit; this sets the active level of SWITCH output pin.

Values: 0 = Rx low, Tx high
1 = Tx low, Rx high

Example:

```
#define ECHO_SWITCH_LEVEL 0 /* Set active level of SWITCH o/p pin */
```

ECHO_LVD_ENABLE

Description: Specifies the value to place Echo's LVDE bit; this enables or disables the low-voltage detection module on Echo.

Values: 0 = disabled
1 = enabled

Example:

```
#define ECHO_LVD_ENABLE 0 /* Enable low voltage detection */
```

ECHO_MCU_BUS_SPEED

Description: Specifies the MCU BUSCLK speed. This is used to calculate various delays.

Values: 0–maximum MCU bus speed

Example:

```
#define ECHO_MCU_BUS_SPEED 8000000 /* In Hz */
```

ECHO_RX_TIMEOUT_BITS

Description: Specifies the number of bit-times to wait on a byte arriving when receiving a message before assuming the message has been corrupted or lost. This value must be greater than 8 bit-times to allow the next byte to arrive and be processed on Echo.

Values: 8–255

Example:

```
#define ECHO_RX_TIMEOUT_BITS 16 /* Number of bit-times before timeout */
```

ECHO_AUTO_SWITCH

Description: This feature can be enabled in bank-switching mode. By this switch you can set automatic switching to both banks after a transmit is done.

Values: 0 = disabled
1 = enabled

Example:

```
#define ECHO_AUTO_SWITCH  1 /* if is used both bank, then can switch automatically to both
banks used after each transmit */
```

11.2 Separate Parameters for Each Bank

11.2.1 Bank Select

ECHO_BANK_A_EN, ECHO_BANK_B_EN

Description: This defines which bank is used: bank A, bank B, or both. If you use only one bank, bank A is recommended.

Values:
0 = Bank is disabled
1 = Bank is enabled

Example:

```
#define ECHO_BANK_A_EN     1
```

11.2.2 Modulation Parameters

ECHO_MODE_VALUE

Description: This defines the RF modulation type used. On/off keying (OOK) or frequency shift keying (FSK).

Values:
ECHO_OOK OOK modulation
ECHO_FSK FSK modulation

Example:

```
#define ECHO_MODE_VALUE ECHO_FSK
```

ECHO_MODE_OOKREF

Description: This defines the value of the OOKREF bit in Echo’s CONFIG2 register. OOKREF controls the data slicer reference.

Values:
0 = fixed reference
1 = adaptive reference

Example:

```
#define ECHO_MODE_OOKREF 1 /* Select the data slicer reference */
```

ECHO_BAND_VALUE

Description: Selects the frequency band that Echo transmits and receives in.

Values:
ECHO_F 304MHz 304 MHz band
ECHO_F 315MHz 315 MHz band
ECHO_F 434MHz 434 MHz band
ECHO_F 868MHz 868 MHz band
ECHO_F 916MHz 916 MHz band

Example:

```
#define ECHO_BAND_VALUE ECHO_F434MHz
```

ECHO_DATA_RATE

Description: Defines the data rate in bits per second (bps) before Manchester encoding.

Values: Integer in one of the ranges below:
2000–2800
4000–5600
8000–10600
16000–22400

Example:
```
#define ECHO_DATA_RATE 2400 /* Set Echo data rate in Hz */
```

11.2.3 ID and Header set up

ECHO_ID_LENGTH

Description: Selects the length of the IDs that will be sent and received (in bits). If the length is six or less, the least significant bits of `ECHO_ID_VALUE` are placed in Echo’s ID register. If the length is 8 bits, the top 6 bits of `ECHO_ID_VALUE` are placed in the ID register and the full `ECHO_ID_VALUE` is used when calculating the checksum. A full 8 bit ID is sent during transmission. This allows compatibility with Tango and Romeo devices.

Values: 2, 4, 5, 6, or 8 bits.

Example:
```
#define ECHO_ID_LENGTH 8 /* Length of ID (2,4,5,6 or 8 bits) */
```

ECHO_ID_VALUE

Description: The ID value that Echo will recognize. The actual value placed into Echo’s ID register also depends on `ECHO_ID_LENGTH`.

Values: 0x00–0xFF

Example:
```
#define ECHO_ID_LENGTH 8 /* Length of ID (2,4,5,6 or 8 bits)*/
```

ECHO_HEADER_LENGTH

Description: Selects the length of the headers that will be sent and received (in bits). If the length is six or less, the least significant bits of `ECHO_HEADER_VALUE` are placed in Echo’s `HEADER` register.

Values: 1, 2, 4, or 6 bits.

Example:
```
#define ECHO_HEADER_LENGTH 6 /* Length of header (1,2,4,6 bits) */
```

ECHO_HEADER_VALUE

Description: The header value that Echo will recognize. The actual value placed into Echo's HEADER register also depends on ECHO_HEADER_LENGTH (see ECHO_HEADER_LENGTH).

Values: 0x00–0xFF

Example:

```
#define ECHO_HEADER_VALUE 0x16 /* Header word recognized by Echo */
```

11.2.4 On/off control

ECHO_SOE_VALUE

Description: This defines whether the internal strobe oscillator is enabled on Echo.

Values: 0 = disabled
1 = enabled

Example:

```
#define ECHO_SOE_VALUE 1
```

ECHO_RXON_VALUE

Description: Defines the receiver on time when the internal strobe oscillator is used. This value can be calculated from the Echo data sheet.

Values: 1–15

Example:

```
#define ECHO_RXON_VALUE 15
```

ECHO_RXOFF_VALUE

Description: Defines the receiver off time when the internal strobe oscillator is used. This value can be calculated from the Echo data sheet.

Values: 0–7

Example:

```
#define ECHO_RXOFF_VALUE 7
```

11.2.5 Miscellaneous

ECHO_AFFC_VALUE

Description: Specifies the source of configuration for the average filter frequency.

Values: 0 = Filter cutoff frequency set by datarate bits DR[1:0] in CONFIG2 register
1 = Filter cutoff frequency set by bits AFF[1:0] in CONFIG3 register

Example:

```
#define ECHO_AFFC_VALUE 0
```

ECHO_AFF_VALUE

Description: Specifies the average filter cutoff frequency when `ECHO_AFFC_VALUE` is set to 1.

Values: 0 = 0.5 kHz
1 = 1 kHz
2 = 2 kHz
3 = 4 kHz

Example:

```
#define ECHO_AFF_VALUE 0
```

ECHO_EDD_VALUE

Description: Specifies the value of the envelope detector decay rate.

Values: 0 = slow decay
1 = fast decay

Example:

```
#define ECHO_EDD_VALUE 0
```

ECHO_IFLA_VALUE

Description: Specifies the value of thief level attenuation bit.

Values: 0 = no attenuations
1 = 20 db attenuation

Example:

```
#define ECHO_IFLA_VALUE 0
```

11.3 Optional I/O Pins

ECHO_RSSIC

Description: This defines the I/O pin used control Echo’s RSSIC pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_RSSIC PTED_PTED2 /* Define pin used for RSSIC */
```

ECHO_RSSIC_DDR

Description: This defines the data direction bit for the I/O pin used control Echo’s RSSIC pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_RSSIC_DDR PTEDD_PTEDD2
```

ECHO_STROBE

Description: This defines the I/O pin used control Echo’s STROBE pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_STROBE PTED_PTED1 /* Define pin used for Strobe */
```

ECHO_STROBE_DDR

Description: This defines the data direction bit for the I/O pin used control Echo’s STROBE pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_STROBE_DDR PTEDD_PTEDD1
```

ECHO_ENABLEPA

Description: This defines the I/O pin used control Echo’s ENABLEPA pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_ENABLEPA PTED_PTED4 /* Pin used for Power amp enable */
```

ECHO_ENABLEPA_DDR

Description: This defines the data direction bit for the I/O pin used control Echos ENABLEPA pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_ENABLEPA_DDR PTEDD_PTEDD4
```

ECHO_ENABLELNA

Description: This defines the I/O pin used control Echo’s ENABLELNA pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_ENABLELNA PTED_PTED5 /* Define pin used for LNA */
```

ECHO_ENABLELNA_DDR

Description: This defines the data direction bit for the I/O pin used control Echo’s ENABLELNA pin.

Values: Any I/O pin configurable as an output can be used. Use the naming convention specified in the CodeWarrior header file.

Example:

```
#define ECHO_ENABLELNA_DDR PTEDD_PTEDD5
```

12 Adding the Echo Driver to an Application

To add the Echo driver to an application:

1. Copy the files Echo.c and Echo.h to the Sources directory for the project you are using.
2. Add the Echo.c and Echo.h driver files to the project. In CodeWarrior, right click on Sources folder, then select Add Files…
3. Add the line `#include “Echo.h”` to the main application program file.
4. Add the following lines to the main application program to give access to the Echo drivers global storage:

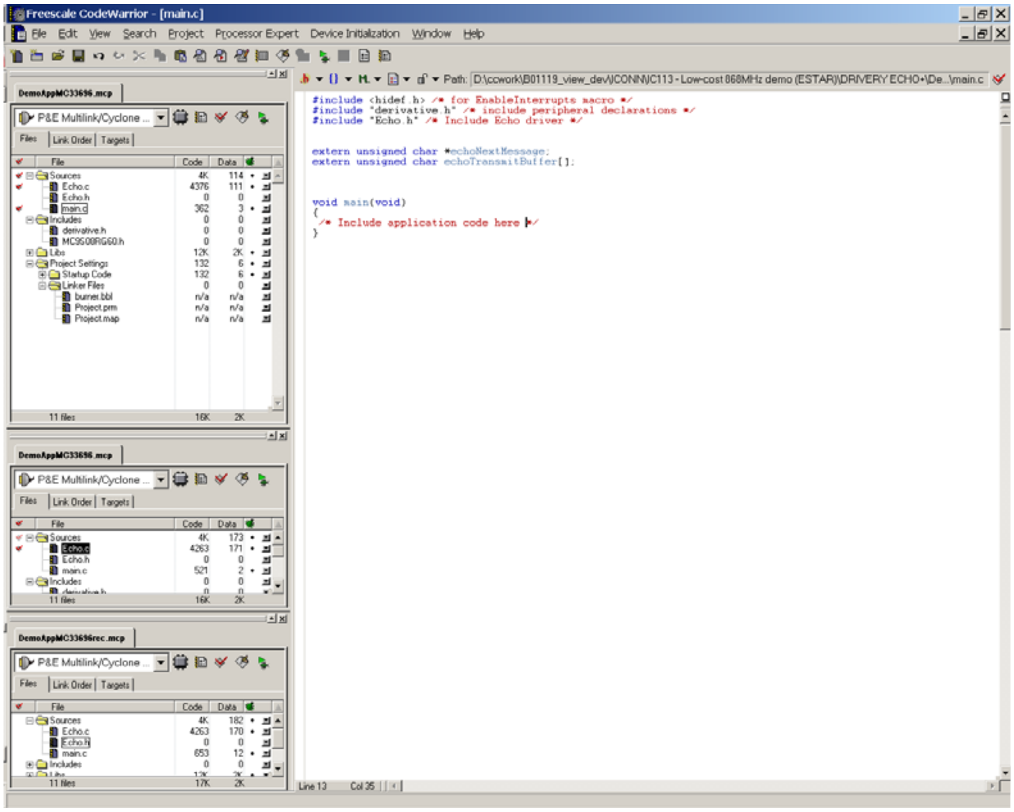
```
extern unsigned char *echoNextMessage;
extern unsigned char echoTransmitBuffer[];
```

5. Decide which I/O pins in your application will control Echo functions. Modify Echo.h to link these pins to the Echo driver.
6. Modify Echo.h to define the other parameters as required by the application.
7. Modify the project parameter file to link the timer channel and SPI to the Echo services `Echo_TxTimer_Interrupt` and `Echo_RxSPI_Interrupt`, respectively.

The files are now added to the project and you are ready to begin using the Echo driver.

Figure 12 shows a simple application template with the Echo.c and Echo.h files added to the sources folder and the necessary additions to main.c.

Figure 13 shows the project parameter file set up to link the Echo services `Echo_TxTimer_Interrupt` and `Echo_RxSPI_Interrupt` to their corresponding interrupt vectors on the RG60.


Freescale CodeWarrior - [main.c]
File Edit View Search Project Processor Expert Device Initialization Window Help
Path: D:\ccwork\B01119_view_dev\CONN\C113 - Low-cost 868MHz demo (ESTAR)\DRIVERY ECHO+\De...\main.c
P&E Multilink/Cyclone ...
Files
Link Order
Targets
Line 13
Col 35

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "Echo.h" /* Include Echo driver */

extern unsigned char *echoNextMessage;
extern unsigned char echoTransmitBuffer[];


void main(void)
{
 /* Include application code here */
}
```

DemoAppMC33696.mcp

File	Code	Data
Sources	4K	114
Echo.c	4376	111
Echo.h	0	0
main.c	362	3
Includes	0	0
derivative.h	0	0
MC9S08RG60.h	0	0
Libs	12K	2K
Project Settings	132	6
Startup Code	132	6
Linker Files	0	0
burner.bbl	n/a	n/a
Project.prm	n/a	n/a
Project.map	n/a	n/a
11 files	16K	2K

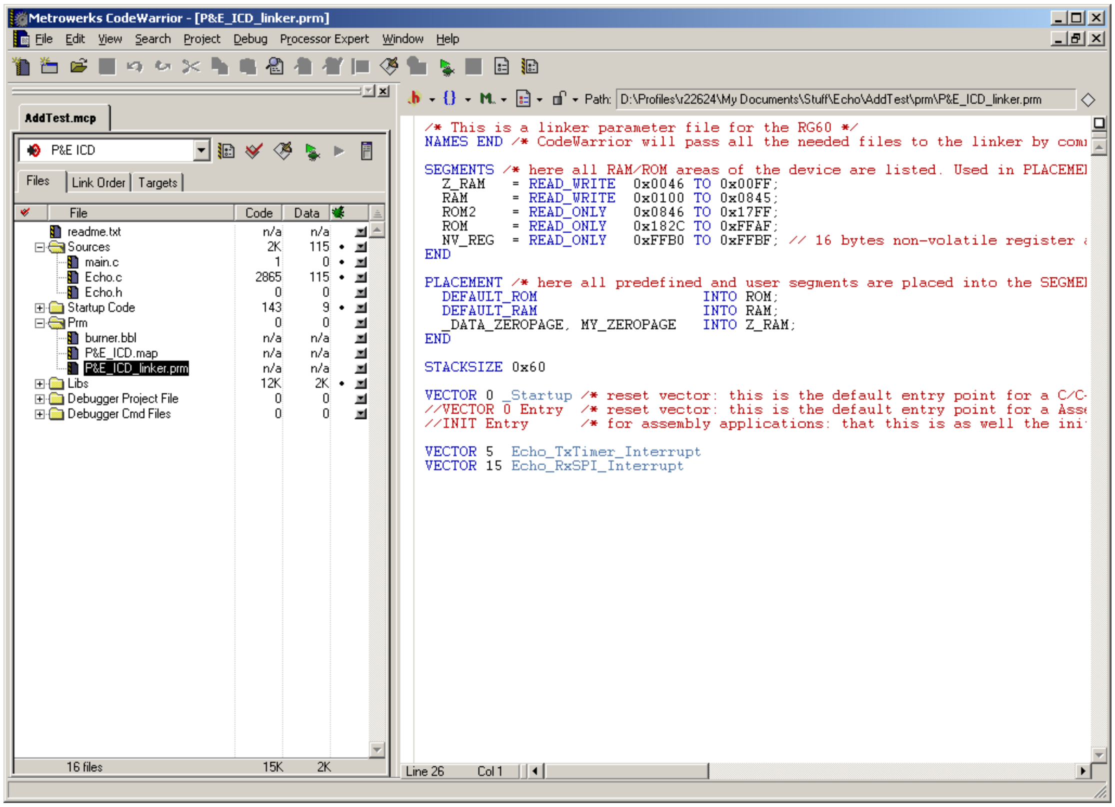
DemoAppMC33696.mcp

File	Code	Data
Sources	4K	173
Echo.c	4263	171
Echo.h	0	0
main.c	521	2
Includes	0	0
derivative.h	0	0
11 files	16K	2K

DemoAppMC33696rec.mcp

File	Code	Data
Sources	4K	182
Echo.c	4263	170
Echo.h	0	0
main.c	653	12
Includes	0	0
Libs	12K	2K
11 files	17K	2K

Figure 12. Application Template Including Echo Files


Metrowerks CodeWarrior - [P&E_ICD_linker.prm]
File Edit View Search Project Debug Processor Expert Window Help
AddTest.mcp
P&E ICD
Files
Link Order
Targets
Path: D:\Profiles\r22624\My Documents\Stuff\Echo\AddTest\prm\P&E_ICD_linker.prm
Line 26
Col 1

File	Code	Data
readme.txt	n/a	n/a
Sources	2K	115 •
main.c	1	0 •
Echo.c	2865	115 •
Echo.h	0	0
Startup Code	143	9 •
Prm	0	0
burner.bbl	n/a	n/a
P&E_ICD.map	n/a	n/a
P&E_ICD_linker.prm	n/a	n/a
Libs	12K	2K •
Debugger Project File	0	0
Debugger Cmd Files	0	0
16 files	15K	2K

```
/* This is a linker parameter file for the RG60 */
NAMES END /* CodeWarrior will pass all the needed files to the linker by comı

SEGMENTS /* here all RAM/ROM areas of the device are listed. Used in PLACEMEı
  Z_RAM    = READ_WRITE  0x0046 TO 0x00FF;
  RAM      = READ_WRITE  0x0100 TO 0x0845;
  ROM2     = READ_ONLY   0x0846 TO 0x17FF;
  ROM      = READ_ONLY   0x182C TO 0xFFAF;
  NV_REG   = READ_ONLY   0xFFB0 TO 0xFFBF; // 16 bytes non-volatile register ‹
END

PLACEMENT /* here all predefined and user segments are placed into the SEGMEı
  DEFAULT_ROM                     INTO ROM;
  DEFAULT_RAM                     INTO RAM;
  _DATA_ZEROPAGE, MY_ZEROPAGE     INTO Z_RAM;
END

STACKSIZE 0x60

VECTOR 0 _Startup /* reset vector: this is the default entry point for a C/C-
//VECTOR 0 Entry  /* reset vector: this is the default entry point for a Asse
//INIT Entry      /* for assembly applications: that this is as well the ini

VECTOR 5  Echo_TxTimer_Interrupt
VECTOR 15 Echo_RxSPI_Interrupt
```

Figure 13. Linker Parameter Files

```
/*****************************************************************************/
/*                          THIS SECTION CONTAINS VALUES YOU MUST DEFINE!         */
/*                                                                                */
#include "derivative.h"                      /* Include peripheral declarations */

/***********************************************/
/*                                             */
/* Common properties for both BANKs in MC33696  */
/*                                            */
/***********************************************/


#ifdef __HCS08__
/* --- Define required pins FOR HCS08!!! ---------------                          */
#define ECHO_CONFB                    PTBD_PTBD6       /* Define pin used for CONFB      */
#define ECHO_CONFB_DDR                PTBDD_PTBDD6

#define ECHO_SEB                      PTBD_PTBD5       /* Define pin used for SEB        */
#define ECHO_SEB_DDR                  PTBDD_PTBDD5

#else

/* --- Define required pins FOR HC08!!! ---------------                            */
#define ECHO_CONFB                    PTD_PTD1        /* Define pin used for CONFB       */
#define ECHO_CONFB_DDR                DDRD_DDRD1

#define ECHO_SEB                      PTD_PTD0        /* Define pin used for SEB         */
#define ECHO_SEB_DDR                  DDRD_DDRD0
```

```
#endif

/* --- Setup SPI --------------------------                                           */
#define ECHO_SPI_ADDRESS              0x28      /* Location of SPI registers             */
                                                /* Address varies from MCU to MCU        */
#define ECHO_SPI_CLOCK_SPEED          8000000   /* SPI clock speed                       */

/* --- Timer set up -----------------------                                              */
#define ECHO_TIMER_ADDRESS            0x40      /* Location of 1st timer register        */
#define ECHO_TIMER_CHANNEL0                 /* Define which timer channel to use     */
                                            /* Note:timer channels start from 0      */

#define ECHO_TIMER_CLOCK_SOURCE1/* Use to set clock source for timer                  */
                                            /* 1 = Bus clock                         */
                                            /* 2 = XCLK- note,not all MCUs have XCLK */
                                            /* 3 = Ext clock                         */


#define ECHO_USE_DATACLK  0                 /* Indicate if the Ext clock is DATACLK   */
                                            /* output from Echo. If using DATACLK     */
                                            /* and ECHO_TIMER_CLOCK_SOURCE = 3,       */
                                            /* ECHO_TIMER_CLOCK_SPEED is replaced     */
                                            /* with Fdataclk calculated instead.      */
                                            /* 0 = dataclock not enabled              */
                                            /* 1 = dataclock enabled                  */

#define ECHO_TIMER_CLOCK_SPEED8000000/* Set timer clock speed in Hz                    */

#define ECHO_TIMER_PRESCALE1                /* Specify timer prescaler value          */

#define ECHO_TIMER_DISABLE1                 /* Allows driver to turn off timer after  */
                                            /* use. Delete this #define if you want   */
                                            /* timer to stay on                       */

#define ECHO_CRYSTAL_FREQUENCY24190660/* Crystal frequency (in Hz)                      */
                                            /* Typical values used                    */
                                            /* RF Output                              */
                                            /* 315MHz  - 17581400                     */
                                            /* 434MHz  - 24190660                     */
                                            /* 868MHz  - 24161390                     */

/* --- Buffering parameters ---------------                                               */
#define ECHO_MAX_DATA_SIZE14                /* Max length of data field in msg        */

#define ECHO_RX_BUF_COUNT4                  /* Number of Rx buffers to allocate       */
                                            /* An additional buffer is allocated      */
                                            /* for internal use by the driver         */
                                            /* Note: each buffer consumes             */
                                            /* ECHO_MAX_DATA_SIZE+2 bytes in memory!  */

/* --- On/off control ---------------------                                               */
#define ECHO_SOE_VALUE    1                 /* 0 = strobe oscillator disabled         */
                                            /* 1 = strobe oscillator enabled          */


#define ECHO_RX_TIMEOUT_BITS32              /* Number of bit-times to wait on         */
```

```
                                            /* the next byte arriving before          */
                                            /* assuming an error and timing out       */
                                            /* back to the ready state (up to 255)    */

/* --- Low Voltage status -------------------                                             */
#define ECHO_LVD_ENABLE   0                 /* Enable low voltage detection           */
                                            /* 0 = Disabled                           */
                                            /* 1 = Enabled                            */

/* --- Misc -------------------------------                                               */
#define ECHO_MCU_BUS_SPEED8000000           /* In Hz                                  */

#define ECHO_SWITCH_LEVEL0                  /* Set active level of SWITCH o/p pin     */
                                            /* 0 = Rx low, Tx high                    */
                                            /* 1 = Tx low, Rx high                    */
#define ECHO_AUTO_SWITCH  1/* if is used both bank, then can switch automatically
                                             to both banks used after each transmit   */
                                            /* 0 = Switching to both banks manually   */
                                            /* 1 = Switching to both banks automatically */
/************************************************/
/*                                              */
/*       Properties for  BANK A in MC33696      */
/*                                             */
/************************************************/

/* --- Echo Bank A Enable ------------                                                     */
#define ECHO_BANK_A_EN     1 /* Switch if you want use  Bank A in MC33696                  */
                                            /* 0 - Bank B disable                      */
                                            /* 1 - Bank B enable                       */

#if ECHO_BANK_A_EN == 1
  /* --- Echo modulation parameters ---------                                              */
  #define ECHO_MODE_VALUE ECHO_FSK          /* ECHO_OOK = OOK reception                */
                                            /* ECHO_FSK = FSK reception                */
  #define ECHO_MODE_DSREF1                  /* Select the data slicer reference in     */
                                            /* OOK mode: 0 = fixed, 1 = adaptive       */
                                            /* Can delete/ignore if using FSK           */

  #define ECHO_BAND_VALUE ECHO_F434MHz/* ECHO_F304MHz, ECHO_F315MHz,                      */
                                            /* ECHO_F434MHz, ECHO_F868MHz,              */
                                            /* or ECHO_F916MHz                          */

  #define ECHO_DATA_RATE  19200             /* Set Echo data rate in Hz (before         */
                                            /* Manchester encoding)                     */

  /* --- ID and Header set up --------------                                                */
  #define ECHO_ID_LENGTH  6                 /* 6 Length of ID (2,4,5,6 or 8 bits)      */
  #define ECHO_ID_VALUE   0xC8              /* 11 ID word recognized by Echo            */

  #define ECHO_HEADER_LENGTH4               /* Length of header (1,2,4,6 bits)          */
  #define ECHO_HEADER_VALUE0x06             /* Header word recognized by Echo (must     */
                                            /* not be present in Preamble/ID)           */

  /* These values are calculated from the Echo datasheet or supporting driver spreadsheet   */
  #define ECHO_RXON_VALUE 2                 /* On time:  1->15                          */
  #define ECHO_RXOFF_VALUE2                 /* Off time: 0->7                           */
  /* --- Misc -------------------------------                                              */
  #define ECHO_AFFC_VALUE  0  /* Average Filter Frequency Control                          */
                              /* 0=Filter cutoff freq set by datarate bits DR[1:0]         */
                              /* 1=Filter cutoff freq set by AFF bits AFF[1:0]             */

  #define ECHO_AFF_VALUE   0                /* Average Filter Frequency                 */
                                            /* Can  ignore if ECHO_AFFC is set to 0     */
                                         /* 0 = 0.5KHz                                     */
                                         /* 1 = 1KHz                                       */
                                         /* 2 = 2KHz                                       */
                                         /* 3 = 4KHz                                       */


  #define ECHO_EDD_VALUE  1              /* Envelope Detector Decay rate control           */
                                         /* Sets decay rate of envelope detector           */
                                         /* 0 = slow decay for minimum ripple              */
                                          /* 1 = fast decay                     */


  #define ECHO_IFLA_VALUE 0              /* IF Level Attenuation                           */
                                         /* 0 = no attenuation                             */
                                         /* 1= 20dB attenuation(in OOK mode only)          */
#endif

/************************************************/
/*                                              */
/*       Properties for  BANK B in MC33696      */
/*                                             */
/************************************************/

/* --- Echo Bank B Enable ------------           */
#define ECHO_BANK_B_EN0 /* Switch if you want use Bank B in MC33696                  */
                          /* 0 - Bank B disable                                            */
                          /* 1 - Bank B enable                                             */

#if ECHO_BANK_B_EN == 1
  /* --- Echo modulation parameters ---------                                           */
  #define ECHO_MODE_VALUE_BECHO_FSK         /* ECHO_OOK = OOK reception            */
                                            /* ECHO_FSK = FSK reception            */
  #define ECHO_MODE_DSREF_B1                /* Select the data slicer reference in */
                                            /* OOK mode: 0 = fixed, 1 = adaptive   */
                                            /* Can delete/ignore if using FSK      */

  #define ECHO_BAND_VALUE_BECHO_F434MHz/* ECHO_F304MHz, ECHO_F315MHz,               */
                                            /* ECHO_F434MHz, ECHO_F868MHz,         */
                                            /* or ECHO_F916MHz                     */

  #define ECHO_DATA_RATE_B  19200           /* Set Echo data rate in Hz (before    */
                                            /* Manchester encoding)                */

  /* --- ID and Header set up ---------------                                           */
  #define ECHO_ID_LENGTH_B6                 /* 6 Length of ID (2,4,5,6 or 8 bits)  */
  #define ECHO_ID_VALUE_B 0xC8              /* 11 ID word recognized by Echo       */

  #define ECHO_HEADER_LENGTH_B4             /* Length of header (1,2,4,6 bits)     */
  #define ECHO_HEADER_VALUE_B0x06           /* Header word recognized by Echo (must */
                                            /* not be present in Preamble/ID)      */
  /* These values are calculated from the Echo datasheet or supporting driver spreadsheet*/
  #define ECHO_RXON_VALUE_B2                /* On time:  1->15                       */
  #define ECHO_RXOFF_VALUE_B2               /* Off time: 0->7                        */

  /* --- Misc -------------------------------                                           */
  #define ECHO_AFFC_VALUE_B  0      /* Average Filter Frequency Control               */
                                    /* 0=Filter cutoff freq set by datarate bits DR[1:0]*/
                                    /* 1=Filter cutoff freq set by AFF bits AFF[1:0]   */

  #define ECHO_AFF_VALUE_B   0              /* Average Filter Frequency              */
                                            /* Can  ignore if ECHO_AFFC is set to 0 */
                                            /* 0 = 0.5KHz                            */
                                            /* 1 = 1KHz                              */
                                            /* 2 = 2KHz                              */
                                            /* 3 = 4KHz                    */


  #define ECHO_EDD_VALUE_B  1               /* Envelope Detector Decay rate control  */
                                            /* Sets decay rate of envelope detector  */
                                            /* 0 = slow decay for minimum ripple     */
                                            /* 1 = fast decay        */


  #define ECHO_IFLA_VALUE_B 0               /* IF Level Attenuation                  */
                                            /* 0 = no attenuation                    */
                                            /* 1= 20dB attenuation(in OOK mode only)*/
#endif

/*****************************************************************************************/
/* These may be omitted depending on the hardware setup                                 */
#ifdef __HCS08__
 /* Edit this part if you have project with HCS08 mcu */



//#define ECHO_RSSIC         PTED_PTED2        /* Define pin used for RSSIC            */
//#define ECHO_RSSIC_DDR     PTEDD_PTEDD2      /* If hardwired,delete #defines         */

//#define ECHO_STROBE        PTAD_PTAD3        /* Define pin used for Strobe           */
//#define ECHO_STROBE_DDR    PTEDD_PTEDD1      /* If hardwired,delete#defines          */

/* These are required for use with Motorola's rf modules                               */
//#define ECHO_ENABLEPA      PTED_PTED4        /* Define pin used for Power amp enable */
//#define ECHO_ENABLEPA_DDR  PTEDD_PTEDD4      /* If hardwired, delete #defines        */

//#define ECHO_ENABLELNA     PTED_PTED5        /* Define pin used for LNA              */
//#define ECHO_ENABLELNA_DDR PTEDD_PTEDD5      /* If hardwired,delete #defines         */

#else
 /* Edit this part if you have project with HC08 mcu */

#define ECHO_RSSIC           PTD_PTD2          /* Define pin used for RSSIC            */
#define ECHO_RSSIC_DDR       DDRD_DDRD2        /* If hardwired, delete #defines        */

#define ECHO_STROBE          PTA_PTA3          /* Define pin used for Strobe           */
#define ECHO_STROBE_DDR      DDRA_DDRA3        /* If hardwired,delete #defines         */
```

```
/* These are required for use with Motorola's rf modules                             */
//#define ECHO_ENABLEPA      PTED_PTED4        /* Define pin used for Power amp enable */
//#define ECHO_ENABLEPA_DDR  PTEDD_PTEDD4      /* If hardwired, delete #defines        */

//#define ECHO_ENABLELNA     PTED_PTED5        /* Define pin used for LNA              */
//#define ECHO_ENABLELNA_DDR PTEDD_PTEDD5      /* If hardwired,delete #defines         */

#endif
/*****************************************************************************************/
```

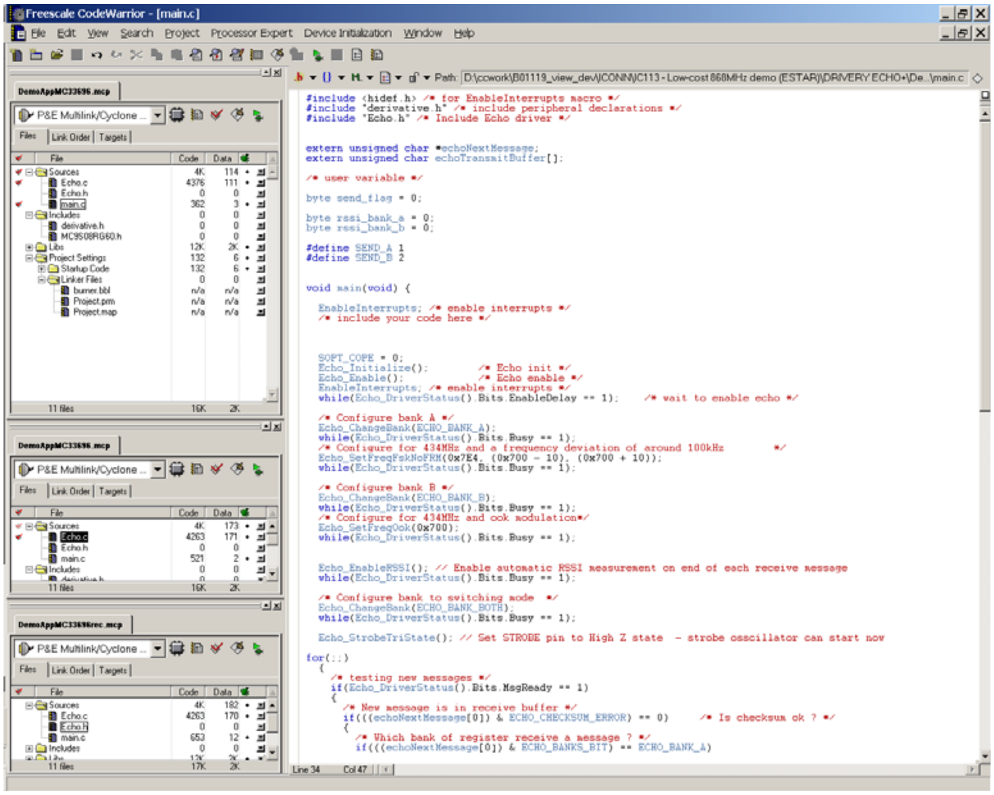
Figure 14. Example Echo.h header file

13 Example Echo Application

This example uses the Echo.h configuration shown in Figure 14 and, as such, has been configured to run on the DEMO9S08RG60 demonstration board.

The example toggles an LED when a message is sent or received and allows messages to be transmitted by pressing a switch. The message consists of two data bytes, the first of which is incremented with each message sent and the second remains constant at 1. The example is shown in Figure 15.

This example can be placed on multiple Echo devices without change. Pressing the switch on one device will toggle the LED’s on all others.



```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */
#include "Echo.h" /* Include Echo driver */


extern unsigned char *echoNextMessage;
extern unsigned char echoTransmitBuffer[];

/* user variable */

byte send_flag = 0;

byte rssi_bank_a = 0;
byte rssi_bank_b = 0;

#define SEND_A 1
#define SEND_B 2


void main(void) {

  EnableInterrupts; /* enable interrupts */
  /* include your code here */



  SOPT_COPE = 0;
  Echo_Initialize();          /* Echo init */
  Echo_Enable();              /* Echo enable */
  EnableInterrupts; /* enable interrupts */
  while(Echo_DriverStatus().Bits.EnableDelay == 1);     /* wait to enable echo */

  /* Configure bank A */
  Echo_ChangeBank(ECHO_BANK_A);
  while(Echo_DriverStatus().Bits.Busy == 1);
  /* Configure for 434MHz and a frequency deviation of around 100kHz        */
  Echo_SetFreqFskNoFRM(0x7E4, (0x700 - 10), (0x700 + 10));
  while(Echo_DriverStatus().Bits.Busy == 1);

  /* Configure bank B */
  Echo_ChangeBank(ECHO_BANK_B);
  while(Echo_DriverStatus().Bits.Busy == 1);
  /* Configure for 434MHz and ook modulation*/
  Echo_SetFreqOok(0x700);
  while(Echo_DriverStatus().Bits.Busy == 1);


  Echo_EnableRSSI(); // Enable automatic RSSI measurement on end of each receive message
  while(Echo_DriverStatus().Bits.Busy == 1);

  /* Configure bank to switching mode  */
  Echo_ChangeBank(ECHO_BANK_BOTH);
  while(Echo_DriverStatus().Bits.Busy == 1);

  Echo_StrobeTriState(); // Set STROBE pin to High Z state  - strobe osscillator can start now

for(;;)
  {
    /* testing new messages */
    if(Echo_DriverStatus().Bits.MsgReady == 1)
    {
      /* New message is in receive buffer */
      if(((echoNextMessage[0]) & ECHO_CHECKSUM_ERROR) == 0)     /* Is checksum ok ? */
      {
        /* Which bank of register receive a message ? */
        if(((echoNextMessage[0]) & ECHO_BANKS_BIT) == ECHO_BANK_A)
```

Figure 15. Example Echo Application

14 Communicating with Romeo2 and Tango3 Devices

The major difference between Romeo2, Tango3, and Echo is the ID and header that is by Echo and Romeo's data managers. This can affect communication between the devices.

14.1 ID Length Mismatch

There is a mismatch between the ID lengths expected by the data managers in Romeo2 and Echo. Echo has a variable length ID and header, whereas Romeo2 has a fixed 8 bit ID and 4 bit, fixed format, header, either 0110 or 1001.

The header for Echo can be setup for 4 bits and set to 0110 or 1001 using the configuration options in the Echo header file. The ID on Echo is limited to 2, 4, 5, or 6 bits in length and thus not directly compatible with Romeo2's data manager and the Tango3 device driver.

There are two cases which need to be considered: Echo is transmitting to Romeo2 and Echo is receiving from Tango.3

14.1.1 Case (i) – Echo Transmitting

When Echo is transmitting, configure the driver to send 8 bits for the ID and thus allow Romeo2 to be addressed. The driver has complete control of the exact bits sent and can send 8 bits at the ID stage. Sending 8 bits can be achieved by setting `ECHO_ID_LENGTH` in Echo.h to 8: this sends all 8 bits of the ID in the transmit buffer.

14.1.2 Case (ii) – Echo Receiving

The Tango3 driver transmits a fixed length 8 bits in the ID sequence. Echo can receive messages with 8 bit ID fields. Any 6 bits from the 8 transmitted may be matched by the Echo data manager to the 6 bits in the ID register. For this reason, choose IDs that are unique for all devices (if desired).

14.2 Compatible Set Up

To be compatible, several things must be configured correctly:

- The headers must be same
  - Echo must be set up for 4 bits: 0110
  - Romeo must be set up for header detection (HE = 1)
  - Tango must send using the header format (`TangoSendPreamble_ID` and `TangoSendData`)
- The data rates must be the same, between 2 kbps and 10 kbps
- The modulation type must be the same, either OOK or FSK
- The transmission and reception frequencies must be the same: Echo must be set up to ensure transmission and reception on exactly 315 MHz, 434 MHz, etc.
- Echo must be set up with `ECHO_ID_LENGTH` as 8 bits

Using a common set of features and compensating for the ID length mismatch, Echo can communicate with Romeo and Tango, employing their existing drivers.

The example configuration in Figure 13 and application in Figure 14 allow communication with Romeo devices and are capable of receiving communications from Tango3 devices. Tango3 can then communicate with Echo by sending to ID 0x55. Likewise, Romeo2 can receive messages from Echo with an ID of 0x55.

Full source code for this example application is supplied with this document.

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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

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

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

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

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

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

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

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

freescale™
semiconductor