UART Implementation Using the High-End Timer

Andreas Dannenberg

ABSTRACT

This application report describes how the TMS470 high-end timer (HET) peripheral can be used to implement multiple additional full-duplex high-speed universal asynchronous receiver transmitter (UART) interfaces with interrupt capability and zero ARM7 CPU overhead. The solution is provided for both TMS470R1A256 and TMS470R1B1M devices, but the information applies to other TMS470 family members as well. The sample code described in this application report can be downloaded from http://www.ti.com/lit/zip/SPNA097.

1 Overview

The TMS470 HET peripheral is a complex high-performance RISC coprocessor that operates independently from the main ARM7 CPU and can be used to implement complex timed I/O operations running in the background. More information on the HET can be found in the *TMS470R1x High-End Timer (HET) Reference Guide*.[1]

In this application report, the HET is used to implement multiple UART interfaces. The goal is to provide hardware UART-module-like functionality with independent background transmission/reception, indications that a character was received or sent, and interrupts, allowing for an event-oriented application program flow. Figure 1 shows a conceptual overview of the interaction among the different components.

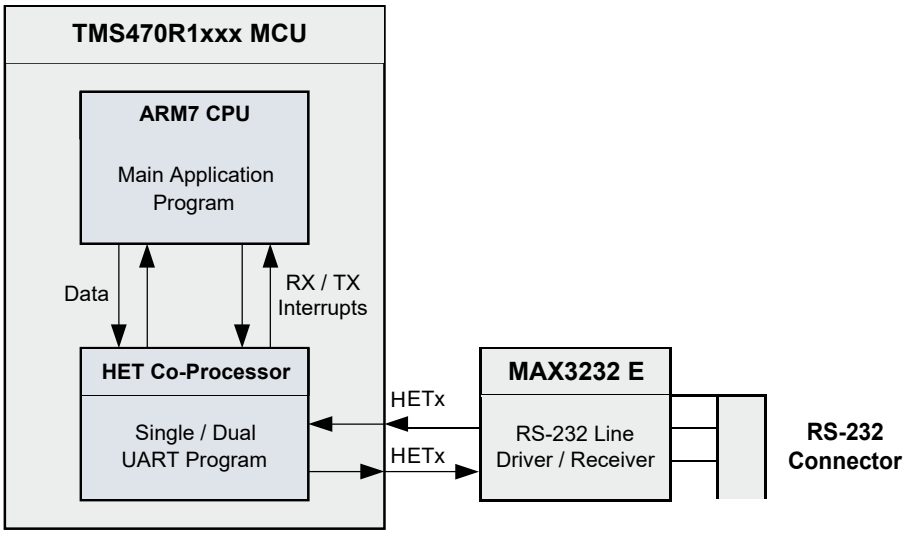

Figure 1. TMS470 HET UART Implementation Overview

The HET program runs in the background, independent of the main ARM7 CPU, and performs all the tasks associated with an asynchronous UART communication. The incoming UART data stream delivered to an HET device I/O pin is decoded. Upon the reception of a full character, the ARM7 program is notified, and it then can directly fetch the entire received data byte from the HET internal RAM. For outgoing data streams, the ARM7 just passes the data it wishes to transmit to the HET program and initiates the transmission. After this, the entire transmission is handled by the HET program, and a UART data stream is output to an HET device I/O pin. During this process, the ARM7 CPU is free for application-related tasks and does not perform any critical low-level UART-related timings.

2 Theory of Operation

In this UART implementation, the HET program processes the incoming and outgoing data streams bit by bit. For this, conditional HET program flow is used, which means that only one bit can be processed for each UART channel within one HET program loop. The HET UART program must adhere to the general rule that the entire HET program must be processed within one loop resolution (LR) clock period, while leaving enough time slots for the ARM7 to access the HET RAM. The HET UART code presented in this application report consists of independent entities that implement transmission and reception. These building blocks can be combined to create any number of transmission or reception channels as required by the main application, as long as the HET requirements are met. See the *TMS470R1x High-End Timer (HET) Reference Guide* for more details regarding HET operation.[1]

For the transmission process, the LR clock period must be equal to or less than the target bit time. A transmission can be started any time, due to the asynchronous nature of the UART protocol. Changes to the signal line are required only at the end of every bit period. Figure 2 shows a UART transmission of one start bit, eight data bits, no parity bit, and one stop bit. Each output signal transition is preloaded into the HET pin logic in advance and becomes active at the beginning of the next LR cycle. For this example, a total of ten transitions is needed and preloaded. Note that the falling edge for the start bit is generated in the next LR cycle directly after transmission start.

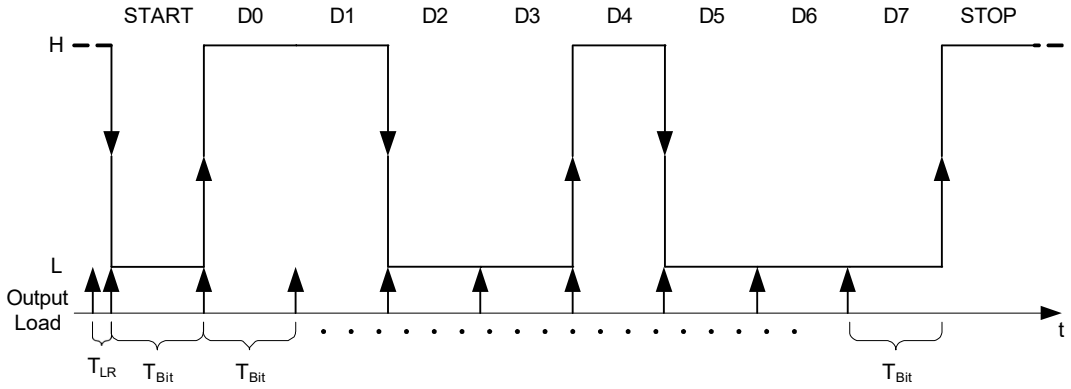

Figure 2. HET UART Transmission Operating Principle

For the reception process, the operational principle is that the HET UART polls the receive signal line with a frequency higher than the Baud rate until a zero logic level (start bit) is detected. This is necessary, as the incoming data is asynchronous to the HET timing. The value of the first data bit is then determined 1.5 bit periods after the start bit was detected, which is in the middle its bit period. Then, all other bits are read one bit period, T_{Bit}, apart from each other. For this process to work, a rather fine time resolution is needed to minimize errors due to the start-bit detection principle. For example, with an LR clock frequency used for start-bit detection that is eight times the target Baud rate, the maximum possible offset error introduced by this method is 12.5%. This means that the sample points for the actual data bits are off center by a maximum of 12.5% of a bit time. Figure 3 shows the timing diagram of a UART reception with one start bit, eight data bits, no parity, and one stop bit. In this case, a total of nine samples are taken after the start-bit high-to-low transition is detected.

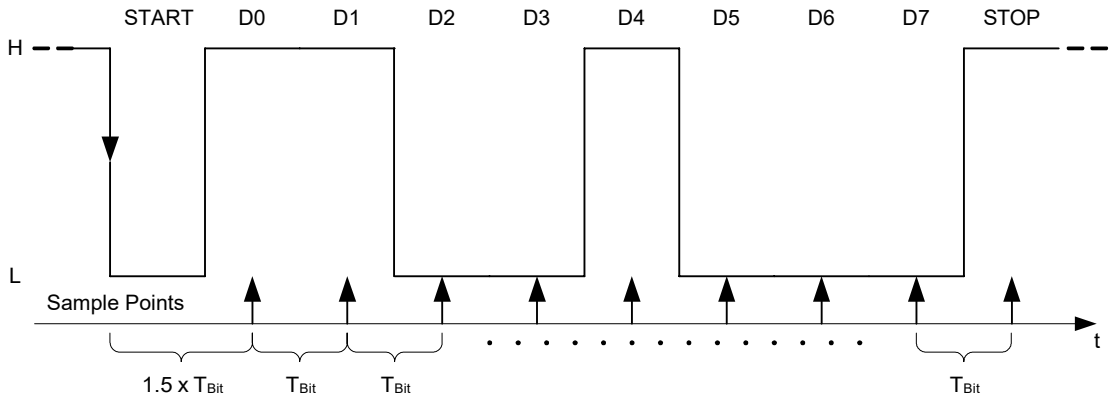

Figure 3. HET UART Reception Operating Principle

3 Software Description

3.1 Overview

The software package is provided for two different TMS470 devices, each implementing two different UART configurations. Both interrupt- and polling-driven program flow also are demonstrated. Note that all of the HET and ARM7 codes almost are identical and only differ in terms of TMS470 and UART timing configuration. Table 1 provides an overview of the provided source code files and their respective functions. In order to build and run the example code, a new project in IAR Embedded Workbench™ using one of the TI-provided TMS470-specific project templates must be created. This ensures that the HET assembler and the associated custom build steps are executed properly. Then, the HET source code file (*.het) together with either the interrupt- or polling-driven C application file that is based on the same configuration must be added. For example, to build a project that implements a dual high-speed 115,200-baud UART on a TMS470R1A256 device, the following two files must be added to the project:

- A256_HET_UART_2X115200_H.het
- A256_HET_UART_2X115200_Int.c

To fulfill the requirements that an HET program must meet regarding available number of time slots while minimizing bit-timing errors, different HET clock configurations have been selected, depending on the operating case (Table 2). In most cases, the default crystal oscillator that is populated on the respective TI development kit is utilized.[2][3] However, in the case of the TMS470R1A256 implementing two 115,200-baud high-speed UARTs, a special 11.0592-MHz crystal was selected to achieve the lowest possible bit-timing errors.

Table 1. Software Overview

FUNCTION	TMS470R1xxx, CRYSTAL FREQUENCY	SOURCE FILE NAME	DESCRIPTION
Single full-duplex UART, 9600 baud	A256, 12 MHz	A256_HET_UART_9600_H.het	HET source code implementing a single full-duplex 9600-baud UART
		A256_HET_UART_9600_H.c	Output file created by HET assembler
		A256_HET_UART_9600_H.h	Output file created by HET assembler
		A256_HET_UART_9600_Int.c	TMS470 program demonstrating the use of the HET code by implementing an interrupt-driven UART echo
		A256_HET_UART_9600_Polling.c	TMS470 program demonstrating the use of the HET code by implementing a UART echo in polling mode
	B1M, 7.5 MHz	B1M_HET_UART_9600_H.het	HET source code implementing a single full-duplex 9600-baud UART
		B1M_HET_UART_9600_H.c	Output file created by HET assembler
		B1M_HET_UART_9600_H.h	Output file created by HET assembler
		B1M_HET_UART_9600_Int.c	TMS470 program demonstrating the use of the HET code by implementing an interrupt-driven UART echo
		B1M_HET_UART_9600_Polling.c	TMS470 program demonstrating the use of the HET code by implementing a UART echo in polling mode
Dual full-duplex UART, 115,200 baud	A256, 11.0592 MHz	A256_HET_UART_2X115200_H.het	HET source code implementing two full-duplex 115,200-baud UARTs
		A256_HET_UART_2X115200_H.c	Output file created by HET assembler
		A256_HET_UART_2X115200_H.h	Output file created by HET assembler
		A256_HET_UART_2X115200_Int.c	TMS470 program demonstrating the use of the HET code by implementing two interrupt-driven UART echos
	B1M, 7.5 MHz	B1M_HET_UART_2X115200_H.het	HET source code implementing two full-duplex 115,200-baud UARTs
		B1M _HET_UART_2X115200_H.c	Output file created by HET assembler
		B1M _HET_UART_2X115200_H.h	Output file created by HET assembler
		B1M _HET_UART_2X115200_Int.c	TMS470 program demonstrating the use of the HET code by implementing two interrupt-driven UART echos

Table 2. HET Clock Selection

TMS470R1xxx, CRYSTAL FREQUENCY	f_{SYSCLK} (MHz)	f_{HR} (MHz)	f_{LR} (kHz)	TIME SLOTS AVAILABLE PER LR	UART SPEED (BAUD)	LR CYCLES PER UART BIT
A256, 12 MHz	48	12	375	127	9600	39.06
B1M, 7.5 MHz	60	15	468.75	127	9600	48.83
A256, 11.0592 MHz	44.2368	14.7456	921.6	47	115,200	8
B1M, 7.5 MHz	60	30	937.5	63	115,200	8.14

Information regarding device-specific HET pin connections can be found in the respective TMS470 device data sheets.[4][5]

3.2 Data Reception

The following paragraphs discuss the HET and the ARM7 program flow of the UART implementation. Note that when it is indicated that data is stored at or loaded from an HET program location, this always refers to the data field that is embedded in the instruction.

The HET program constantly polls the receive signal line with a frequency higher than the baud rate. The HET program implementation of the RX routine is shown in Figure 4. The data field of RX_Start is used to determine if bits should be received (RX_Start != 1), or a start bit should get detected (RX_Start = 0). In the case of start-bit detection, the logic level of the receive pin is checked. If it is high, the HET routine exits. If the logic level is low, this indicates that a new start bit is detected. Upon start-bit detection, the variables RX_PrdCtr, RX_BitCtr, and RX_Start, which are needed during reception, are initialized. RX_PrdCtr is used to count the number of HET LR clock periods until the next action needs to take place and is decremented on each receive program iteration. In order to capture the first data bit, it is loaded with an LR count value equal to 1.5 UART bit times. When PrdCtr counts down to zero, the logic level captured on the receive pin is shifted into RX_Shift and the receive bit counter (RX_BitCtr) is decremented. Note that a shift-capable dummy pin is used to implement the bit-shift functionality. After storing the current state of the RX signal, RX_PrdCtr is loaded with a count value equal to one UART bit time to prepare for the capture of the next incoming data bit. This process continues until RX_BitCtr has reached zero. Then, the received data is transferred from RX_Shift into RX_Buffer, and an ARM7 interrupt is generated. This also implements a double-buffering scheme, effectively allowing the HET program to receive a byte while one is waiting to be read out by the application program.

The main program reads out the received value directly from the data field of RX_Buffer. This is demonstrated in the function HetUARTGetByte(). As an alternative, in case a polling-driven program flow is desired, it is possible to access RX_Buffer directly to see if a new data byte was received. For this, the ARM7 function HetUARTRxDataAvailable() is provided. It checks whether a valid stop bit has been received, which itself indicates that a properly framed UART sequence was received. In this case, the function returns a non-zero value.

The UART data reception function takes 15 words of HET program memory. A minimum of 2 and a maximum of 11 (worst case) HET time slots (SYSCLK cycles) are consumed during execution. The eighth HET instruction (as seen from the beginning of the code fragment) is generating the UART receive interrupt. See the *TMS470R1x High-End Timer (HET) Reference Guide* for more information regarding the HET interrupt handling.[1]

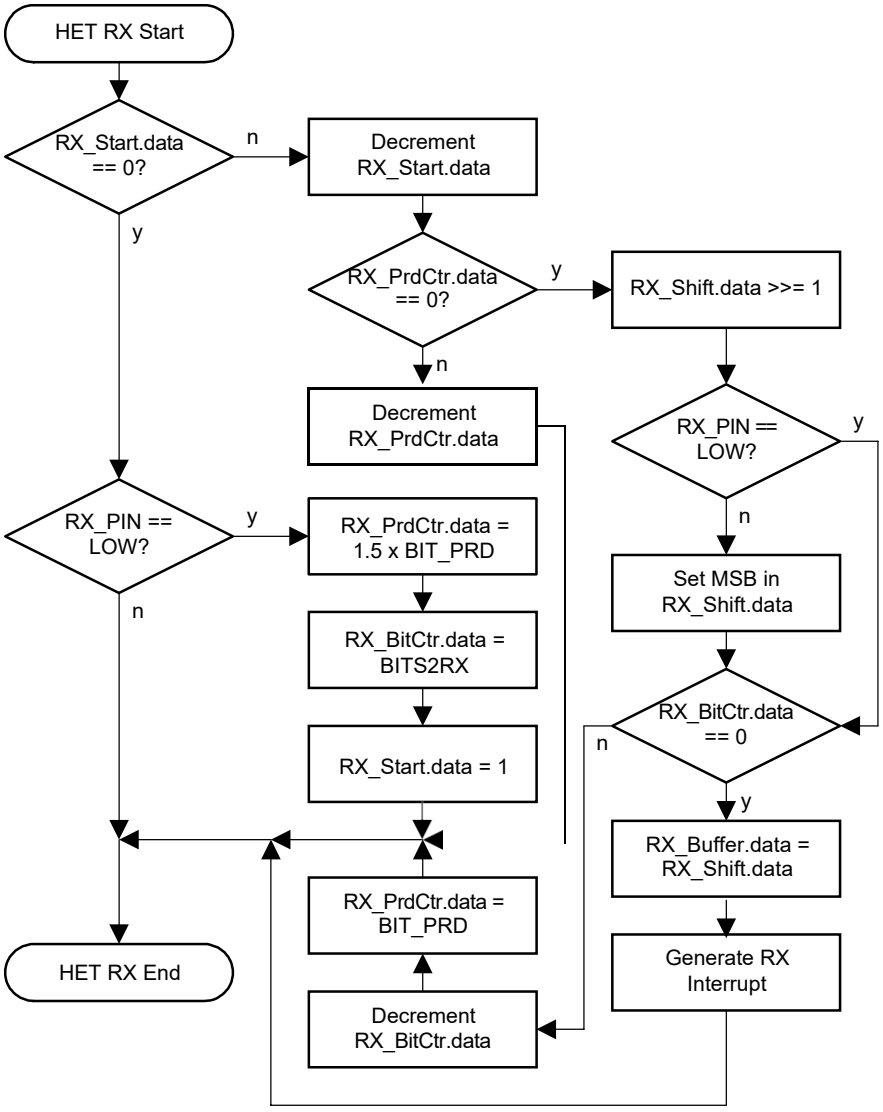

Figure 4. HET Receive Program Flow

3.3 Data Transmission

A UART transmission is initiated by the ARM7 program by calling the function HetUARTPutByte(). This function first adds a zero start bit and a one stop bit to the data to be sent. This value then gets loaded into the data field of the TX_Shift HET instruction, which serves as a transmit buffer. Then, the number of bits to be transmitted is loaded into TX_Bit. In the case of the attached example, this is ten, as the communication scheme used is one start bit, eight data bits, no parity bit, and one stop bit. The parameters that are loaded can easily be adapted to custom requirements to accommodate different data bit lengths or additional stop and parity bits. After the communication data is loaded, the actual transmission which will be performed by the HET program is started by loading one into TX_Start. The ARM7 function is left and the CPU can continue to process other tasks.

The HET data transmission routine is only active once a transmission has been initiated (see Figure 5). TX_Start is used to determine if the TX routine should be executed (TX_Start != 0). Upon execution, TX_PrdCtr is used to count LR cycles until the next transceiver bit is to be processed. When TX_PrdCtr has reached zero, the transmit bit counter TX_BitCtr is checked. A non-zero value indicates that there are still bits to process. If so, TX_BitCtr is decremented, and the LSB of TX_Shift is shifted into the HET Z-flag. Depending on the Z-flag value, the HET UART transmit pin is either set (Z = 1) or reset (Z = 0). This value is latched internally and appears on the output pin with the next HET LR cycle. After the output bit is written, TX_PrdCtr is loaded with an LR count value equal to one UART bit time and used to trigger the next transmit operation. As a last step, TX_Start is set to one so the transmit code is executed again with the next HET LR cycle. In the case that all bits are processed, (TX_BitCtr = 0), an HET interrupt is generated and the TX_Start is not loaded with one. The interrupt indicates to the ARM7 program that all bits were transmitted and that new data can be loaded into the transmit buffer. If polling-mode operation is desired, the data field of TX_BitCtr can be read out directly from the ARM7 program. If it is zero, it is safe to initiate a new transmission. An ARM7 function called HetUARTTxBufferEmpty() is provided to facilitate this.

One channel of UART data transmission functionality consumes ten words of HET program memory. A minimum of one and a maximum of nine (worst case) HET time slots are used, depending on the program flow. The transmit buffer ready interrupt is generated by the second instruction, as seen from the beginning of the code fragment.

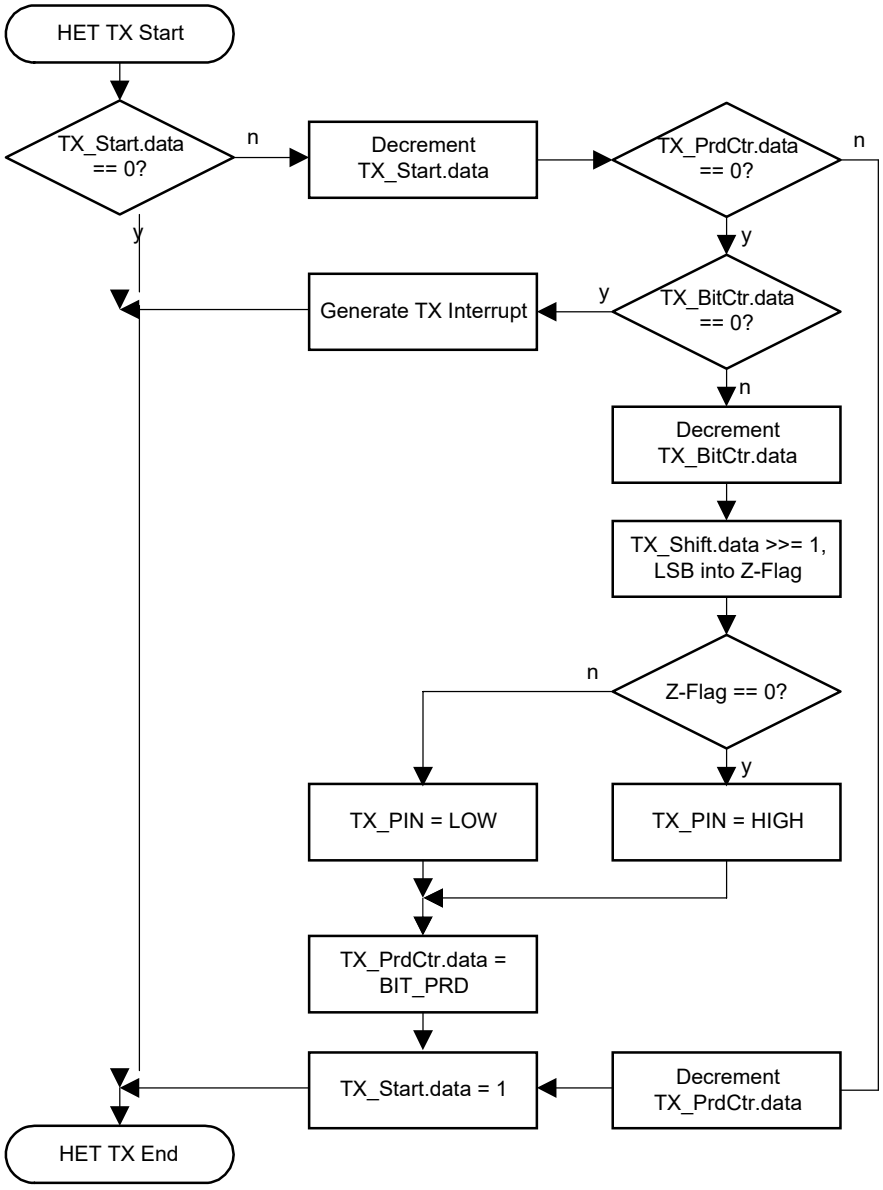

Figure 5. HET Transmit Program Flow

4 References

1. *TMS470R1x High-End Timer (HET) Reference Guide* (SPNU199)
2. *TMS470R1A256 Kickstart Development Kit* (TI Part Number: TMS-FET470A256)
3. *TMS470R1B1M Kickstart Development Kit* (TI Part Number: TMDS-FET470R1B1M)
4. *TMS470R1A256 Microcontroller* data sheet (SPNS100)
5. *TMS470R1B1M Microcontroller* data sheet (SPNS109)

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265