Application Report
SPNA082–July 2005

I2C Communication with the TMS470R1x

John Fahrenbruch — *Automotive Embedded Controls*

ABSTRACT

This application note describes how to configure the internal registers of a TMS470R1x (TMS470) device for setting up inter-integrated communication (I2C) with an industry standard 24LC00 I2C memory device. This application report assumes the reader is familiar with the I2C protocol and is meant to supplement the *TMS470 Inter-Integrated (I2C) Reference Guide* (literature number SPNU223) and the device-specific data sheet, which may be obtained from Texas Instruments (TI).

Contents

List of Figures

List of Tables

1 Clock Registers

The I2C specification states two bit rates (100 bps and 400 bps) for successful operation of I2C devices. The TMS470 has an elaborate internal register-controlled clocking system that offers complete control over the I2C clocking scheme. Figure 1 illustrates the internal clock division for the SCL clock signal.

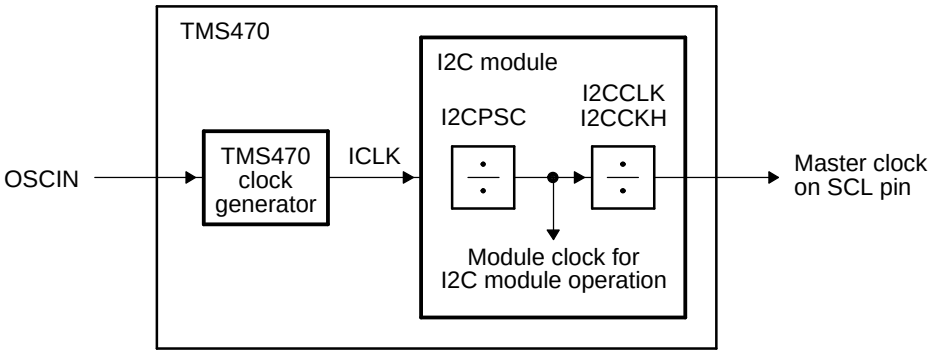

Figure 1. Clocking Diagram for the I2C Module

The OSCIN input is the external oscillator input to the TMS470 into the clock generation unit. The ICLK frequency can be numerically determined using the following formulas:

```
f(sys)= M x f(osc) / R, where M = {4 or 8} and R = {1,2,3,4,5,6,7,8} when PLLDIS = 0    Equation 1
f(sys)= f(osc) / R, where R = {1,2,3,4,5,6,7,8} when PLLDIS = 1.                        Equation 2
f(ICLK) = f(sys) / X, where X = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}.              Equation 3
```

PLLDIS is an input pin on the TMS470 and can be set externally. See section 4 in *TMS470R1x Zero-Pin Phase-Locked Loop (ZPLL) Clock Module Reference Guide* (literature number 212) for a detailed description of R and M. To configure the global control register (GCR), values for R and M have been chosen such that R = 1 for divide-by-2 and M = 1 for a multiply-by-4. Therefore, the lower nibble (GCR(3–0)) is set equal to 0x9h.

For PLLDIS = 0 and selecting an oscillator clock of 10 MHz, the frequency of the system clock can be determined to be:

```
f(sys) = 4 * 10MHz / 2 = 20MHz
```

To determine the frequency of the ICLK, the value of X (Equation 3) must be determined. The peripheral clock register (PCR) is described in the *TMS470R1x System Module Reference Guide* (literature number SPNU189). Bits PCR(4–1) are set according to the desired clock division. For the purpose of this example, a divide-by-2 will be used and the PCR(4–1) bits will be set to 0001. From Equation 3, the ICLK frequency can now be determined with X = 2.

```
f(ICLK) = 20 MHz/2 = 10MHz
```

The divide values for the module clock must now be calculated to meet the following constraint:

```
6.7 MHz ≤ Module Clock ≤ 13.3 MHz
```

Good design practice would be to divide the clock to be somewhere in the middle of the design constraint and not push the envelope to the endpoints. Because of this consideration and because ICLK = 10 MHz, a divide-by-1 was chosen. The frequency of the module clock can be determined from the following formula:

```
ModuleClockFrequency = ICLK/(I2CPSC + 1)                    Equation 4
```

I2CPC is an eight-bit prescale register located at offset 0x30. Clearing the I2CPC register to equal 0x00 will set the module clock equal to the ICLK (refer to Figure 1). The module clock frequency is calculated to be:

```
ModuleClockFrequency = 10 MHz/(0+1) = 10MHz
```

The I2CCKL and I2CCKH registers are used to further divide the module clock frequency and set the duty cycle of the serial clock (master clock) on the SCL pin. The I2CCKL and I2CCKH registers can be used to change the duty cycle of the serial clock. To achieve a 50% duty cycle, the I2CCKl and I2CCKH registers must be equal. The following equation is solved for I2CCLK (I2CCK = I2CCKL = I2CCKH). In the following equation d is defined by Table 1.

Table 1. Equation D

I2CPSC	*d*
0	7
1	6
Greater than 1	5

```
SCL = ModuleClockFrequency / [(I2CCKL + d) + (I2CCKH + d)]
50%dutcycycle -> I2CCK = i2CCKL = I2CCKH
therefore,
                                                        Equation 5
I2CCK = [f(osc)/(2xDataRate)] - 7
I2CCK = [10MHz/(2x100KHz0] - 7
I2CCK = 43
```

Therefore, loading registers I2CCKL and I2CCKH with 0x2Bh (43 decimal) will yield a bit rate of 100 bps.

2 Write Sequence

The flow chart in Figure 2 describes the event sequence for an I2C write and verify. The associated gel script can be found in Section 4. Polling is used to determine when an event occurred before moving on to the next task. Many I2C devices allow for sequential access by internally incrementing an address pointer once initialized. This can be easily implemented with the TMS470 by appropriately changing the number of bytes transmitted and placing the transmit/receive code inside a looping structure.

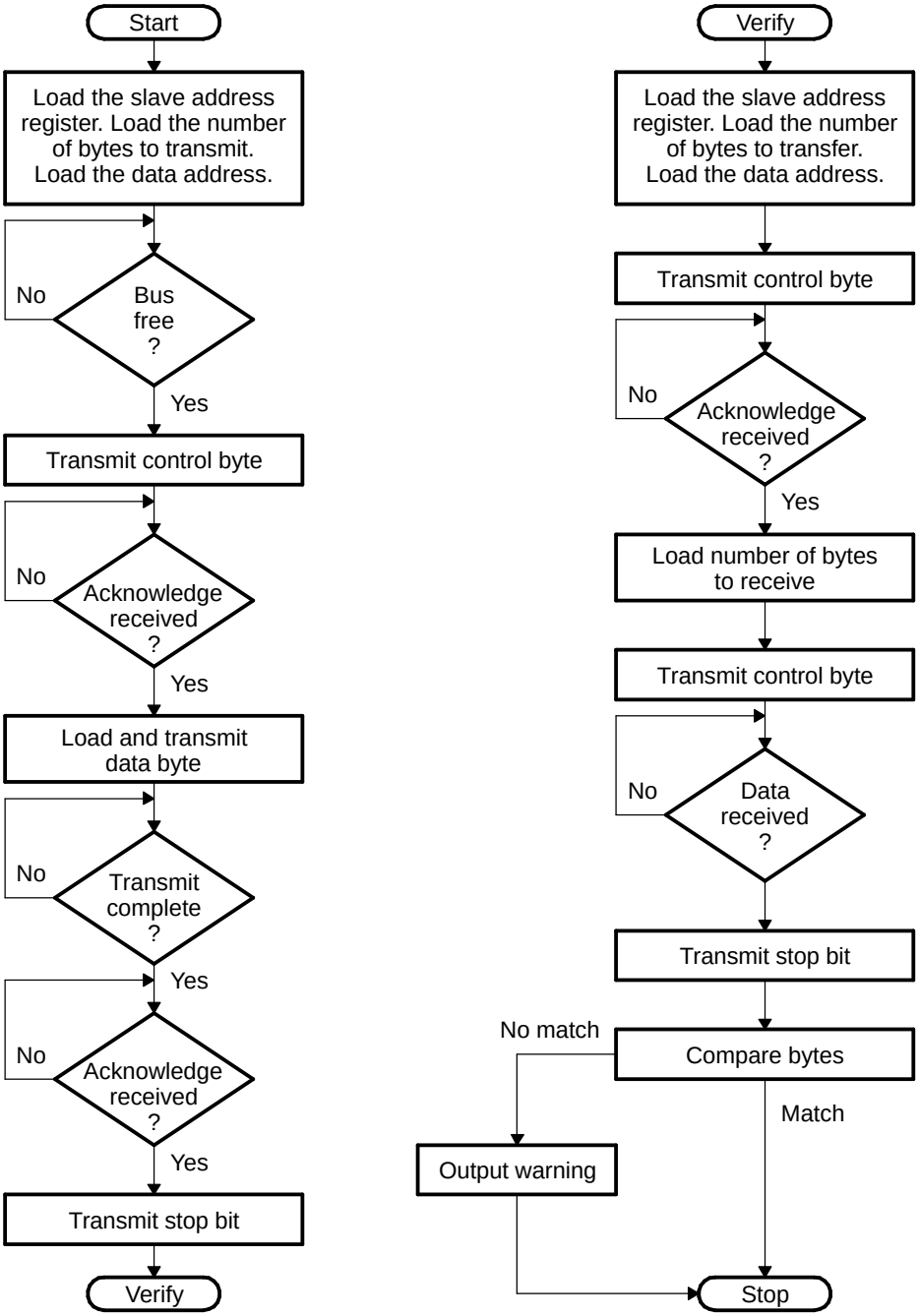

Figure 2. Write Flow Chart

3 Read Sequence

The read sequence is similar to the write sequence and is shown in Figure 3. The associated gel script can be found in Section 4.

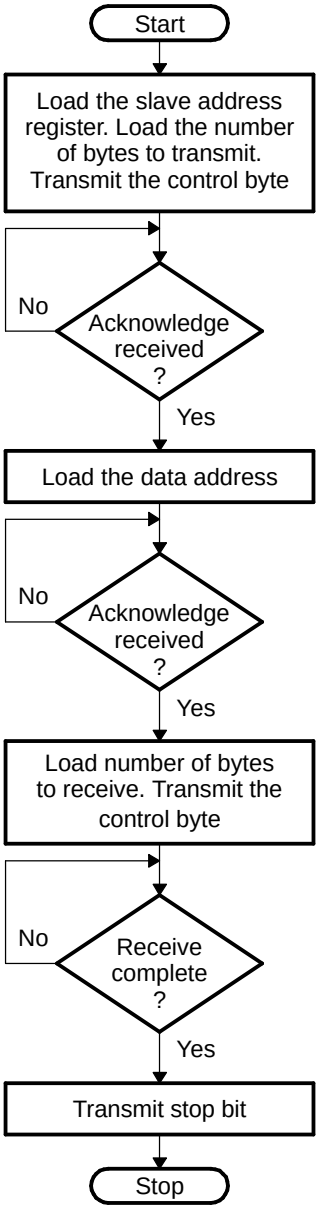

Figure 3. Read Flow Chart

4 Gel Script

```
/********************************************************************** */
/*                                                                      */
/* This gel file tests the functionality of the I2C port 1 for a        */
/* Texas Instruments TMS470R1x by performing a read/write-verify to an */
/* industry standard 24LC00 I2C EEPROM.                                 */
/*                                                                      */
/* File Name: TMS470R1xI2Cv1_0.gel                                      */
/* Author: J. Fahrenbruch                                               */
/*                                                                      */
/* Revision History:                                                    */
/*                                                                      */
/* 5/25/2005: Initaial Release                                         */
/***********************************************************/
#define MFBAHR0      0xFFFFFE00
#define MFBALR0      0xFFFFFE04
#define MFBAHR1      0xFFFFFE08
#define MFBALR1      0xFFFFFE0C
#define MFBAHR2      0xFFFFFE10
#define MFBALR2      0xFFFFFE14
#define MFBAHR3      0xFFFFFE18
#define MFBALR3      0xFFFFFE1C
#define PCR0               0xFFFFFD30
#define CLKCNTL     0xFFFFFFD0
#define GCR                0xFFFFFFDC
#define INPUT       0
#define OUTPUT       1
#define input           0
#define output       1
#define I             0
#define O           1
#define i            0
#define o           1
/* I2C1 Defines */
#define I2C1OAR      0xfff7d800
#define I2C1IMR      0xfff7d804
#define I2C1SR       0xfff7d808
#define I2C1CKL      0xfff7d80c
#define I2C1CKH      0xfff7d810
#define I2C1CNT      0xfff7d814
#define I2C1DRR      0xfff7d818
#define I2C1SAR      0xfff7d81c
#define I2C1DXR      0xfff7d820
#define I2C1MDR      0xfff7d824
#define I2C1IVR      0xfff7d828
#define I2C1EMR      0xfff7d82c
#define I2C1PSC      0xfff7d830
#define I2C1DIR      0xfff7d834
#define I2C1DOUT     0xfff7d838
#define I2C1DIN      0xfff7d83c
#define I2C1PFNC     0xfff7d848
#define TXMASK     0x0010  // TXRDY Mask I2CSR bit 4
#define RXMASK     0x0004  // RXRDY Mask I2CSR bit 3
#define BBMASK     0x1000  // Bus Busy Mask I2CSR bit 12
#define ACKMASK  0x0002  // Acknowledge mask I2CSR bit 1
StartUp()
{
    Initialize_TMS470R1x();
    GEL_TextOut("************************************************* \n");
    GEL_TextOut("*
                * \n");
    GEL_TextOut("*  Welcome to the TMS470R1x I2C test program               * \n");
    GEL_TextOut("*
                * \n");
    GEL_TextOut("*************************************************\n");
```

```
    GEL_TextOut("*
                * \n");
    GEL_TextOut("*
                * \n");
    GEL_TextOut("*
                * \n");
    GEL_TextOut("*
                * \n");
    GEL_TextOut("*
                * \n");
    GEL_TextOut("*
                * \n");
    GEL_TextOut("*
                * \n");
    GEL_TextOut("************************************************* \n");
    GEL_TextOut("*
                * \n");
    GEL_TextOut("*  Version 1.0  5/25/2005 JF Initial Release                          * \n");
    GEL_TextOut("*
                * \n");
    GEL_TextOut("************************************************* \n");

}
menuitem "TMS470"
/* This function should be used instead of the Debug->Restart menu
   selection when running ARM programs in mixed or 16 BIS (Thumb)
   mode.  It ensures the processor is in ARM mode at the entry
   point of TI C runtimes.                                        */
hotmenu ResetRestart()
{
    GEL_Reset();
    GEL_Restart();
}
/* This function is used instead of Debug-> Reset menu selection when
   running ARM programs. It ensures the processor is in ARM mode and
   then goes to the beginning of the program (main)                  */
hotmenu ResetGoMain()
{
    GEL_Reset();
    GEL_Go(main);
}
hotmenu ResetPeripherals()
{
    *(unsigned int *)PCR0    = 0;        /* Peripheral reset is asserted (default at reset)
      */
    *(unsigned int *)PCR0    = 0x03;  /* Peripheral reset is deasserted, ICLK = SYSCLK/2   */
}
hotmenu Set_CCS_Memory_                  /* Memory Map Used to Debug From SRAM */
{
  GEL_MapOn();
  GEL_MapReset();
  GEL_MapAdd(0x00000000,0,0x00004000,1,1);   /* SRAM */
  GEL_MapAdd(0x00200000,0,0x00000800,1,1);   /* Space allocated for HET */
  GEL_MapAdd(0xFFE88000,0,0x0000C000,1,1);   /* Flash Control Registers */
  GEL_MapAdd(0xFFF00000,0,0x00080000,1,1);   /* Peripheral Register Frame */
  GEL_MapAdd(0xFFF80000,0,0x00080000,1,1);   /* System Module Control Registers */
}
/* This function is used to setup the memory map for the TMS470 family
   of devices    */
hotmenu Initialize_TMS470R1x()
{
  Set_CCS_Memory_Map_RAM();
  *(unsigned int *)MFBAHR0 = 0x00000000;    /* int FLASH/boot FLASH/R0M                */
  *(unsigned int *)MFBALR0 = 0x00000100;              /* Enable the memory map
     */
  *(unsigned int *)MFBAHR1 = 0x00000000;    /* Starting address is 0x0000
```

```
*/
  *(unsigned int *)MFBALR1 = 0x00000000;    /* Block size is 0 Kb                          */
  *(unsigned int *)MFBAHR2 = 0x00000000;    /* Starting address is 0x0000
*/
  *(unsigned int *)MFBALR2 = 0x00000050;    /* Block size is 16 Kb           */
  *(unsigned int *)MFBAHR3 = 0x00000020;    /* HET Starting address is 0x002000000       */
  *(unsigned int *)MFBALR3 = 0x00000010;    /* Block size is 2 Kb                          */
  *(unsigned int *)PCR0    = 0x00000003;
}
/*************************************************/
/*** Output the ICLK or SYSCLK on the clkout pin ***/
/*************************************************/
menuitem"Clock Output for Test";
// Output the ICLK on clkout pin
hotmenu Output_ICLK_on_CLKOUTPUT_Pin()
{
    *(unsigned int *)CLKCNTL |= 0x30;
}
// Output the SYSCLK on clkout pin
hotmenu Output_SYSCLK_on_CLKOUTPUT_Pin()
{
    *(unsigned int *)CLKCNTL |= 0x70;
}
/*********************************************/
/*** Initialize the I2C channel 1 registers ***/
/*********************************************/
dialog I2C1_Initalization(ICLK "Enter ICLK Frequency (MHz)", I2C1FREQ "Bit Rate (1 = 100K / 4 =
400K)")

// I2C Base address = 0xfff7d800
{
    long I2C1CLKLv, I2C1PSCv;
    float dr, iclk, modclk1;

    if (I2C1FREQ == 1)
      {


        dr = .100;
      }

    else
      {
        dr = .400;
      }

    I2C1PSCv = 0;
    do
      {
        modclk1 = ICLK/(I2C1PSCv + 1);
        I2C1PSCv++;
      }
    while (modclk1 >= 13.3);  // Need to check for error if I2CPSCv > 255 (not Possible)
    I2C1PSCv--;
    if (I2C1PSCv == 0)
      I2C1CLKLv = ((modclk1 / dr) - 2 * 7) / 2;
    else if (I2C1PSCv == 1)
      I2C1CLKLv = ((modclk1 / dr) - 2 * 6) / 2;
    else if (I2C1PSCv > 1)
      I2C1CLKLv = ((modclk1 / dr) - 2 * 5) / 2;

    GEL_TextOut("The channel 1 I2C was set to operate with the following parameters: \n");
    GEL_TextOut("ICLK input = %f MHz\n",,,,, ICLK);
    GEL_TextOut("Module clock = %f MHz\n",,,,, modclk1);
    GEL_TextOut("I2C1CLKL = %d\n",,,,, I2C1CLKLv);
    GEL_TextOut("I2C1CLKH = %x\n",,,,, I2C1CLKLv);
```

```
    GEL_TextOut("Master clock (SCL) = %f K bits/sec\n",,,,, dr * 1000);

    *(unsigned int *)I2C1MDR = 0x4620;  // Initalize the Mode Register
    *(unsigned int *)I2C1OAR = 0x02;    // Set the Master address
    *(unsigned int *)GCR  = 0x09;       // f(sys)= 10Mhz * 4/2 = 20MHz
    *(unsigned int *)PCR0  = 0x00;        // Put PCR in reset
    *(unsigned int *)PCR0  = 0x02;        // Set the iclk = sysclk/2 = 10MHz
    *(unsigned int *)PCR0  |= 0x01;        // PCR out of reset
    *(unsigned int *)I2C1PSC = I2C1PSCv; // 0x0000;  Set the prescale moduleclk = iclk/1
    *(unsigned int *)I2C1CKL = I2C1CLKLv; //0x1A;    // Set the master clock (SCL) low time
    *(unsigned int *)I2C1CKH = I2C1CLKLv; //0x1A;    // Set the master clock (SCL) high time
                                        // SCLK = 100KHz 50% duty cycle
    *(unsigned int *)I2C1DIR = 0x0F;    // Set SCL and SDL direction as I2C output
}
/*****************************************/
/*** Write data to I2C Channel 1           ***/
/*****************************************/
dialog I2C1_Write(I2C1add "Address to Write",I2C1dta "Data to Write")
{
  int statin, n, a, d, DRRv;
  *(unsigned int *)I2C1SAR = 0x50;     // Load Slave Address
  *(unsigned int *)I2C1CNT = 0x02;     // Number of bytes to transmit
  *(unsigned int *)I2C1DXR = I2C1add;  // Load data address
  do {} while ((*(unsigned int *)I2C1SR & BBMASK) == BBMASK); // Wait for a free bus
  *(unsigned int *)I2C1MDR = 0x6E20;   // Start Transmit of Control Byte
  do {}  while ((*(unsigned int *)I2C1SR & ACKMASK) == ACKMASK); // Wait for acknowledgement
  *(unsigned int *)I2C1DXR = I2C1dta;     // Load data
  do {} while ((*(unsigned int *)I2C1SR & TXMASK) != TXMASK); // Wait for data transmit to
complete
  do {}  while ((*(unsigned int *)I2C1SR & ACKMASK) == ACKMASK); // Wait for acknowledgement

  *(unsigned int *)I2C1MDR = 0x4E20;   // Transmit STOP Bit
    GEL_TextOut("Write Completed to address = %x\n",,,,,I2C1add);
//Verify Write
  GEL_TextOut("Verifing write\n",,,,,);
  // Start Transmit of Control Byte
  *(unsigned int *)I2C1SAR = 0x50;     // Load Control Byte
  *(unsigned int *)I2C1CNT = 0x01;     // Number of bytes to transmit
  *(unsigned int *)I2C1MDR = 0x6620;   // Transmit control byte
  do {}  while ((*(unsigned int *)I2C1SR & ACKMASK) == ACKMASK); // Wait for acknowledgement
  *(unsigned int *)I2C1DXR = I2C1add;     // Load data address
  do {}  while ((*(unsigned int *)I2C1SR & ACKMASK) == ACKMASK); // Wait for acknowledgement

  *(unsigned int *)I2C1CNT = 0x01;
  *(unsigned int *)I2C1MDR = 0x6C20;   // Receive data byte
  do {} while ((*(unsigned int *)I2C1SR & RXMASK) == RXMASK); // Wait to receive data
  DRRv = 0;
  DRRv = *(unsigned int *)I2C1DRR;
  *(unsigned int *)I2C1MDR = 0x4E20;   // Transmit STOP Bit
  *(unsigned int *)I2C1SR &= ~BBMASK;
  if (DRRv == I2C1dta)
    {
      GEL_TextOut("DATA VERIFIED AT Address = %x\n",,,,,I2C1add);
      GEL_TextOut("%x = %x\n\n",,,,,I2C1dta, DRRv);
    }
   else
    {
      GEL_TextOut("*** WRITE FAILED *** DID NOT VERIFY *** AT Address = %x\n",,,,,I2C1add);
      GEL_TextOut("%x <> %x\n\n",,,,,I2C1dta, DRRv);
    }
}

/*****************************************/
/*** Read data from I2C channel 1          ***/
/*****************************************/
dialog I2C1_Read(I2C1add "Address")
```

```
{
    int DRR, mdrin, a, b;
  *(unsigned int *)I2C1SAR = 0x50;     // Load Control Byte
  *(unsigned int *)I2C1CNT = 0x01;     // Number of bytes to transmit
  *(unsigned int *)I2C1MDR = 0x6620;   // Transmit control byte
  do {}  while ((*(unsigned int *)I2C1SR & ACKMASK) == ACKMASK); // Wait for acknowledgement
  *(unsigned int *)I2C1DXR = I2C1add;     // Load data address
  do {}  while ((*(unsigned int *)I2C1SR & ACKMASK) == ACKMASK); // Wait for acknowledgement

  *(unsigned int *)I2C1CNT = 0x01;
  *(unsigned int *)I2C1MDR = 0x6C20;   // Receive data byte
  do {} while ((*(unsigned int *)I2C1SR & RXMASK) == RXMASK); // Wait to receive data
  DRR = 0;
  DRR = *(unsigned int *)I2C1DRR;
  *(unsigned int *)I2C1MDR = 0x4E20;   // Transmit STOP Bit
  *(unsigned int *)I2C1SR &= ~BBMASK;
  GEL_TextOut("Data read at address = %x\n",,,,,I2C1add);
  GEL_TextOut("Data = %x\n\n",,,,,DRR);
}
```

5 Reference Material

1. *TMS470R1x Inter-Integrated Circuit (I2C) Reference Guide* (SPNU223)
2. *TMS470R1X Zero-Pin Phase-Locked Loop (ZPLL) Clock Module Reference Guide* (SPNU212)
3. *TMS470R1x System Module Reference Guide* (SPNU189)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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