Application Report
SPNA098A–July 2006–Revised April 2007

TMS470 I²C Protocol

Keith Engler and Harman Grewal — *TMS470 Applications*

ABSTRACT

The TMS470 family of ARM7 microcontrollers has up to five I²C ports available, depending on which device is selected. The TMS470 I²C engine is very flexible and powerful. This document describes how to read and write I²C data and how to select the I²C clock rate.

Contents

List of Figures

List of Tables

1 Introduction

This document contains specific examples for initializing, reading, and writing I²C data using the TMS470. This document is meant to supplement, but not replace, the *TMS470 I²C Reference Guide*.[1] See the Philips I²C specification [4] for more detailed understanding of the I²C protocol. This document also assumes the reader has a basic understanding of C-programming concepts.

This document has C-code examples for writing data to the DAC8574 (digital-to-analog converter) and reading data from the ADC1110 (analog-to-digital converter) via the I²C bus. Both devices are from Texas Instruments.

This document lists two C-code functions for reading and writing I²C data. Then each function is described in terms of I²C setup, I²C clocking, I²C read, and I²C write.

Each I²C transmission in this document take places between a valid I²C Start and a valid I²C Stop.[4] The TMS470 is capable of performing multiple reads and multiple writes following an I²C Start, but an example for this case is not discussed in this document.

2 TMS470 I²C C-Code Examples

This section has C-code examples for writing digital data to the DAC8574 and reading digital data from the ADC1110. These functions can be used directly with the IAR Embedded Workbench™.

2.1 Read I²C Code Example Using the ADC1110 Analog-to-Digital Converter

This section has C-code examples for reading I²C data from the ADC1110 and the supporting variables to the function.

```
   unsigned char adcp1;       // ADC variable for Most Significant Byte
   unsigned char adcp2;       // ADC variable for Least Significant Byte
   unsigned char config;      // ADC variable for Configuration Register

void ADS1110_ReadVoltage(void)
{
  //I2C 1 Set up
  I2C1PSC = 2;               // Module clock frequency
                             // I2CCLK = ICLK / (PSC + 1)
                             // I2CCLK = 10 MHz
  I2C1CKL = 45;              // Low clock period
  I2C1CKH = 45;              // High clock period
  I2C1OAR = 0x56;            // Set TMS470's Own address to 0x56
  I2C1IMR = 0x0;             // Interrupts disabled
  I2C1CNT = 3;               // Number of byte transactions between I2C Start and Stop
  I2C1SAR = 0x48;            // Set address of ADC to 0x48
  I2C1PFNC = 0;              // Pins function as SDA and SCL pins
  I2C1DIR = SDAFUNC+SCLFUNC; // Set I2C Pin direction
  I2C1MDR |= MST;            // Make TMS470 addr 0x56 the IIC the Master
  I2C1MDR |= NIRS;           // Clear Reset
  I2C1MDR &= ~TRX;           // Clear transmit to Read
  I2C1MDR |= STT + STP;      // Start, Stop and Transmit
  while(!(I2C1SR & 0x0008)); // Verify I2CDRR has been read
  adcp1 = (I2C1DRR);         // Capture Most Significant Byte First
  while(!(I2C1SR & 0x0008)); // Verify I2CDRR has been read
  adcp2 = (I2C1DRR);         // Capture Least Significant Byte Second
  while(!(I2C1SR & 0x0008)); // Verify I2CDRR has been read
  config = (I2C1DRR);        // Capture Configuration Register Value third
}
```

Figure 1. I²C Read (RX) C-Code

2.2 Write I 2C Code Example Using the DAC8574 Digital-to-Analog Converter

This section has C-code examples for writing I2C data to the DAC8574 and the supporting variables to the function.

```
const int sin_table[] = {
32500, 37584, 42543, 47255, 51603, 55481, 58793, 61458, 63409, 64600,
65000, 64600, 63409, 61458, 58793, 55481, 51603, 47255, 42543, 37584,
32500, 27416, 22457, 17745, 13397,  9519,  6207,  3542,  1591,   400,
    0,   400,  1591,  3542,  6207,  9519, 13397, 17745, 22457, 27416,
0xffff
};
const int* ip;                      // Pointer to sin_table

void DAC8574_SetVoltage(void)
{
  I2C1PSC = 2;                      // Module clock frequency
                                    // I2CCLK = ICLK / (PSC + 1)
                                    // I2CCLK = 10 MHz
  I2C1CKL = 45;                     // Low clock period
  I2C1CKH = 45;                     // High clock period
  I2C1OAR = 0x56;                   // Set address 0x56
  I2C1IMR = 0x0;                    // Interrupts disabled
  I2C1CNT = 3;                      // Set count = 3 to write 3 samples
  I2C1SAR = 0x4C;                   // Set address of DAC to 0x4C
  I2C1PFNC = 0;                     // Pins function as SDA and SCL pins
  I2C1DIR = SDAFUNC+SCLFUNC;        // Set I2C Pin direction
  I2C1MDR |= MST;                   // Master
  I2C1MDR |= NIRS;                  // Clear Reset
  I2C1MDR |= STT + STP + TRX;       // Start, Stop and Transmit
  if (*ip == 0xffff)                // If the Last value of sin_table is encountered then
{
  ip = sin_table; }                 // re-point ip back to the sin-table start
  }
  dacp = *ip;                       // load the sin_table value into the DAC
  I2C1DXR = 0x10;                   // Byte1 is the DAC Control Byte (0x10)
  while(!(I2C1SR & 0x0010));        // Verify I2CDXR is empty (TXRDY=1) before continuing
  I2C1DXR = (dacp & 0xFF00) >>8;    // Send the Most Significant Byte first
  while(!(I2C1SR & 0x0010));        // Verify I2CDXR is empty (TXRDY=1) before continuing
  I2C1DXR = (dacp & 0xFF);          // Send the Least Significant Byte second
  while(!(I2C1SR & 0x0010));        // Verify I2CDXR is empty (TXRDY=1) before continuing
  ip++;                             // Increment to the next value in the sin_table
}
```

Figure 2. I2C Write (TX) C-Code

3 TMS470 I²C Setup Description

The TMS470 I²C setup consists of setting the I²C transfer rates to either 100 kbps or 400 kbps, setting the mode of the I²C pin to either GPIO or I²C, and setting the data direction as either a transmit or receive. Figure 3 shows the difference between setup for an I²C read (RX) and an I²C write (TX).

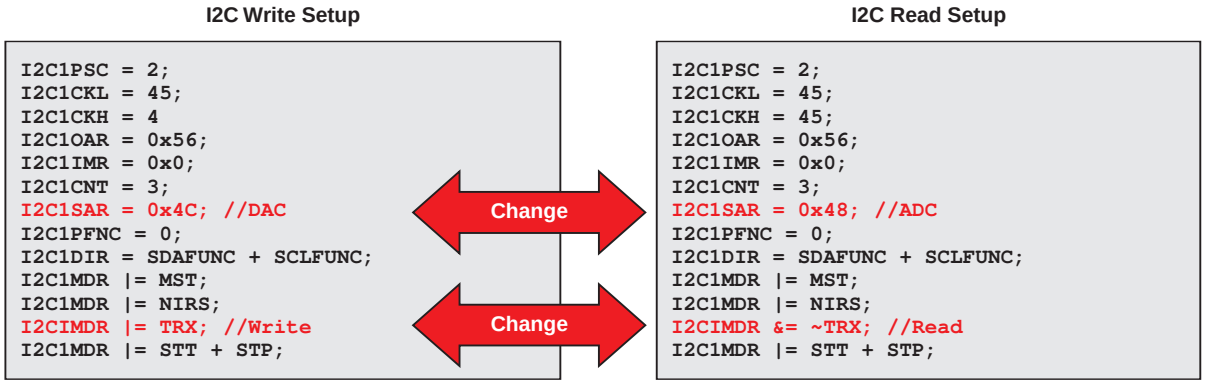

Figure 3. I²C Read (RX) and I²C Write (TX) Setup Comparison

3.1 Hardware Description

The examples in this application report were tested using the following hardware:

- TMS470R1B1M KickStart™ Development Kit from IAR [5]
- HPA-MCU Interface Board [6]
- ADS110 EVM [7]
- DAC8574 EVM [8]

The block diagram for the test setup is shown in Figure 4.

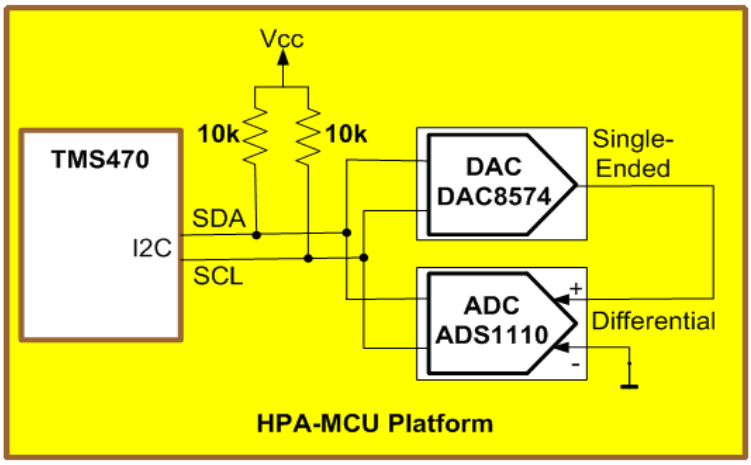

Figure 4. TMS470-ADS1110-DAC8574 Test Setup

This application is tested using the IAR Embedded Workbench, and the application code is provided in this file: http://www-s.ti.com/sc/psheets/spna098/spna098.zip

4 TMS470 I²C-Specific Read and Write Protocol

The I²C read and I²C write is centrally focused on the I²C Status Register (I2CSR). During a write, the status register is polled to verify that the data has been sent from the transmit register (I2CDXR). During a read, the status register is polled to verify the data has been captured into the receive register (I2CDRR).

```
I2C1DXR = 0x10;                 // Send DAC Control Byte 1st
while(!(I2C1SR & 0x0010));      // Verify I2CDXR is empty (TXRDY=1)
I2C1DXR = (dacp & 0xFF00) >>8;  // Send the Least Significant Byte 2nd
while(!(I2C1SR & 0x0010));      // Verify I2CDXR is empty (TXRDY=1)
I2C1DXR = (dacp & 0xFF);        // Send the Most Significant Byte 3rd
while(!(I2C1SR & 0x0010));      // Verify I2CDXR is empty (TXRDY=1)c
```

Figure 5. I²C Write Example

```
while(!(I2C1SR & 0x0008));  // Verify I2CDRR has been read
adcp1 = (I2C1DRR);          // Capture Most Significant Byte 1st
while(!(I2C1SR & 0x0008));  // Verify I2CDRR has been read
adcp2 = (I2C1DRR);          // Capture Least Significant Byte 2nd
while(!(I2C1SR & 0x0008));  // Verify I2CDRR has been read
config = (I2C1DRR);         // Capture Configuration Register 3rd
```

Figure 6. I²C Read Example

5 TMS470 I²C Clock-Rate Selection

The TMS470 supports I²C transfer rates of 100 kbps and 400 kbps. The I²C clock rate is derived from ICLK, which is dependant on the crystal oscillator (FOSC). Figure 7 shows how the I²C clocks are derived.

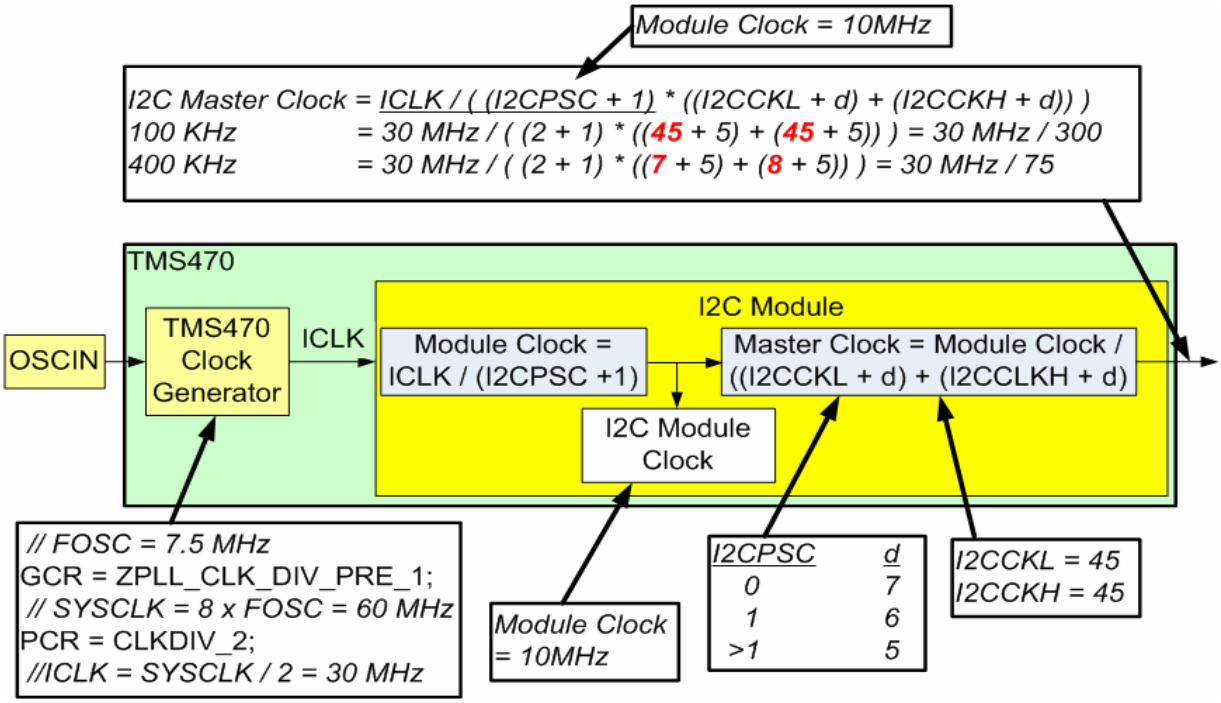

Figure 7. I²C Clocking

The GCR is the Global Control Register. The example in Figure 7 sets the GCR to zero, which creates a system clock that is eight times the FOSC. There are eight possible prescale values, as shown in Table 1. See reference [2] for complete information.

The PCR is the Peripheral Clock Register. The example in Figure 7 sets the PCR to divide the system clock by two. There are 16 possible values, as shown in Table 2. The clock divider in the PCR determines the ICLK value that is the input clock to the I²C module. See reference [3] for complete information.

The I²C master clock is the bit rate of the I²C clock and data lines. I2CCKL and I2CCLKH are low and high times of the I²C clock. The example in Figure 1 uses a 50% duty cycle for the high and low times to produce an exactly 100-kHz I²C clock. The 400-kHz master clock requires a non-uniform clock period. See reference [1] for complete information.

Table 1. Global Control Register (GCR) Bit Definitions

NAME	VALUE	BITS	DESCRIPTION
ZPLL_CLK_DIV_PRE_1	0x0000	2:0	ZPLL clock divider prescale
ZPLL_CLK_DIV_PRE_2	0x0001	2:0	ZPLL clock divider prescale
ZPLL_CLK_DIV_PRE_3	0x0002	2:0	ZPLL clock divider prescale
ZPLL_CLK_DIV_PRE_4	0x0003	2:0	ZPLL clock divider prescale
ZPLL_CLK_DIV_PRE_5	0x0004	2:0	ZPLL clock divider prescale
ZPLL_CLK_DIV_PRE_6	0x0005	2:0	ZPLL clock divider prescale
ZPLL_CLK_DIV_PRE_7	0x0006	2:0	ZPLL clock divider prescale
ZPLL_CLK_DIV_PRE_8	0x0007	2:0	ZPLL clock divider prescale (default)
ZPLL_MULT4	0x0008	3	Multiply by 4 or 8
FLCONFIG	0x0010	4	Flash configuration enable
RESERVED		13:5	Reserved
RTI_CTRL	0x4000	14	RTI control
RST_OSC_FAIL_EN	0x8000	15	Reset on oscillator failure enable

Table 2. Peripheral Control Register (PCR) Bit Definitions

NAME	VALUE	BITS	DESCRIPTION
PENABLE	0x0	0	Peripheral enable
CLKDIV_1	0x00 << 1	4:1	ICLK = SYSCLK/1
CLKDIV_2	0x01 << 1	4:1	ICLK = SYSCLK/2
CLKDIV_3	0x02 << 1	4:1	ICLK = SYSCLK/3
CLKDIV_4	0x03 << 1	4:1	ICLK = SYSCLK/4
CLKDIV_5	0x04 << 1	4:1	ICLK = SYSCLK/5
CLKDIV_6	0x05 << 1	4:1	ICLK = SYSCLK/6
CLKDIV_7	0x06 << 1	4:1	ICLK = SYSCLK/7
CLKDIV_8	0x07 << 1	4:1	ICLK = SYSCLK/8
CLKDIV_9	0x08 << 1	4:1	ICLK = SYSCLK/9
CLKDIV_10	0x09 << 1	4:1	ICLK = SYSCLK/10
CLKDIV_11	0x0A << 1	4:1	ICLK = SYSCLK/11
CLKDIV_12	0x0B << 1	4:1	ICLK = SYSCLK/12
CLKDIV_13	0x0C << 1	4:1	ICLK = SYSCLK/13
CLKDIV_14	0x0D << 1	4:1	ICLK = SYSCLK/14
CLKDIV_15	0x0E << 1	4:1	ICLK = SYSCLK/15
CLKDIV_16	0x0F << 1	4:1	ICLK = SYSCLK/16

5.1 *TMS470 I²C Clock-Rate Selection Code Example*

The TMS470 has five registers that must be set to achieve the I²C transfer rates of 100 kbps and 400 kbps.

- GCR: Global Control Register
- PCR: Peripheral Clock Register
- I2C1PSC: I²C Pre-Scale Register is used to divide down the system clocks
- I2C1CKL: I²C Clock Divider Low Register sets the low time of the I²C clock
- I2C1CKH: I²C Clock Divider High Register sets the high time of the I²C clock

I²C Example for 100 kHz (With System Clock = 60 MHz)

```
PCR = CLKDIV_2;            // ICLK = SYSCLK/2 = 30 MHz
GCR = ZPLL_CLK_DIV_PRE_1;  // SYSCLK = 8 x fOSC = 8 X 7.5 MHz = 60 MHz
I2C1PSC = 2;               // Module Clock Freq = ICLK / (PSC + 1) = 10 MHz
I2C1CKL = 45;              // Low Clock Period = (I2CCKL + d) / Module Clock Freq =
                           // 5.0 usec
I2C1CKH = 45;              // High clock period = (I2CCKH + d) / Module Clock Freq =
                           // 5.0 usec
```

NOTE: d = 5 for any I2CPSC > 1

I²C Example for 400 kHz (With System Clock = 60 MHz)

```
PCR = CLKDIV_2;            // ICLK = SYSCLK/2 = 30 MHz
GCR = ZPLL_CLK_DIV_PRE_1;  // SYSCLK = 8 x fOSC = 8 X 7.5 MHz = 60 MHz
I2C1PSC = 2;               // Module Clock Freq = ICLK / (PSC + 1) = 10 MHz
I2C1CKL = 7;               // Low Clock Period = (I2CCKL + d) / Module Clock Freq =
                           // 1.2 usec
I2C1CKH = 8;               // High clock period = (I2CCKH + d) / Module Clock Freq =
                           // 1.3 usec
```

NOTE: d = 5 for any I2CPSC > 1

If I2CPSC is equal to 0, the value for d is 7. If I2CPSC is equal to 1, the value for d is 8. The value for d is equal to 5 for all values of I2CPSC greater than 1.

6 TMS470 I2C Repeated Start Sequence

The TMS470 supports I2C Repeated Start sequence. The key register for issuing the repeated Start is the I2C1MDR.

To initialize this register for regular I2C mode, set the STP bit in the I2C1MDR register, as shown here:

```
I2C1MDR = 0x00;              // Master
I2C1MDR |= MST;              // Master
I2C1MDR |= NIRS;             // Clear Reset
I2C1MDR |= STT + STP+ TRX;   // Start, Stop and Transmit
```

To initialize this register for Repeated Start I2C mode, do not set the STP bit in the I2C1MDR register, as shown here:

```
I2C1MDR = 0x00;              // Master
I2C1MDR |= MST;              // Master
I2C1MDR |= NIRS;             // Clear Reset
I2C1MDR |= STT + TRX;        // Start, No Stop and Transmit
```

To continue sending data, keep sending the start command:

```
I2C1MDR |= STT;              // Repeated Start
```

The complete program is shown in Figure 8:

```
#include "SinTable_01.h"
  unsigned int dacp;             // DAC variable for mapping to sin_table
  const int* ip;                 // Pointer to sin_table

void DAC8574_Init(void)
{
  //I2C 3 is set to Force Pins A0 and A1 to zero for setting DAC8574 I2C Address
  I2C3PFNC = 0x01;               // I2C3 Pins function as GPIO pins
  I2C3DIR = 0x01;                // SCL pin is an Output SDA is an Input
  I2C3DOUT = 0x00;               // SCL Pin is Driven Low

  //I2C 1 Set up
  I2C1PSC = 2;                   // Module clock frequency
                                 // I2CCLK = ICLK / (PSC + 1)
                                 // I2CCLK = 10 MHz
  I2C1CKL = 45;                  // Low clock period
  I2C1CKH = 45;                  // High clock period
  I2C1OAR = 0x56;                // Set address 0x56
  I2C1IMR = 0x0;                 // Interrupts disabled
  I2C1CNT = 3;                   // Set count = 3 to write 3 samples
  I2C1SAR = 0x4C;                // Set address of DAC to 0x4C
  I2C1PFNC = 0;                  // Pins function as SDA and SCL pins
  I2C1DIR = SDAFUNC+SCLFUNC;     // Set I2C Functions
  I2C1MDR = 0x00;                // Master
  I2C1MDR |= MST;                // Master
  I2C1MDR |= NIRS;               // Clear Reset
  I2C1MDR |= STT + TRX;          // Start, Repeated Start and Transmit

  if (*ip == 0xffff)             // If the Last value of sin_table is encountered then
  {
    ip = sin_table;              // repoint ip back to the sin-table start
  }
  dacp = *ip;                    // load the sin_table value into the DAC
  I2C1DXR = 0x10;                // Byte1 is the DAC Control Byte (0x10)
  while(!(I2C1SR & 0x0010));     // Verify I2CDXR is empty (TXRDY=1) before continuing
  I2C1DXR = (dacp & 0xFF00) >>8; // Send the Most Significant Byte first
  while(!(I2C1SR & 0x0010));     // Verify I2CDXR is empty (TXRDY=1) before continuing
  I2C1DXR = (dacp & 0xFF);       // Send the Least Significant Byte second
  while(!(I2C1SR & 0x0010));     // Verify I2CDXR is empty (TXRDY=1) before continuing
  ip++;                          // Increment to the next value in the sin_table
}
```

```
void DAC8574_Send(void)
{

  if (*ip == 0xffff)             // If the Last value of sin_table is encountered then
  {
    ip = sin_table;              // repoint ip back to the sin-table start
  }
  I2C1MDR |= STT;                // Repeated Start
  dacp = *ip;                    // load the sin_table value into the DAC
  I2C1DXR = 0x10;                // Byte1 is the DAC Control Byte (0x10)
  while(!(I2C1SR & 0x0010));     // Verify I2CDXR is empty (TXRDY=1) before continuing
  I2C1DXR = (dacp & 0xFF00) >>8; // Send the Most Significant Byte first
  while(!(I2C1SR & 0x0010));     // Verify I2CDXR is empty (TXRDY=1) before continuing
  I2C1DXR = (dacp & 0xFF);       // Send the Least Significant Byte second
  while(!(I2C1SR & 0x0010));     // Verify I2CDXR is empty (TXRDY=1) before continuing
  ip++;                          // Increment to the next value in the sin_table
}

void main(void)
{
  __disable_interrupt();            // Disable interrupts.

  PCR = CLKDIV_2;                   // ICLK = SYSCLK/2
  PCR |= PENABLE;                   // Enable peripherals
  GCR = ZPLL_CLK_DIV_PRE_1;         // SYSCLK = 8 x fOSC

  ip = sin_table;                   // Direct the IP Pointer to start of sin_table

  DAC8574_Init();                   // It is important to set and read the voltages

  while (1)                         // Transfer Data infinitely
  {
    DAC8574_Send();                 // Convert the sin-table to an Analog Voltage
    for(int ii=0; ii<100000; ii++); // Short Delay to let DAC Voltages Stabilize
  };
}
```

Figure 8. Repeated Start Code Example

7 Conclusion

This application report gives specific examples for initializing, reading, and writing I^2C data using the TMS470. This document is meant to supplement, not replace, the *TMS470 I^2C Reference Guide*.[1] The C-code examples in this document can be applied to any TMS470 device having the I^2C peripheral.

It is important to note a potential problem within the code examples. If a hardware failure occurs in any instance where there is a While loop writing to the DAC or reading from the ADC, the program could be stuck in an infinite loop waiting for the I^2C transaction to complete. This issue must be addressed by the user during implementation.

8 References

1. *TMS470R1x Inter-Integrated Circuit (I^2C) Reference Guide*, Texas Instruments (SPNU223)
2. *TMS470R1x Zero-Pin Phase-Locked Loop (ZPLL) Clock Module Reference Guide*, Texas Instruments (SPNU212)
3. *TMS470R1x System Module Reference Guide*, Texas Instruments (SPNU189)
4. *The I^2C-Bus Specification* Version 2.1, January 2000, Phillips Semiconductors, document 9398 393 40011
5. *TMS470R1B1M KickStart™ Development Kit* from IAR, Texas Instruments, http://focus.ti.com/docs/toolsw/folders/print/spnc010.html
6. HPA-MCU Interface Board, Texas Instruments (SLAU106)
7. *ADS110 EVM User's Guide*, Texas Instruments (SBAU089)

8. *DAC8574 EVM User's Guide*, Texas Instruments (SLAU109)

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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
Low Power Wireless	www.ti.com/lpw	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