TMS320 DSP
DESIGNER'S NOTEBOOK

C Routines for Setting Up the AIC on the TMS320C5x EVM

APPLICATION BRIEF: SPRA251

Leor Brenman
Digital Signal Processing Products
Semiconductor Group

Texas Instruments
February 1995

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1997, Texas Instruments Incorporated

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Abstract 7
Design Problem 8
Solution 8

C Routines for Setting Up the AIC on the TMS320C5x EVM

Abstract

This document discusses how to use the C programming language to control the AIC on the TMS320C5x EVM. Different ways of doing this are shown, with code listings and examples for each.

Design Problem

How do I control the AIC on the TMS320C5x EVM in C?

Solution

Programming the ’C5x to communicate with the AIC on the EVM involves (1) setting up the ’C5x serial port and (2) resetting and (3) configuring the AIC. Since the ’C5x serial-port registers (SPC, DXR, and DRR) are memory mapped, setting up the ’C5x serial port and reading and writing to the serial-port-receive and transmit registers is easily accomplished in C. For example, the ’C5x serial-port-control register, memory mapped at address 0x0022, could be declared in C as follows:

```
volatile unsigned int *SPC = (volatile unsigned int
*) 0x0022;
```

and accessed as follows:

```
SPCVALUE = *SPC;
*SPC = 0x00c8;
```

Resetting the AIC is accomplished by reading from, and writing to a memory-mapped I/O port connected to a target control register, which is also easily accomplished in C. Finally, configuring the AIC is achieved by writing values to the AIC through the ’C5x serial port. The TMS320 BBS self-extracting file, EVMAIC5X.EXE, contains code necessary to create a library, EVMAIC5X.LIB, that contains the following two functions:

```
void initAic(AIC_CONFIGURATION *aicParams);
void getDefaultAicConfig(AIC_CONFIGURATION
                                          *aicParams);
```

The functions are defined in the file EVMAIC5X.C and are prototyped in the header file EVMAIC5X.H. This file should be included (i.e., #include) in user programs that link to the EVMAIC5X.LIB library. Also included in the file, EVMAIC5X.EXE, is demo code that illustrates how to build an application that uses the library functions.

The code is written entirely in C and provides a starting point for users writing C code to interface ’C5x processors to TLC3204x AICs on any hardware platform.

The code demonstrates the following:

1) Communicating to the AIC in primary and secondary modes in C.

2) Using bit fields in C.

3) Accessing memory-mapped DSP registers and I/O peripherals in C.

4) Using the C language extension asm statement.

Usage

The user should call the function `initAic()` after all other processor and variable initialization is complete. The function globally enables interrupts and enables serial-port-receive interrupts only. After the function is called, serial-port-receive interrupts must be ready to be serviced. That is, a serial-port-receive interrupt service routine (ISR) vector must be installed and a serial-port-receive ISR must be defined. The default AIC configuration for this library is for synchronous operation. Other AIC default parameters can be found in the function `getDefaultAicConfig()`. To use the default configuration, call the function `initAic()` with `NULL` as a parameter, as follows:

```
initAic(NULL);
```

If the AIC is to be operated in asynchronous mode, the user's code must enable serial-port-transmit interrupts after calling `initAic()` and be prepared to service serial-port-transmit interrupts as well as receive interrupts. In addition, to modify any other default AIC configuration setup by the `initAic()` function, simply pass the AIC configuration parameters to the function `initAic()` in a data structure of type `AIC_CONFIGURATION`, defined as follows:

```
typedef struct
{
      AIC_COMMAND_0 command0;
      AIC_COMMAND_1 command1;
      AIC_COMMAND_2 command2;
      AIC_COMMAND_3 command3;
}  AIC_CONFIGURATION;
```

This structure definition as well as those for `AIC_COMMAND_0` through `AIC_COMMAND_3` are defined in the header file EVMAIC5X.H. As an example, the bit-field structure definition for `AIC_COMMAND_3` is shown below:

```
typedef struct
{
      unsigned int command :2;
      unsigned int highpass :1;
      unsigned int loopback :1;
      unsigned int aux :1;
      unsigned int sync :1;
      unsigned int gain :2;
      unsigned int d_8 :1;
      unsigned int sinx :1;
      unsigned int d10out :1;
      unsigned int d11out :1;
```

```
      unsigned int d_cdef :4;
} AIC_COMMAND_3;
```

The function `getDefaultAicConfig()` returns the default configuration. If only a few configuration parameters need to be altered, the user can call this function and then change the parameters as needed. The following example illustrates how to do this:

1) Create a variable of type `AIC_CONFIGURATION:`

```
AIC_CONFIGURATION aicParams;
```

2) `getDefaultAicConfig(&aicParams)`

3) Modify the parameters as necessary:

```
aicParams.command3.loopback = 1;
```

4) Call the function `initAic()` with the address of the `aic` parameter variable:

```
initAic(&aicParams);
```

One additional data structure, `AIC_PRIMARY,` has been defined in EVMAIC5X.H It is useful for handling the primary communications data (i.e., the speech or audio) from the AIC in your interrupt service routine. By using this data structure efficiently, automatic data shifting is accomplished as required by the transmit and receive data format of the AIC. It is defined as follows:

```
typedef union
{
      unsigned int _intval;
      struct
      {
            unsigned int command   :2
                     ; /* Must be initialized to 0 */
            signed int data        :14
      } _bitval;
} AIC_PRIMARY;
```

Assume a global variable called `aicPrimary` of type `AIC_PRIMARY` has been declared; data is read from the ’C5x serial-port receive register, DXR, as follows:

```
#define DXR ((volatile unsigned int *) 0x0021)
aicPrimary._bitval.command = 0
           ; /* Initialize lower 2 bits to 0 */
aicPrimary._intval = *DXR
           ; /* Read all 16 bits */
dataReceived = aicPrimary._bitval.data
           ; /* Useful data is the upper 14 bits */
```

Similarly, the following technique can be used to output data:

```
#define DRR   ((volatile unsigned int *) 0x0020)
aicPrimary._bitval.data = xmitData
                  ; /* Write data to upper 14 bits */
*DXR = aicPrimary._intval; /* Transmit all 16 bits */
```

In the previous two code segments, notice the alternative method used to access DXR and DRR, the serial-port-transmit and receive registers in C.

The following code fragment illustrates the necessary components of a complete application to use this library:

```
#include “evmaic5x.h”


AIC_PRIMARY aicPrimary;

void main(void)
{
      .
      .
      aicPrimary._bitval.command = 0;
      aicPrimary._bitval.data = 0;
      .
      .
      initAic(NULL); /* Initialize AIC just before */
                  /* entering the processing loop */
      for(;;) {}
}

void c_int5(void) /* Serial port receive ISR
                              (must be installed) */
{
      #define DRR ((volatile unsigned int *) 0x0020)
      #define DXR ((volatile unsigned int *) 0x0021)

      signed int recData;
      signed int xmtData;
      .
      .
      aicPrimary._intval = *DXR;
      recData = aicPrimary._bitval.data;

      /* Do processing from recData to xmtData */

      aicPrimary._bitval.data = xmtData;
      *DXR = aicPrimary._intval;
}
```