Application Report
SPRA559A

A Multichannel Serial Port Driver Using DMA on the TMS320C6000 DSP

Zhaohong Zhang
Xiangdong Fu

C6000 Applications

Abstract

This application report presents an implementation of the Texas Instruments (TI™) TMS320C6000 digital signal processor (DSP) serial port driver. The implementation extracts data from the TDM data stream, saves the data to the individual buffer of each channel, and constructs the TDM data stream from individual data buffers.

Contents

Figures

Introduction

In many applications the multichannel PCM data enters and leaves the processor via the on-chip serial port in the form of TDM data stream. The driver function must perform two basic tasks:

- Extract data from the TDM data stream and save it to the individual data buffers for each channel
- Construct the TDM data stream using the data from individual data buffers

The driver also must inform the processor of the amount of data in each buffer waiting for processing or output. This application report illustrates the approach of applying a combination of C6X DMA (direct memory access) functions, such as index addressing, auto address reloading, and interrupt, to handle a serial port I/O in a multichannel/algorithm system.

Approach

A data buffer must be allocated to each channel to save the input or output data. The buffer sizes for all channels should be equal. To support algorithms with different frame sizes, the buffer size must be a common denominator of all frame sizes supported in the system. Considering the processing delay, we also recommend that the buffer size be at least three times the maximum frame size supported. A data counter is also created for each channel to save the amount of data waiting for processing or output. We also define a data block as the greatest common factor of all the frame sizes supported. One data frame contains one or more blocks.

We organize all channel buffers into a single super buffer, as shown in Figure 1. The DMA element index parameter is set up in such a way that the save address pointer is incremented by the channel buffer size. After all of the channel buffers receive a block of data, a DMA interrupt is generated due to the completion of the block transfer.

Two tasks are performed in the interrupt service routine. The first task updates the value in the DMA address reloading register for the start address of the next transfer. The DMA transfers the data using the register value loaded previously for the start address of the current transfer. The start address is directed to the beginning of the buffer when the entire buffer is filled with data.

The second task updates the values of the data counters for all channels. The processing module can determine if a complete frame of data has been received by comparing the data counter with the channel frame size. Because the DMA address reloading registers only must be updated before the completion of the current block transfer, large latency would be tolerated for the response to this interrupt.

Figure 2 shows the block diagram of the DMA block interrupt service routine for receiving data. Each buffer consists of N blocks. The super buffer is named in_buffer. The variable dest is the value to be loaded into the DMA address reloading register. At initialization, the DMA destination address is set to in_buffer[0], index is initialized at one, and the address reloading register is loaded with the address of in_buffer[block_size].

Figure 1. Arrangement of Super Buffer for M Channels

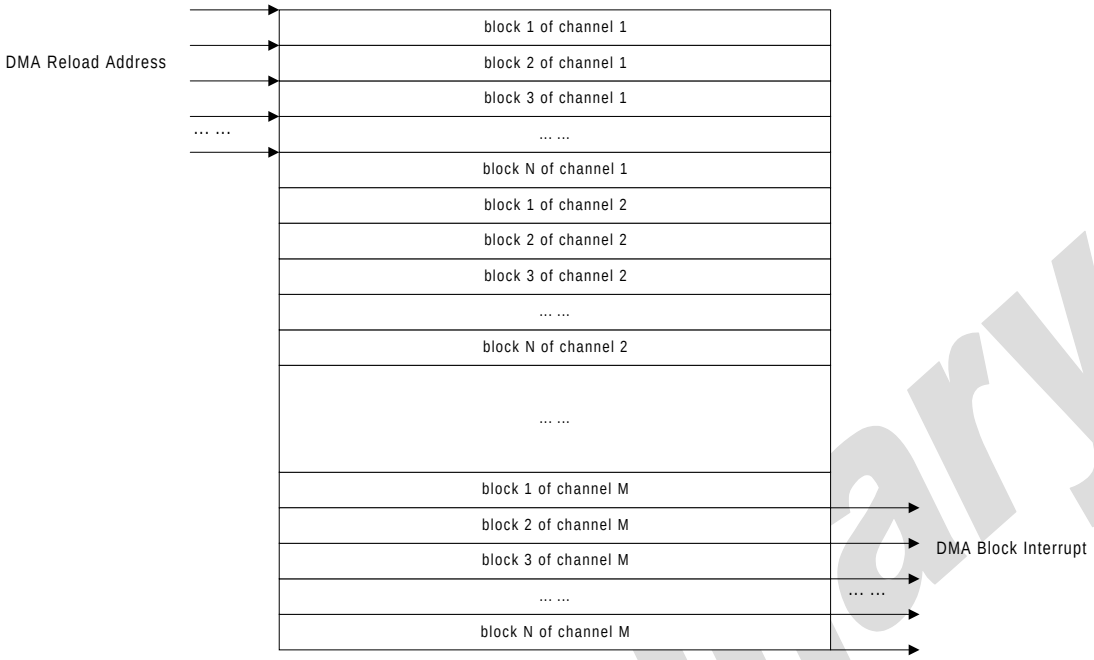
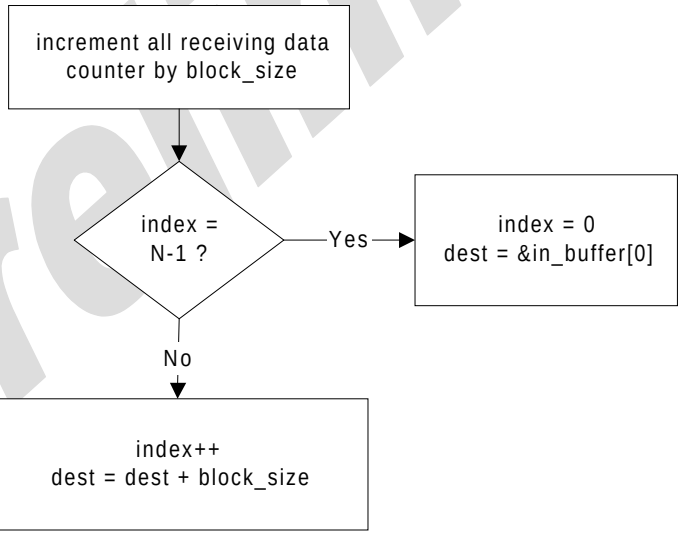


Figure 2. Block Diagram of DMA Block Interrupt Service Routine



For the transmission process, you must decrement all of the transmit counters by the value of block size.

In the following example, the system must support two channels and two vocoders (G.723 and G.729). The frame size for G.723 is 30 ms (240 samples @8 kHz sample rate), and the frame size for G.729 is 10 ms (80 samples @8 kHz sample rate). The greatest common divisor of 240 and 80 is 80. Therefore, the block size chosen is 80, and the buffer size is 720 (three G.723 frames). Figure 3 shows the main program. Figure 4 shows the ISR.

Figure 3. Source Code for the Main Program

```
//main.c
//test dma on C6201 EVM
//
// define McBSP0 registers
//
#define McBSP0_DRR   0x18c0000  //addresses of data receive reg.
#define McBSP0_DXR   0x18c0004  //addresses of data transmit reg.
#define McBSP0_SPCR  0x18c0008  //address of serial port contl. reg.
#define McBSP0_RCR   0x18c000C  //address of receives control reg.
#define McBSP0_XCR   0x18c0010  //address of transmits control reg.
#define McBSP0_SRGR  0x18c0014  //address of sample rate generator
#define McBSP0_MCR   0x18c0018  //address of multi-channel reg.
#define McBSP0_RCER  0x18c001C  //address of receives channel enable
#define McBSP0_XCER  0x18c0020  //address of transmits ch. enable
#define McBSP0_PCR   0x18c0024  //address of pin control reg.

// define interrupt registers
//
#define IMH          0x19c0000       //Interrupt Multiplexer High
#define IML          0x19c0004       //Interrupt Multiplexer Low

// define DMA channel 0
//
#define DMA0_DST     0x1840018       //DMA0 destination
#define DMA0_PCR     0x1840000       //dma0 primary control
#define DMA0_SCR     0x1840008       //dma0 secondary control
#define DMA0_SRC     0x1840010       //dma0 source
#define DMA0_CNT     0x1840020       //dma0 transfer count

// define DMA global registers

#define DMA_RELOAD_A  0x1840028       //dma global reload a
#define DMA_INDEX_A   0x1840030       //dma global index a
#define DMA_ADR_B     0x184003C       //dma global address reload b


#include <stdio.h>
#include <stdlib.h>

    extern cregister volatile unsigned int CSR;
    extern cregister volatile unsigned int IFR;
    extern cregister volatile unsigned int ISR;
    extern cregister volatile unsigned int ICR;
    extern cregister volatile unsigned int IER;

#define DATA_SIZE  2
#define BLOCK_SIZE 80
#define NUM_BLOCK  9
```

```
#define CHANNEL 2
#define BUFFFER_SIZE   NUM_BLOCK*BLOCK_SIZE
#define FRAME_INDEX    -(((CHANNEL-1)*BUFFER_SIZE)-1)*DATA_SIZE
#define ELEMENT_INDEX  BLOCK_SIZE*DATA_SIZE
#define DMA_CNT        ((BLOCK_SIZE<<16) + CHANNEL)
#define DMA_INDEX      ((FRAME_INDEX<<16) + ELEMENT_INDEX)

short  in_buffer[CHANNEL*BUFFER_SIZE];
int    dest;
short  i, index, data_counter[2];

void main()                      //main program
{
    short i;
    CSR=0x100;
    IER=1;
    ICR=0xffff;
    index = 1;
    data_counter[0] = 0;
    data_counter[1] = 0;
    *(int *)McBSP0_SPCR = 0;     //reset serial port
     dest = (int)&codec_in[0]+ BLOCK_SIZE*DATA_SIZE;

// Config DMA0 for RX0 (CODEC)

    *(int *)DMA0_SCR = 0x80;
    *(int *)DMA0_SRC = McBSP0_DRR;          //RX0 is the source
    *(int *)DMA0_DST =(int)&in_buffer[0];   //in_buffer is the dest
    *(int *)DMA0_PCR =0x4a0341c0 ;  //set dma0 primary control
    *(int *)DMA0_CNT =DMA_CNT;     //set dma0 block transfer count
    *(int *)DMA_INDEX_A=DMA_INDEX;
    *(int *)DMA_RELOAD_A = DMA_CNT; //set dma0 transfer count
    *(int *)DMA_ADR_B = dest;

    *(int *)DMA0_PCR |= 3;     //Start DMA0 with auto-initialization

// set up McBSP0
    *(int *)McBSP0_PCR = 0;   //set pin control reg.;
    *(int *)McBSP0_RCR = 0x10140;//set rx ctl reg. 2-16bit data/frame
    *(int *)McBSP0_XCR = 0x10140;//set tx ctl reg. 2-16biydata/frame
    *(int *)McBSP0_SPCR = 0x12001;//setup SP ctl reg. for slave;


    *(int *)IML &= 0xfffffff0;            //clr content of INT4
    *(int *)IML |= 0x8;                   //assign DMA0 to INT4,
    IER |= 0x12;                          //unmask INT4
    CSR|= 1;                              //enable interrupt

Loop:
/*
     check and adjust the data counter,
     execute the channel processing module
*/
    goto Loop;
}
```

Figure 4. Source Code for the DMA Block ISR

```
interrupt void
DMA0_ISR()
{
    *(int *)DMA0_SCR = 0x80;  //clear block condition bit
    data_counter[0]+=BLOCK_SIZE;  //increment data counter values
    data_counter[1]+=BLOCK_SIZE;
    if (index == NUM_OF_BLOCK-1)
    {
         index = 0;
         dest =(int)&in_buffer[0];
    }
    else
    {
         index++;
         dest = dest + BLOCK_SIZE*DATA_SIZE;
    }
     *(int *)DMA_ADR_B = dest;
}
```

References

Refer to the following application reports to learn more about the multichannel/algorithm system on the TMS320C6000 DSP.

1. Xiangdong Fu and Zhaohong Zhang, *A Multichannel/Algorithm Implementation on the TMS320C6000 DSP*, SPRA556, July 1999, Texas Instruments.
2. Xiangdong Fu and Zhaohong Zhang, *Host-Side Design of a Multichannel/Algorithm System on the TMS320C6000 DSP*, SPRA558, July 1999, Texas Instruments.
3. Zhaohong Zhang and Xiangdong Fu, *Target-Side Design of a Multichannel/Algorithm System on the TMS320C6000 DSP*, SPRA560, July 1999, Texas Instruments.

TI Contact Numbers

INTERNET

TI Semiconductor Home Page
www.ti.com/sc

TI Distributors
www.ti.com/sc/docs/distmenu.htm

PRODUCT INFORMATION CENTERS

Americas
Phone +1(972) 644-5580
Fax +1(972) 480-7800
Email sc-infomaster@ti.com

Europe, Middle East, and Africa
Phone
Deutsch +49-(0) 8161 80 3311
English +44-(0) 1604 66 3399
Español +34-(0) 90 23 54 0 28
Francais +33-(0) 1-30 70 11 64
Italiano +33-(0) 1-30 70 11 67
Fax +44-(0) 1604 66 33 34
Email epic@ti.com

Japan
Phone
International +81-3-3344-5311
Domestic 0120-81-0026
Fax
International +81-3-3344-5317
Domestic 0120-81-0036
Email pic-japan@ti.com

Asia
Phone
International +886-2-23786800
Domestic
Australia 1-800-881-011
TI Number -800-800-1450
China 10810
TI Number -800-800-1450
Hong Kong 800-96-1111
TI Number -800-800-1450
India 000-117
TI Number -800-800-1450
Indonesia 001-801-10
TI Number -800-800-1450
Korea 080-551-2804
Malaysia 1-800-800-011
TI Number -800-800-1450
New Zealand 000-911
TI Number -800-800-1450
Philippines 105-11
TI Number -800-800-1450
Singapore 800-0111-111
TI Number -800-800-1450
Taiwan 080-006800
Thailand 0019-991-1111
TI Number -800-800-1450
Fax 886-2-2378-6808
Email tiasia@ti.com

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products 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 APPLICATIONS 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, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

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

TI assumes no liability for applications assistance or customer product design. TI does not 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. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty, or endorsement thereof.