PC/TMS320C54x Evaluation Module Communication Interface

P. Geremia

Literature Number: BPRA051
Texas Instruments Europe
March 1997

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.

Contents

List of Figures

List of Tables

PC/TMS320C54x Evaluation Module Communication Interface

ABSTRACT

This application note describes, and illustrates with examples, how to communicate between a host PC and the TMS320C54x Evaluation Module (EVM) using the handshake-free communication channels A and B. The software includes basic C functions for the host, C and assembler routines for the EVM as well as communication examples.

1. Introduction

The TMS320C54x evaluation module (EVM) is a PC/AT plug-in that lets you evaluate characteristics of the ‘C54x digital signal processor (DSP) to see if the DSP meets your application requirements. You can also create your software to run on board or expand the system in a variety of ways.

The host/target communication system provides a simple way to pass data between the target (EVM) and the host (PC) while maintaining real-time operation. The message protocol is **user-defined**. The system can be interrupt-driven, polled, or a mixture of the two. Channels A and B can be used to pass information between the host and the target and can be used independently or together.

In this document, you will find a brief overview of the host/target communication interface and useful, ready-to-use software modules illustrated by comprehensive examples.

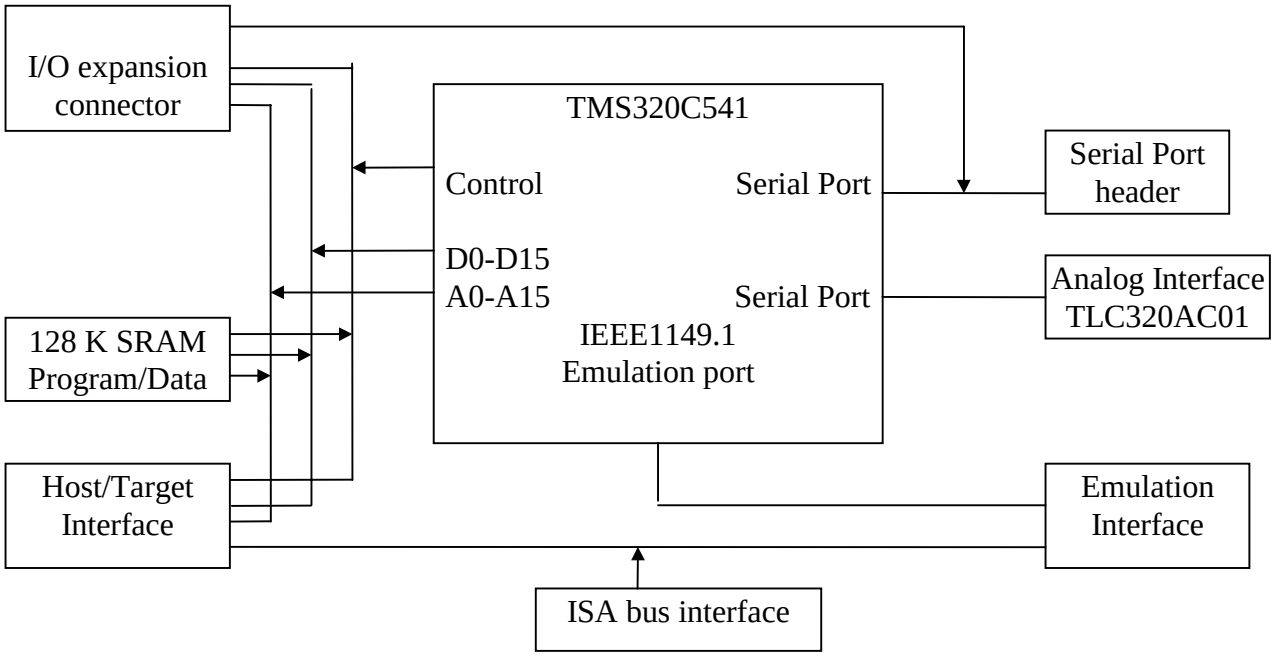

Figure 1: TMS320C54x EVM Block Diagram

2. PC/AT Host Interface

The host interface provides the buffering, host I/O decode, and access control. The host communicates to the EVM via 38 16-bit I/O locations. The first 32 I/O locations support emulation; six locations support host/target communications and controls. Each I/O is defined with an offset from the base address (Table 1: PC I/O offset for communication channels).

Table 1: PC I/O offset for communication channels

I/O offset	Register	Size	Register type
0x800	Channel A	16	read/write
0x802	Reserved	16	read/write
0x804	Channel B	16	read/write
0x806	Channel B	16	read/write
0x808	Status/Control	16	read/write
0x80A	Status/Control	16	read/write

The channel A is a single, bi-directional register for transferring data:

- writing operation overwrites the current value and generates an interrupt (INT1_) to the target, sets the AXST bit (bit 0) in the host control register, and sets the ARST (bit 1) in the target control register.
- reading operation clears the ARST bit (bit 1) in the host control register and clears the AXST bit (bit 0) in the target control bit register.

The channel B is a 64-deep, bi-directional FIFO register for transferring data:

- writing operation (to either offset 0x804 or 0x806) fills the FIFO. Data is ignored if the FIFO is full.
- writing operation to offset 0x806 has the side effect of generating an interrupt (INT2_) to the target. A transmit-FIFO-full signal also generates an INT2_ to the target.
- reading operation is the same for either offset 0x804 or 0x806.

The status/control register provides system-level control and status for the EVM. Host writes to offsets 0x808 and 0x80A are identical except that writes to offset 0x80A reset the ARST, AXST, BRST, BXST flags for both host and target and the BRST2 flag for the host. This should be done by the host at the start of the host/target communication. Note that the debugger resets (via the RESET bit) only the target and emulation.

The host control register is defined in Table 2: Host Control Register overview (I/O offset 0x808 and 0x80A) and the bit definitions are defined in Table 3: Host Control Register description (I/O offset 0x808 and 0x80A).

Table 2: Host Control Register overview (I/O offset 0x808 and 0x80A)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	MPMC	BTIE	ATIE	HBIO	BRST2	BRST		REV1	REV0	XF	BIO	BXST		ARST	AXST
RW	RW	RW	RW	RW	R	R	R	R	R	R	R	R	R	R	R

Table 3: Host Control Register description (I/O offset 0x808 and 0x80A)

Bit	Name	Description
0	AXST	Channel A transmit status. If AXST=1, the host has written to its channel A register. The AXST flag is cleared when the target reads channel A.
1	ARST	Channel A receive status. If ARST=1, the target has written to its channel A register. The ARST flag is cleared when the host reads channel A
3-2	BXST	Channel B transmit status: **Bit 3 Bit2 Channel B transmit Status** 0 0 buffer empty 0 1 buffer less than half full 1 1 buffer half or more full 1 0 buffer full
4	BIO	BIO_ input to target processor
5	XF	External flag from target processor (status)
6	REV0	Card revision status bit 1
7	REV1	Card revision status bit 0
9-8	BRST	Channel B receive status: **Bit9 Bit8 Channel B receive Status** 0 0 buffer empty 0 1 buffer less than half full 1 1 buffer half or more full 1 0 buffer full
10	BRST2	Channel B receive status bit 2. if BRST2=1, the target has written to channel B, forcing an interrupt. The BRST2 flag is cleared when the host reads its channel B
11	HBIO	host BIO_ signal to target processor
12	ATIE	Channel A target interrupt enable. If ATIE=1, the channel A receive conditions generate a host interrupt
13	BTIE	Channel B target interrupt enable. If ATIE=1, the channel B receive conditions generate a host interrupt
14	MP/MC_	Microprocessor/Microcomputer mode select. The EVM powers in micro-Computer mode
15	RESET	Software reset. If RESET=1, the target processor and emulation logic are reset, but the host/target communication flags are not reset.

3. TMS320C54x EVM - Target Interface

On the target side, the channel A is a single 16-bit bi-directional register mapped into two I/O port locations and the channel B is a single, bi-directional, 64-deep FIFO buffer that is also mapped into two I/O port locations. Moreover, sixteen parallel I/O ports (PIO) are for applications and are available on the parallel expansion connector J1. The I/O mapping is given in Table 4. All I/O accesses require at least two wait states.

Table 4: I/O address for communication channels

Port Address	Name	Usage
0x0 to 0xF	PIO	User expansion
0x10	Channel A	Communications
0x11	Channel A	Communications
0x12	Channel B	Communications
0x13	Channel B	Communications
0x14	Status	Target status/control

When using Channel A, the communication rules are defined as follows:

- reading operations are identical for both locations.
- writing operation to location 0x10 generates a host interrupt **only** if the host control bit ATIE is set. Writing operation to location 0x11 generates a host interrupt **regardless** of the value of ATIE. The only exception to this is when both ATIE and BTIE have been zero since the last interrupt was serviced. In this case, it is necessary to cycle either ATIE or BTIE high and then low in order to use this feature (target write generating a host interrupt).

Channel B has the following rules:

- reading operations are identical for both locations.
- If the host control bit BTIE is set, writes to either location that fills the buffer generate a host interrupt. Additionally if the BTIE is set, writes to port 0x13 generate a host interrupt whether the FIFO is full or not.

The status I/O port is for target control and provides general-purpose control, status, and discrete bit I/O. Refer to Table 5 for a description of the target control register. USRBIN0 and USRBIN1, USRBOT0 and USRBOT1and USRBOT2 are discrete TTL-compatible inputs and outputs, respectively, that are available for use on the expansion connectors. Inputs are pulled high with 4.7 kΩ resistors.

Table 5: Target Control Register overview (port address 0x14)

15	14	13	12	11 10 9 8	7	6	5 4	3 2	1	0
AIC-RST	USR-BOT2	USR-BOT1	USR-BOT0	Reserved	USR-BIN1	USR-BIN0	BRST	BXST	ARST	AXST
RW	RW	RW	RW	R R R R	R	R	R R	R R	R	R

A description of the target control register bits is given in Table 6.

Table 6: Target Control Register description (port address 0x14)

Bit	Name	Description
0	AXST	Channel A transmit status. If AXST=1, the target has written to its channel A register. The AXST flag is cleared when the host reads channel A.
1	ARST	Channel A receive status. If ARST=1, the host has written to its channel A register. The ARST flag is cleared when the target reads channel A
3-2	BXST	Channel B transmit status: **Bit 3 Bit2 Channel B transmit Status** 0 0 buffer empty 0 1 buffer less than half full 1 1 buffer half or more full 1 0 buffer full
5-4	BRST	Channel B receive status: **Bit5 Bit4 Channel B receive Status** 0 0 buffer empty 0 1 buffer less than half full 1 1 buffer half or more full 1 0 buffer full
6	USRBIN0	User discrete input bit 0
7	USRBIN1	User discrete input bin 1
8	-	Reserved
9	-	Reserved
10	-	Reserved
11	-	Reserved
12	USRBOT0	User discrete output bit 0
13	USRBOT1	User discrete output bit 1
14	USRBOT2	User discrete output bit 2
15	AICRST	If AICRST=0, the analog interface circuit is reset

4. Software Modules

The software modules provided include ready-to-use functions for handling PC/EVM communications for both host (C-routines) and target (C and Assembler routines) as well as comprehensive examples.

4.1 Host Communication Routines

The functions are available in the module HOSTIO (HOSTIO.H, HOSTIO.C). The I/O offset is defined in HOSTIO.H as well as the base address for the host (if you have changed this address, please modify the corresponding value e.g. BASE - refer to the TMS320C54x Evaluation module, Technical reference page 2-4). The available routines are:

- porttype read_A(void);
  This function is used to read from Channel A.
- void write_A(porttype d);
  This function is used to write to Channel A.
- porttype read_B(void);
  This function is used to read from Channel B.
- void write_B(unsigned int portaddress,porttype d);
  This function is used to write to Channel B (either at offset 0x804 or 0x806).
- void reset_HI(porttype d);
  This function is used to reset the host/target interface flags and to program the host control register.
- void host_setup_com(void);
  This function is used to initialize the communication with the EVM.

Note: porttype is a 16bit type (here type-defined as integer)

4.1.1 read_A()

This function checks if there is data available in channel A by polling the ARST bit from host control register. When data is available, the function reads the value and exits.

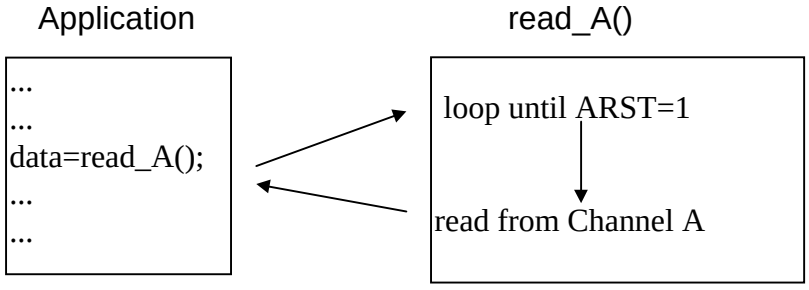

Figure 2: read_A() function (HOSTIO)

4.1.2 write_A()

This function checks if the buffer is empty in order not to overwrite the previous value by polling the AXST bit from host control register. When the buffer is empty, the function writes the value and exits.

In either case, a target interrupt (INT1_) is generated.

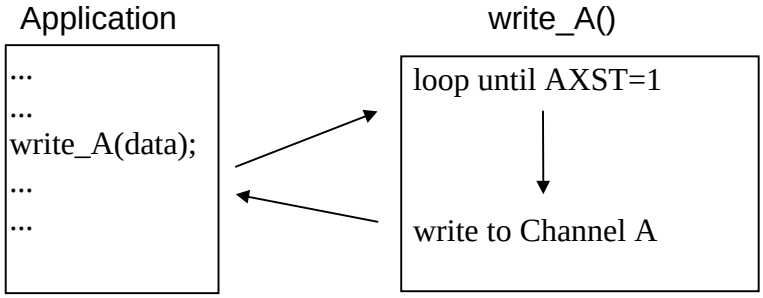

Figure 3: write_A() function (HOSTIO)

4.1.3 read_B()

This function checks if the buffer is not empty by polling the BRST bits from host control register. When data is available, the function writes the value and exits.

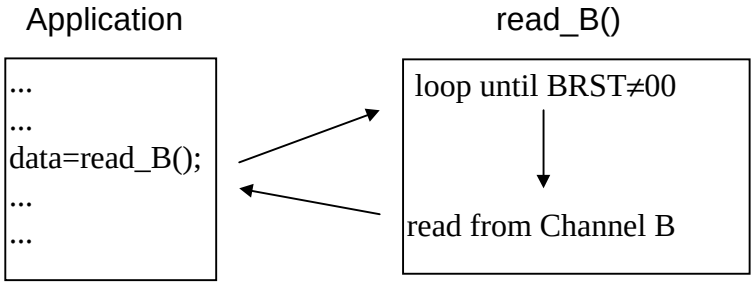

Figure 4: read_B() function (HOSTIO)

4.1.4 write_B()

This function checks that the buffer is not already full by polling the BXST bits from host control register. When it is not, the function writes the value to the address specified in its parameters (e.g. should be CH_B or CH_B_I) and exits. Writing to address CH_B_I generates a target interrupt (INT2_).

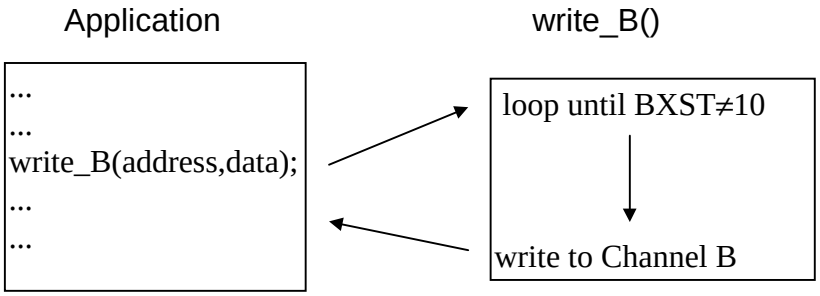

Figure 5: write_B() function (HOSTIO)

4.1.5 reset_HI()

This function programs the host control register with the value given as parameter and resets all host/target communication flags. The function does not reset either the target or the emulation.

4.1.6 host_setup_com()

This function generates the following sequence on the BIO_ pin until it receives an acknowledge from the target in the channel A:

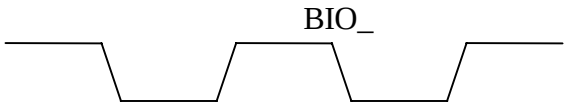

Figure 6: BIO_ signal for setting host/target communication

4.2 Target Communication Routines

For the target, two modules are available: the first module is purely in C and the second one written solely in assembly language. The C-functions are available in the module EVMIO (EVMIO.H, EVMIO.C). For the assembler routines, each routine is a macro which can be used to build a macro library (here called EVMIO.LIB) for the target communication interface.

4.2.1 C-functions

The available functions are listed below:

- porttype rdio_A(void);
  This function is used to read from Channel A.
- void wrio_a(porttype d);
  This function is used to write to Channel A. It generates a host interrupt if ATIE is set.
- void wrio_a_HINT(porttype d);
  This function is used to write to Channel A. It generates a host interrupt regardless of ATIE.
- porttype rdio_b(void);
  This function is used to read from Channel B.
- void wrio_b(porttype d);
  This function is used to write to Channel B. If BTIE is set and the buffer full, a host interrupt is generated.
- porttype wrio_b_HINT(porttype d);
  This function is used to write to Channel B. If BTIE is set, a host interrupt is always generated.
- void evm_setup_com(void);
  This function is used to initialize the communication with the host.

Note: porttype is a 16bit type (here type-defined as integer)

4.2.1.1 rdio_a()

This function checks if there is data available in channel A by polling the ARST bit from target control register. When data is available, the function reads the value and exits.

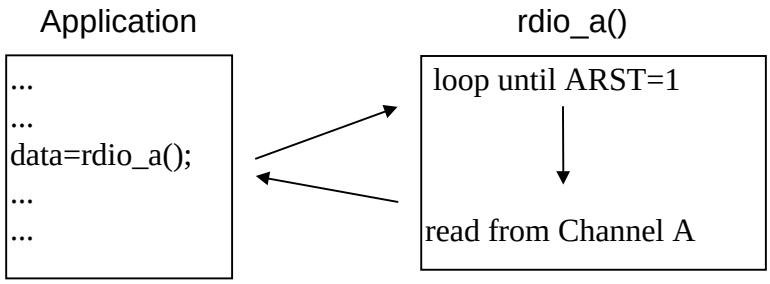

Figure 7: rdio_a() function (EVMIO)

4.2.1.2 wrio_a()

This function checks if the buffer is empty in order not to overwrite the previous value by polling the AXST bit from target control register. When the buffer is empty, the function writes the value and exits.

Host interrupt is generated depending on ATIE bit.

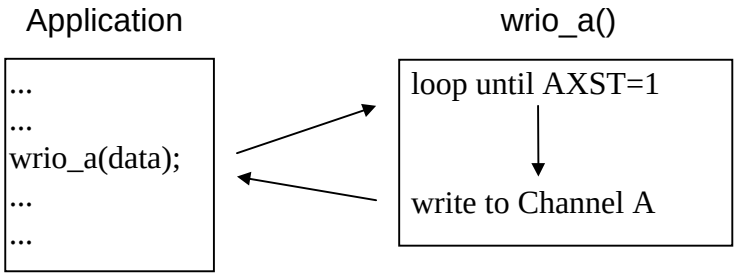

Figure 8: wrio_a() function (EVMIO)

4.2.1.3 wrio_a_HINT()

This function checks if the buffer is empty in order not to overwrite the previous value by polling the AXST bit from target control register. When the buffer is empty, the function writes the value and exits.

Host interrupt is generated regardless of ATIE bit.

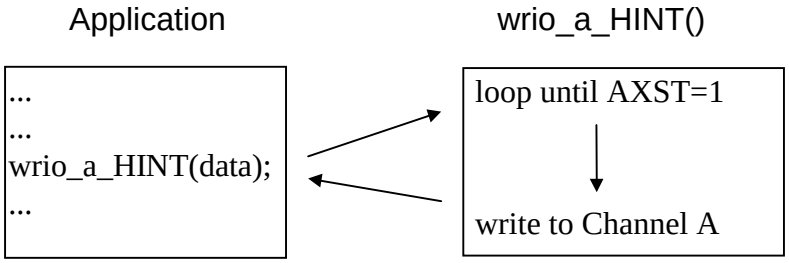

Figure 9: wrio_a_HINT() function (EVMIO)

4.2.1.4 rdio_b()

This function checks if the buffer is not empty by polling the BRST bits from target control register. When data is available, the function reads the value and exits.

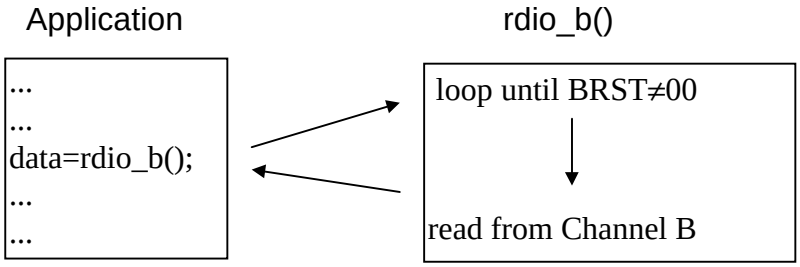

Figure 10: rdio_b() function (EVMIO)

4.2.1.5 wrio_b()

This function checks if the buffer is not already full by polling the BXST bits from target control register. When it is not, the function writes the value and exits. If the buffer is full and BTIE is set, a host interrupt is generated.

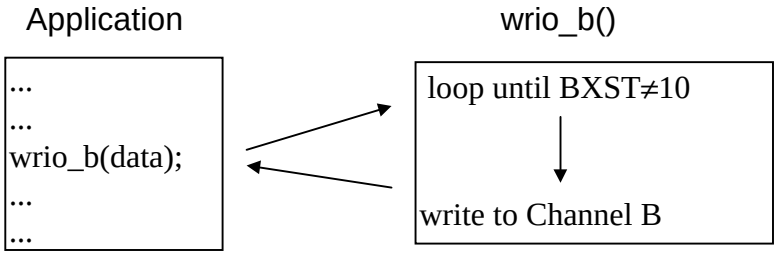

Figure 11: wrio_b() function (EVMIO)

4.2.1.6 wrio_b_HINT()

This function checks the buffer is not already full by polling the BXST bits from target control register. When it is not, the function writes the value and exits. If BTIE is set, then a host interrupt is generated.

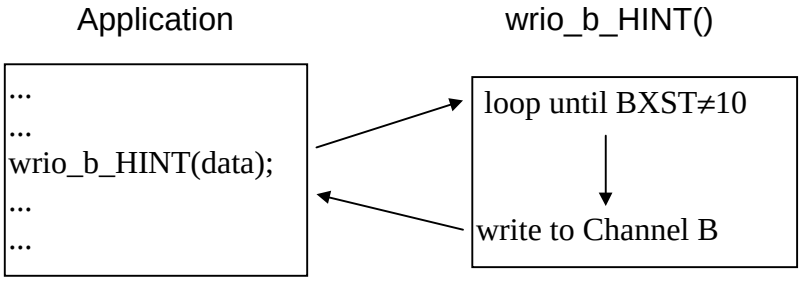

Figure 12: wrio_b_HINT() function (EVMIO)

4.2.1.7 evm_setup_com()

This function loops until the sequence generated by the host_setup_com() function (**Error! Reference source not found.**) appears on the BIO_ pin and then sends an acknowledge to the host using the channel A.

4.2.2 Assembler routines

The assembler routines are defined as macros that you can use either by getting them from the macro library (EVMIO.LIB) or by including them individually. The following macros are available (the macro parameter addr selects the address to read from/write to and should be a 16-bit single data-memory operand (SMEM)) :

- rdio_a .macro addr
  This macro is used to read from Channel A. (file RDIO_A.ASM)
- wrio_a .macro addr,int
  This macro is used to write to Channel A. If the macro parameter int is omitted, it generates a host interrupt if ATIE is set. If the macro parameter int is equal to INT then a host interrupt will be generated regardless of ATIE. (file WRIO_B.ASM)
- rdio_b .macro addr
  This macro is used to read from Channel B. (file RDIO_B.ASM)
- wrio_b .macro addr,int
  This macro is used to write to Channel B. If the macro parameter int is omitted, it generates a host interrupt if BTIE is set and if the FIFO is full. If the macro parameter int is equal to INT then a host interrupt will be always generated if BTIE is set. (file WRIO_B.ASM)
- evmsetup .macro
  This macro is used to initialize a communication with the host. (EVMSETUP.ASM)

These macros use the accumulator A so, if needed by your application software, you should save it (AG,AH and AL) before using the macro. An example is given in the test programs. The algorithms used by these macros are the same as used in the corresponding C-functions.

The file IOPORT.INC contains some I/O addresses used by the target interface and should be included in your application software.

4.2.2.1 rdio_a .macro addr

The function checks if there is data available in channel A by polling the ARST bit from target control register. When data is available, the macro reads the value and exits.

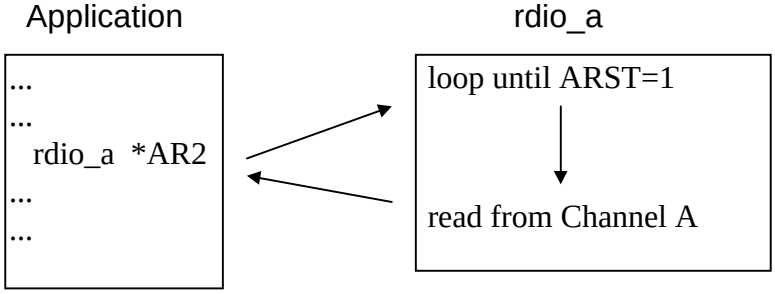

Figure 13: rdio_a macro (EVMIO.LIB)

4.2.2.2 wrio_a .macro addr,int

The function checks if the buffer is empty in order not to overwrite the previous value by polling the AXST bit from target control register. When the buffer is empty, the macro writes the value and exits.

Host interrupt is generated regardless of ATIE bit if the application software does not omit the int parameter (ex: wrio_a *(#100h),INT), otherwise a host interrupt is generated only if ATIE is set.

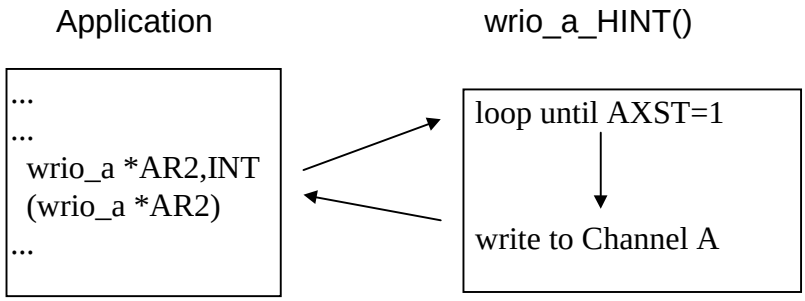

Figure 14: wrio_a macro (EVMIO.LIB)

4.2.2.3 rdio_b .macro addr

The function checks if the buffer is not empty by polling the BRST bits from target control register. When data is available, the function reads the value and exits.

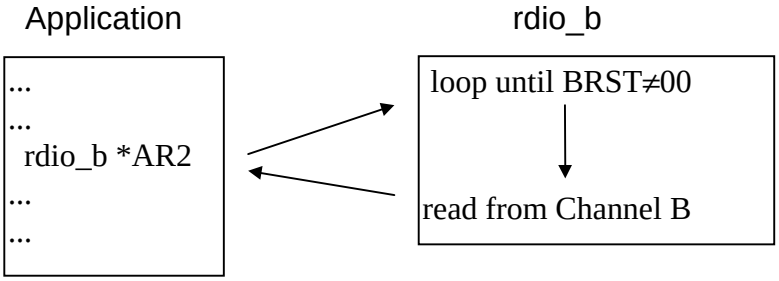

Figure 15: rdio_b macro (EVMIO.LIB)

4.2.2.4 wrio_b .macro addr,int

The macro checks the buffer is not already full by polling the BXST bits from target control register. When it is not, the macro writes the value and exits. If the parameter int is not omitted (ex: wrio_b *AR2,INT) and if BTIE is set, then a host interrupt is always generated. If int is omitted and BTIE is set, then a host interrupt is generated only if the buffer is full.

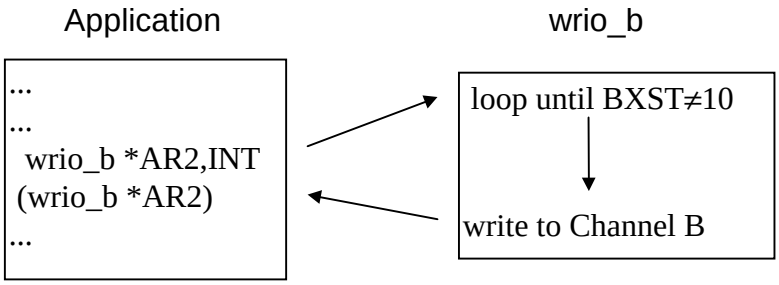

Figure 16: wrio_b macro (EVMIO.LIB)

4.2.2.5 evmsetup .macro

This macro loops until the sequence generated by the host_setup_com() function appears on the BIO_ pin and then sends an acknowledge to the host using the channel A.

4.3 Test Programs

A few test programs are delivered along with the communication routines (see Table 7). Each is associated with its own batch file which basically performs the following steps:

- download code into EVM memory and release RESET signal, thus running the EVM program
- run the executable on the PC.

IMPORTANT:

* When using the MAKEFILE files or the batch files, make sure you have checked for your own path settings.

** When running the examples, make sure you give numbers large enough for the block size and the number of blocks otherwise the program will not be able to calculate the transfer speed.

Table 7: Examples description matrix

Example number	PC	EVM
1	HOST.EXE (non-interrupt driven communication using C module HOSTIO)	TARGET.OUT (interrupt driven communications using assembler routines from EVMIO library with context saving)
2	HOST1.EXE (non-interrupt driven communication using C module HOSTIO)	TARGET1.OUT (non-interrupt driven communications using assembler routines from EVMIO library with no context saving)
3	HOST2.EXE (non-interrupt driven communication using C module HOSTIO)	TARGET2.OUT (non-interrupt driven communications using C module EVMIO with no context saving)

4.3.1 Example 1

This example illustrates a communication from host to target using channels A and B. The communication is interrupt-driven (EVM is interrupted). On the EVM side, the accumulator A content is saved and restored before and after data transfers.

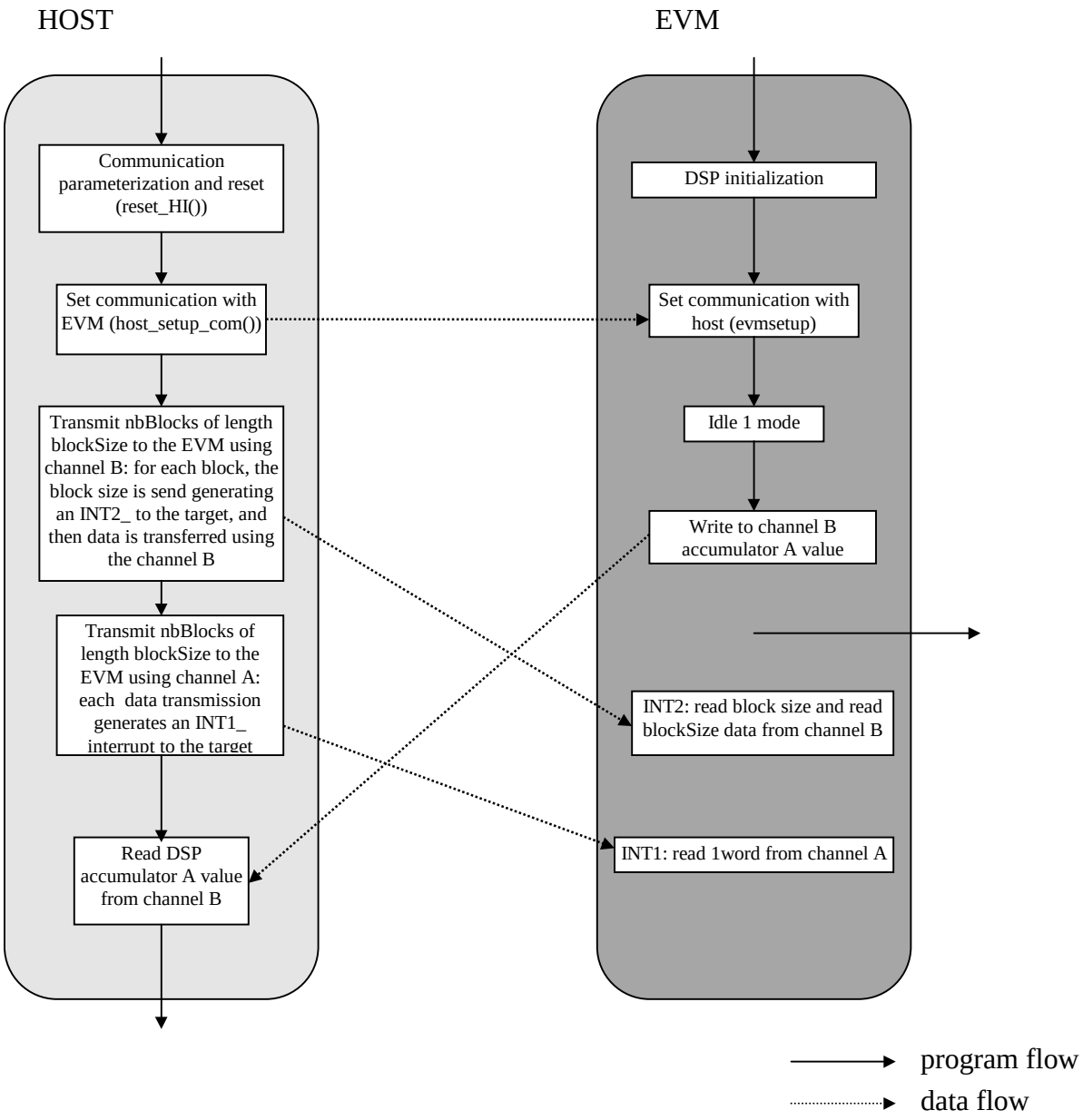

Figure 17: host/target communication, example 1

4.3.2 Example 2

This example illustrates a communication from host to target and from target to host using channel A. The communication is not interrupt-driven. In this example, on the EVM side, the accumulator A content is not preserved by the software during the data transfers.

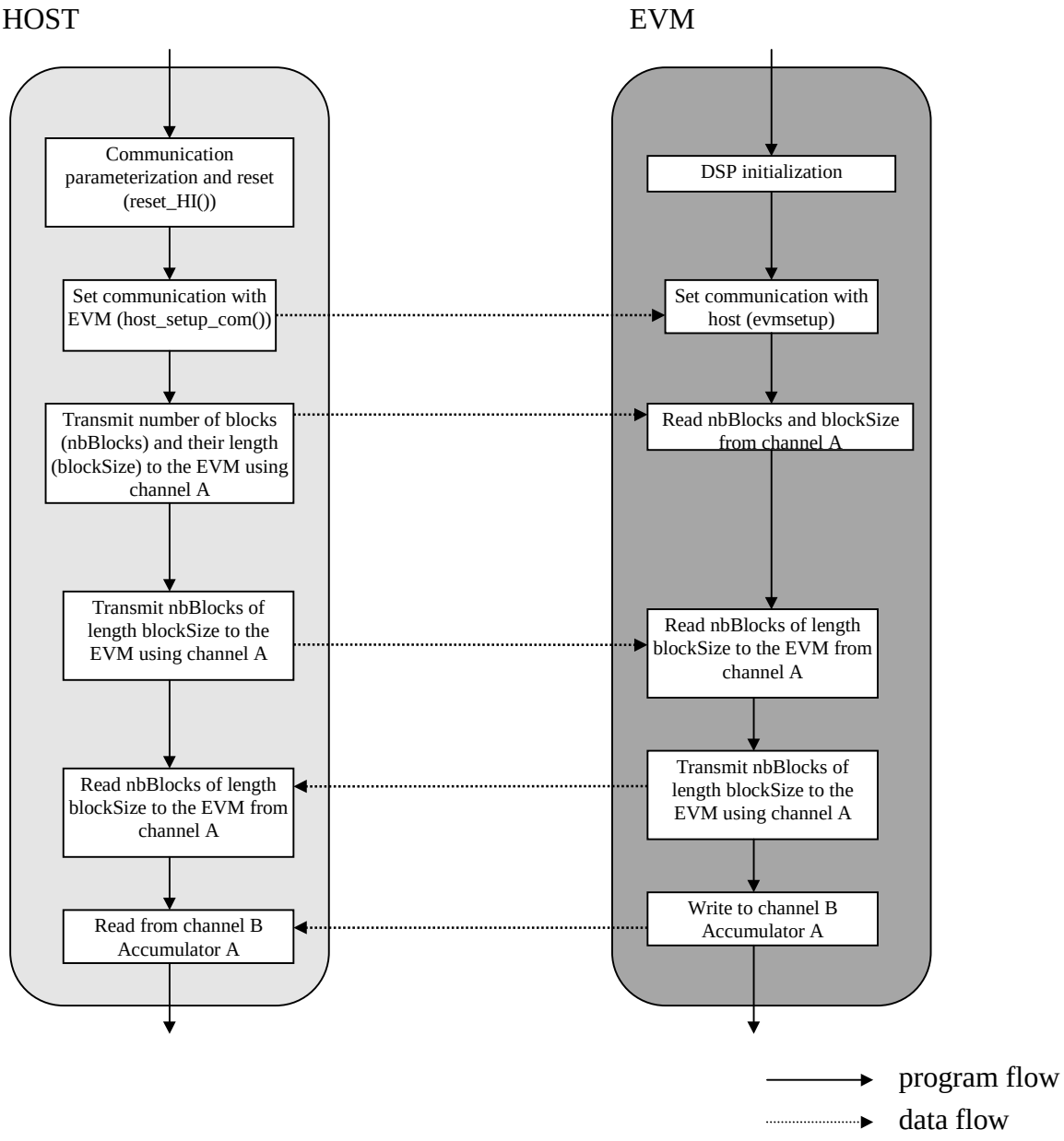

Figure 18: host/target communication, example 2

4.3.3 Example 3

This example illustrates a communication from host to target and from target to host using channels A and B. The communication is not interrupt-driven. In this example, the EVM software is written in C using the EVMIO module.

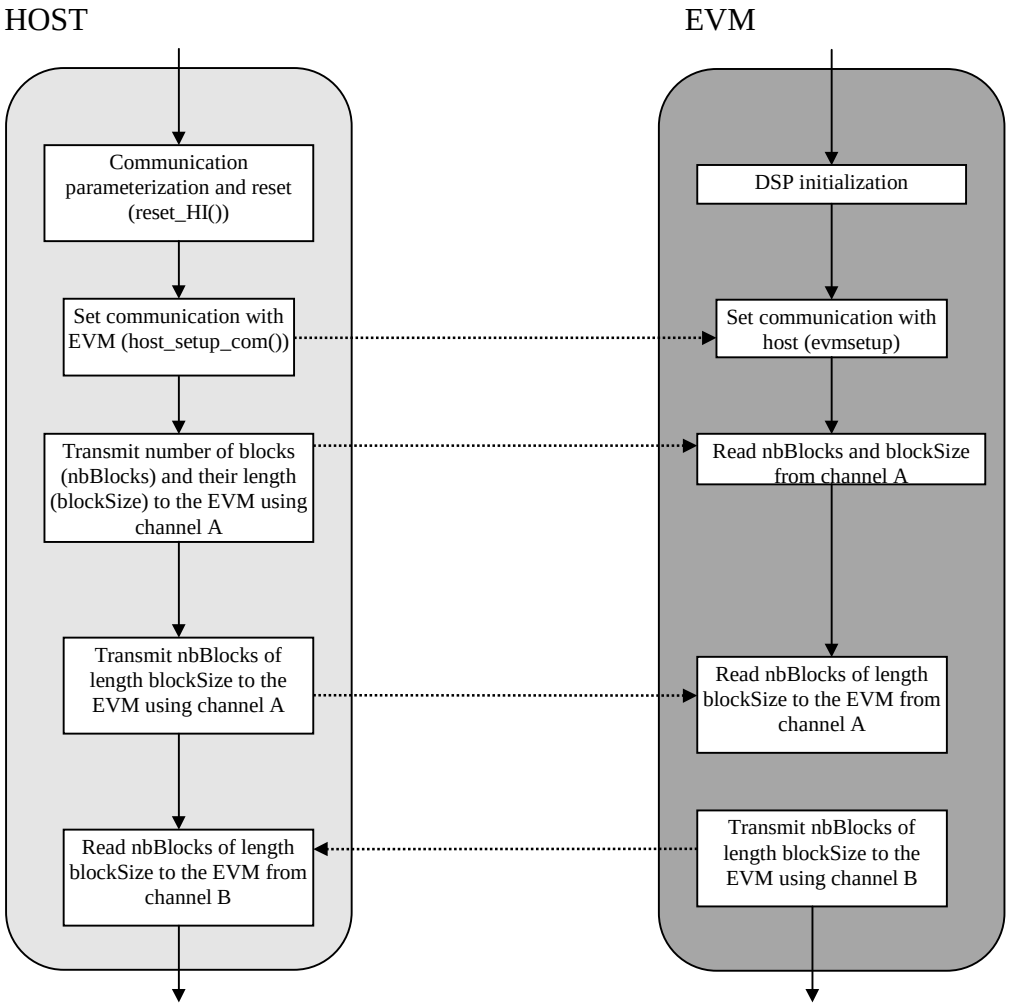

Figure 19: host/target communication, example 3

5. Conclusion

The aim of this application note was to provide basic understanding of the communication between a host PC and the TMS320C54x Evaluation Module as well as to provide basic functions/routines for use in software where host/target communication is needed. The use of these basic functions/routines is illustrated by comprehensive examples to shorten development time for user-defined applications.

6. Host communication modules listings

6.1 HOSTIO.H

```
/*********************************************************
 *                                                       *
 *  (C) Copyright 1996, Texas Instruments Incorporated   *
 *                                                       *
 *********************************************************/

/* C54x host communication routines */


#ifndef __HOSTIO
#define __HOSTIO

/* declare i/o ports */
typedef int porttype;   /* type for ioports should be 16 bits */

/* give them mnemonics */

#define BASE    0x240   /* base address (what you set your switches to) */
#define CH_A    0x800   /* offsets from base */
#define CH_B    0x804
#define CH_B_I  0x806
#define STATUS  0x808
#define RST_HI  0x80a

/* read from channel A */
porttype       read_A(void);

/* write to channel A */
void write_A(porttype d);

/* read from channel B */
porttype read_B(void);

/* write to channel B */
void write_B(unsigned int portaddress,porttype d);

/* reset host/target communication flags */
void reset_HI(porttype d);

/* set connection with EVM */
void host_setup_com(void);

#endif
```

6.2 HOSTIO.C

```
/*********************************************************
 *                                                       *
 *  (C) Copyright 1996, Texas Instruments Incorporated   *
 *                                                       *
 *********************************************************/

/* C54x host communication routines */

#include <dos.h>
#include "hostio.h"

/* read values from channel A */
porttype read_A()
{
       /* wait for data */
       while ((inport(BASE+STATUS) & 0x0002) == 0);
       /* read data */
       return (porttype)inport(BASE+CH_A);
}


/* write values to channel A */
void write_A(porttype d)
{
       /* make sure we don't overwrite */
       while ((inport(BASE+STATUS) & 0x0001) != 0);
       /* write data */
       outport(BASE+CH_A,(int)d);
```

```
        return;
}

/* read values from channel B */
porttype read_B()
{
        /* wait for data */
         while ((inport(BASE+STATUS) & 0x0300) == 0);
         /* read data */
         return (porttype)inport(BASE+CH_B);
}

/* write values to channel B */
void write_B(unsigned int portaddress,porttype d)
{
        /* buffer full? */
         while ((inport(BASE+STATUS) & 0x000c) == 0x08);
         /* write data */
         outport(BASE+portaddress,(int)d);
         return;
}

/* reset the Host Interface */
void reset_HI(porttype d)
{
        /* make sure don't reset evm */
        outport(BASE+RST_HI, ~0x8000 & d);
        return;
}

/* Setup communication with EVM */
void host_setup_com(void)
{
        /* check for acknowledgment from EVM */
        while ((inport(BASE+STATUS)&0x2)!=0x2)
        {
                /* pulse HBIO */
                outport(BASE+STATUS,inport(BASE+STATUS) |  0x800);
                outport(BASE+STATUS,inport(BASE+STATUS) & ~0x800);
        }
        /* receive acknowledgment */
        inport(BASE+CH_A);
        return;
}
```

7. Target communication modules/routines listings

7.1 EVMIO.H

```
/*******************************************************
 *                                                     *
 *  (C) Copyright 1996, Texas Instruments Incorporated   *
 *                                                     *
 *******************************************************/

/* C54x evm communication routines */

#ifndef __EVMIO
#define __EVMIO


#define DUMMY  0xffff

/* type for ioports should be 16 bits */
typedef int porttype;

/* read values from channel A */
porttype rdio_a();

/* write values to channel A */
void wrio_a(porttype d);

/* write values to channel A - generate host interrupt */
void wrio_a_HINT(porttype d);

/* read values from channel B */
porttype rdio_b();

/* write values to channel B */
void wrio_b(porttype d);

/* write values to channel B */
void wrio_b_HINT(porttype d);

/* Setup communication with host */
void evm_setup_com(void);

#endif
```

7.2 EVMIO.C

```
/*******************************************************
 *                                                     *
 *  (C) Copyright 1996, Texas Instruments Incorporated   *
 *                                                     *
 *******************************************************/

/* C54x evm communication routines */

#include "evmio.h"

 /* (target control register)          */
ioport unsigned port14;       /* EVM STATUS REGISTER */
ioport unsigned port13;       /* EVM CHANNEL B INT */
ioport unsigned port12;       /* EVM CHANNEL B */
ioport unsigned port11;       /* EVM CHANNEL A INT */
ioport unsigned port10;       /* EVM CHANNEL A */



/* read values from channel A */
porttype rdio_a()
{
       /* wait for data */
       while ((port14 & 0x02) == 0);
              /* read data */
       return (porttype)port10;
}


/* write values to channel A */
void wrio_a(porttype d)
{
```

```
        /* make sure we don't overwrite */
        while ((port14 & 0x0001) != 0);
        port10=d; /* write data */
        return;
}

/* write values to channel A - generate host interrupt */
void wrio_a_HINT(porttype d)
{
        /* make sure we don't overwrite */
        while ((port14 & 0x0001) != 0);
        port11=d; /* write data generating Host Interrupt */
                  /* regardless of ATIE value */
        return;
}

/* read values from channel B */
porttype rdio_b()
{
        /* wait for data */
        while ((port14 & 0x030) == 0);
        /* read data */
        return (porttype)port12;
}

/* write values to channel B */
void wrio_b(porttype d)
{
        /* wait: buffer is full */
        while ((port14 & 0x0C) == 0x8);
        port12=d; /* write data */
        return;
}

/* write values to channel B - generate host interrupt if BTIE set*/
void wrio_b_HINT(porttype d)
{
        /* wait: buffer is full */
        while ((port14 & 0x0C) == 0x8);
        port13=d; /* write data generating Host Interrupt */
                  /* if BTIE set */
        return;
}

/* Setup communication with host */
void evm_setup_com(void)
{
        asm("init1  XC 2,BIO ");
        asm("       B  init1 ");
        asm("init2  XC 2,NBIO");
        asm("       B  init2 ");
        asm("init3  XC 2,BIO ");
        asm("       B  init3 ");
        wrio_a(DUMMY);
        return;
}
```

7.3 IOPORT.INC

```
*********************************************************
*                                                       *
*  (C) Copyright 1996, Texas Instruments Incorporated   *
*                                                       *
*********************************************************

* This file contains the constant definition to use with the EVM

CH_A    .set    00010h
CH_A_I  .set    00011h
CH_B    .set    00012h
CH_B_I  .set    00013h
STATUS  .set    00014h
        .asg    1,TRUE
        .asg    0,FALSE
INT     .set    TRUE
```

7.4 EVMSETUP.ASM

```
*********************************************************
*                                                       *
*  (C) Copyright 1996, Texas Instruments Incorporated   *
*                                                       *
*********************************************************

************************************
* Channel A/B communication routines *
************************************

******************************
* setup communication with PC *
******************************


evmsetup .macro

        .mlist

init1   xc      2,BIO
        b       init1
init2   xc      2,NBIO
        b       init2
init3   xc      2,BIO
        b       init3
ack:                                    ;send acknowledge
        portr   STATUS, *(AL)           ;read target control register
        and     #00001h, A              ;Get Channel A transmit status
        bc      ack, ANEQ               ;Branch if previous data not read by host
        portw   *(#0ffffh), CH_A        ;send to IO address CH_A

        .endm
```

7.5 RDIO_A.ASM

```
*********************************************************
*                                                       *
*  (C) Copyright 1996, Texas Instruments Incorporated   *
*                                                       *
*********************************************************

************************************
* Channel A communication routines *
************************************

*************************
* Channel A read routine *
*************************

rdio_a  .macro  addr
read_a?
        portr   STATUS, *(AL)           ;read target control register
        and     #00002h, A              ;Get Channel A receive status
        bc      read_a?, AEQ            ;Branch if no data available
        portr   CH_A,:addr:            ;Read from IO address CH_A
        .endm
```

7.6 RDIO_B.ASM

```
*********************************************************
*                                                       *
*  (C) Copyright 1996, Texas Instruments Incorporated   *
*                                                       *
*********************************************************

************************************
* Channel B communication routines *
************************************

*************************
* Channel B read routine *
*************************

rdio_b  .macro  addr
read_b?
        portr   STATUS, *(AL)           ;Read Target control register
```

```
        and     #30h, A                 ;Get Channel B receive status
        bc      read_b?, AEQ            ;branch if buffer empty
        portr   CH_B,:addr:            ;Read from IO address CH_B
        .endm
```

7.7 WRIO_A.ASM

```
*********************************************************
*                                                       *
*  (C) Copyright 1996, Texas Instruments Incorporated   *
*                                                       *
*********************************************************

***********************************
* Channel A communication routines *
***********************************

*************************
* Channel A write routine *
*************************

wrio_a  .macro  addr, int
write_a?
        portr   STATUS, *(AL)         ;read target control register
        and     #00001h, A            ;Get Channel A transmit status
        bc      write_a?, ANEQ        ;Branch if previous data not read by host
        .nolist
        .if     ($symlen(int) = 0)
        .eval   FALSE, int
        .endif
        .list
        .if     (int = TRUE)
        portw  :addr:, CH_A_I ;send to IO address CH_A_I
        .else
        portw  :addr:, CH_A           ;send to IO address CH_A
        .endif
        .endm
```

7.8 WRIO_B.ASM

```
*********************************************************
*                                                       *
*  (C) Copyright 1996, Texas Instruments Incorporated   *
*                                                       *
*********************************************************

***********************************
* Channel B communication routines *
***********************************

*************************
* Channel B write routine *
*************************

wrio_b  .macro  addr, int
write_b?
        portr   STATUS, *(AL)           ;read target control register
        and     #0000ch, A              ;Get Channel B transmit status
        sub     #8,A                    ;Buffer full?
        bc      write_b?, AEQ           ;Branch if buffer is full
        .nolist
        .if     ($symlen(int) = 0)
        .eval   FALSE, int
        .endif
        .list
        .if     (int = TRUE)
        portw  :addr:, CH_B_I          ;send to IO address CH_B_I
        .else
        portw  :addr:, CH_B            ;send to IO address CH_B
        .endif
        .endm
```

8. Test programs listings

8.1 UTILS.H

```
/*****************************************************
 *                                                   *
 *  (C) Copyright 1996, Texas Instruments Incorporated   *
 *                                                   *
 *****************************************************/

#ifndef UTILS
#define UTILS

#include <time.h>

typedef void (*fptr)();


/* get block size and number of blocks */
void getParam(int *nb, int *size);

/* calculate transfer speed */
double comSpeed(double nbWords, clock_t start, clock_t stop);

#endif
```

8.2 UTILS.C

```
/*****************************************************
 *                                                   *
 *  (C) Copyright 1996, Texas Instruments Incorporated   *
 *                                                   *
 *****************************************************/

#include "utils.h"
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

/* get block size and number of blocks */
void getParam(int *nb, int *size)
{
        char line[80];

        /* get the block size */
        printf("\nBlock size: ");
        gets(line);
        sscanf(line,"%d",size);
        /* get the number of blocks to transmit/receive */
        printf("\nNumber of blocks: ");
        gets(line);
        sscanf(line,"%d",nb);

        return;
}

/* division by zero handler */
void divideByZero(void)
{
        puts("\nDivision by zero");
        puts("Please increase the number of blocks and/or the block size");
        return;
}

/* evaluation communication speed */
double comSpeed(double nbWords, clock_t start, clock_t stop)
{
        double rate,time;

        /* specify exception handler */
        signal(SIGFPE,(fptr)divideByZero);
        time=(stop-start)/CLK_TCK;
        if (time==0.0)
        {
                /* exception */
                raise(SIGFPE);
                rate=-1.0;
```

```
        }
        else
                /* rate in Mbps */
                rate=(nbWords*16.0)/(1.0e6*time);

        return rate;
}
```

8.3 HOST.C

```
/*******************************************************
 *                                                     *
 *  (C) Copyright 1996, Texas Instruments Incorporated   *
 *                                                     *
 *******************************************************/

#include <stdio.h>
#include <values.h>
#include <stdlib.h>
#include <time.h>
#include <dos.h>

#include "hostio.h"
#include "utils.h"

void main(void)
{
        int *table;
        int i,j,k;
        unsigned int blockSize,nbBlocks;
        clock_t start,stop;
        double rate;

        puts("\n\n");
        puts("*********************************");
        puts("*     Communication Test        *");
        puts("*     c541 EVM <---> PC         *");
        puts("*********************************");

        /* get number of blocks and the block size */
        getParam(&nbBlocks,&blockSize);

        table=(int *)malloc(blockSize*sizeof(int));
        for (i=0; i<blockSize; i++)
                /* table[i]=random(MAXINT) */
                table[i]=i;

        /* Reset host/target communications */
        puts("\n\nReset host/target communication");
        reset_HI((porttype)0x4000);

        /* initialize host/target connection */
        printf("Initializing connection...");
        host_setup_com();
        puts("\nConnection established.");


        /* Example 1: */
        /* Transmission using Channel B - PC->EVM (Interrupt driven) */
        printf("\n\nPC reads Host Control Register: 0x%x",inport(BASE+STATUS));
        puts("\nExample 1: Transmission using Channel B - PC->EVM (Interrupt
        driven)");
        printf("Transmitting...");
        start=clock();
        for (i=0; i<nbBlocks; i++)
        {
                /* send packet size */
                /* BlockSize-1 transmitted because */
                /* loop includes value 0 */
                write_B(CH_B_I,(porttype)(blockSize-1));
                /* send packet */
                for (j=0; j<blockSize; j++)
                        write_B(CH_B,table[j]);
        }
        stop=clock();
        printf("Done");
        rate=comSpeed((double)nbBlocks*(double)blockSize,start,stop);
        printf("\nChannel B: Transmission rate is %g Mbps\n",rate);
        printf("PC reads Host Control Register: 0x%x",inport(BASE+STATUS));
```

```
        /* Example 2: */
        /* Transmission using Channel A - PC->EVM (Interrupt driven) */
        printf("\n\nPC reads Host Control Register: 0x%x",inport(BASE+STATUS));
        puts("\nExample 2: Transmission using Channel A - PC->EVM (Interrupt
        driven)");
        printf("Transmitting...");
        start=clock();
        for (i=0; i<nbBlocks; i++)
                for (j=0; j<blockSize; j++)
                        write_A(table[j]);
        stop=clock();
        printf("Done");
        printf("\nChannel A: Transmission rate is %g Mbps\n",rate);
        printf("PC reads Host Control Register: 0x%x",inport(BASE+STATUS));

        /* get accumulator A value from DSP */
        printf("\n\nAccumulator A = 0x%02x",read_B());
        printf("%04x",read_B());
        printf("%04x\n\n",read_B());

        free(table);
        return;
}
```

8.4 TARGET.ASM

```
*********************************************************
*                                                       *
*  (C) Copyright 1996, Texas Instruments Incorporated   *
*                                                       *
*********************************************************

*****************************************************
*       Example of communication with the C541 EVM *
*****************************************************



        .def    Reset                   ; Entry point symbol


****************************************
*       Stack setup                     *
****************************************
BOS     .usect  "stack",0fffh           ; setup stack
TOS     .usect  "stack",1               ; Top of stack at reset

****************************************
*       Interrupt vectors               *
****************************************
        .sect   "vectors"

Reset:  bd      Entry                   ; reset vector
        stm     #TOS,SP                 ; Setup stack

        .loop 16
        .space  4*16
        .endloop

INT1:   bd      rd_A                    ; INT1 vector
        pshm    AG
        pshm    AH

INT2:   bd      rd_B                    ; INT2 vector
        pshm    AG
        pshm    AH



****************************************
*       Main Program                    *
****************************************
        .sect   "program"
Entry
        .include        ioport.inc
        .mlib           "evmio.lib"
        .mmregs

        stm     #1111111111000000b,PMST ;setup PMST
*                 ||||||||||||||||
*                 ||||||||||||++++- dont care
*                 ||||||||||+----- AVIS
```

```
*                ||||||||||+------ OVLY    = 0 ext prg RAM
*                |||||||||+------- MP / MC = 1 Microprocessor mode
*                +++++++++-------- IPTR    = 0xff8 vectors at 0xff80
*
        stm     #10000b,TCR             ;stop the timer
        stm     #2000h,SWWSR            ;program 2 WS for IO space
                                        ;require by EVM

        stm     #100h,AR2               ;set receive address for channel A
        stm     #200h,AR3               ;set receive address for channel B
        stm     #300h,AR4               ;set transmit address for channel B

        ssbx    SXM                     ;sign extension mode is ON
        stm     #3,BK                   ;set circular buffer size to 3

        evmsetup                        ;setup comm with host PC


        stm     #110b,IMR               ;mask all interrupts except INT1 and INT2
        stm     #0ffffh,IFR             ;clear all pending interrupts
        rsbx    INTM                    ;enable globally interrupts

        ld      #-1,A                   ;load -1 (0xfffffffff) in Acc A


        idle    1                       ;waiting for interrupts to wake up

                                        ;copy contents of Acc A (40 bits)
                                        ;in this case -1 (0xfffffffff)
        mvkd    AG,*AR4+%               ;copy AG
        sth     A,*AR4+%                ;copy AH
        stl     A,*AR4+%                ;copy AL

        pshm    AG                      ;save context
        pshm    AH
        pshm    AL                      ;context saved

        wrio_b  *AR4+%                  ;transmit AG
        wrio_b  *AR4+%                  ;transmit AH
        wrio_b  *AR4+%                  ;transmit AL

        popm    AL                      ;restore context
        popm    AH
        popm    AG                      ;context restored
done:   B       done


************************
* INT1 interrut routine *
************************
rd_A:   pshm    AL                      ;context saved
        rdio_a  *AR2                    ;read from channel A
        popm    AL                      ;restore context
        popm    AH
        popm    AG                      ;context restored
        rete

************************
* INT2 interrut routine *
************************
rd_B:   pshm    AL                      ;context saved
        rdio_b  *(BRC)                  ;read number of words to be transmitted
        rptb    B_end-1                 ;repeat block
          rdio_b  *AR3                  ;read from channel B
B_end:  popm    AL                      ;restore context
        popm    AH
        popm    AG                      ;context restored
        rete
```

8.5 HOST1.C

```
/*******************************************************
 *                                                     *
 *  (C) Copyright 1996, Texas Instruments Incorporated   *
 *                                                     *
 *******************************************************/

#include <stdio.h>
#include <values.h>
#include <stdlib.h>
#include <time.h>
```

```
#include <dos.h>

#include "hostio.h"
#include "utils.h"

void main(void)
{
        int *table;
        int i,j,k;
        unsigned int blockSize,nbBlocks;
        clock_t start,stop;
        double rate;

        puts("\n\n");
        puts("*********************************");
        puts("*     Communication Test        *");
        puts("*     c541 EVM <---> PC         *");
        puts("*********************************");

        /* get number of blocks and the block size */
        getParam(&nbBlocks,&blockSize);

        table=(int *)malloc(blockSize*sizeof(int));
        for (i=0; i<blockSize; i++)
               /* table[i]=random(MAXINT) */
               table[i]=i;

        /* Reset host/target communications */
        puts("\n\nReset host/target communication");
        reset_HI((porttype)0x4000);

        /* initialize host/target connection */
        printf("Initializing connection...");
        host_setup_com();
        puts("\nConnection established.");

        /* Example 1: */
        /* Transmission using Channel A - PC->EVM (not interrupt driven) */
        printf("\n\nPC reads Host Control Register: 0x%x",inport(BASE+STATUS));
        puts("\nExample 1: Transmission using Channel A - PC->EVM (not interrupt
        driven)");
        /* transmit the number of blocks */
        write_A(nbBlocks-1);
        /* transmit the block size */
        write_A(blockSize-1);
        printf("Transmitting...");
        start=clock();
        for (i=0; i<nbBlocks; i++)
               /* transmit 1 block */
               for (j=0; j<blockSize; j++)
                      write_A(table[j]);
        stop=clock();
        printf("Done");
        rate=comSpeed((double)nbBlocks*(double)blockSize,start,stop);
        printf("\nChannel A: Transmission rate is %g Mbps\n",rate);
        printf("PC reads Host Control Register: 0x%x",inport(BASE+STATUS));

        /* Example 2: */
        /* Transmission using Channel A - EVM->PC (not interrupt driven) */
        printf("\n\nPC reads Host Control Register: 0x%x",inport(BASE+STATUS));
        puts("\nExample 2: Transmission using Channel A - EVM->PC (not interrupt
        driven)");
        printf("Transmitting...");
        start=clock();
        for (i=0; i<nbBlocks; i++)
               /* receive 1 block */
               for (j=0; j<blockSize; j++)
                      read_A();
        stop=clock();
        printf("Done");
        rate=comSpeed((double)nbBlocks*(double)blockSize,start,stop);
        printf("\nChannel A: Transmission rate is %g Mbps\n",rate);
        printf("PC reads Host Control Register: 0x%x",inport(BASE+STATUS));

        printf("\n\nAccumulator A = 0x%02x",read_B());
        printf("%04x",read_B());
        printf("%04x\n\n",read_B());

        free(table);
        return;

}
```

8.6 TARGET1.ASM

```
*******************************************************
*                                                     *
*  (C) Copyright 1996, Texas Instruments Incorporated   *
*                                                     *
*******************************************************

************************************************
*       Example of communication with the C541 EVM *
************************************************



        .def    Reset                   ; Entry point symbol

        .bss    nbBlocks,1
        .bss    blockSize,1

****************************************
*       Stack setup                     *
****************************************
BOS     .usect  "stack",0fffh           ; setup stack
TOS     .usect  "stack",1               ; Top of stack at reset

****************************************
*       Interrupt vectors               *
****************************************
        .sect   "vectors"

Reset:  bd      Entry                   ; reset vector
        stm     #TOS,SP                 ; Setup stack


****************************************
*       Main Program                    *
****************************************
        .sect   "program"
Entry
        .include        ioport.inc
        .mlib           "evmio.lib"
        .mmregs

        .asg nop,PIPELINE

        stm     #1111111111000000b,PMST ;setup PMST
*                 ||||||||||||||||
*                 ||||||||||||+++- dont care
*                 |||||||||||+----- AVIS
*                 ||||||||||+------ OVLY    = 0 ext prg RAM
*                 |||||||||+------- MP / MC = 1 Microprocessor mode
*                 +++++++++-------- IPTR    = 0xff8 vectors at 0xff80
*
        stm     #10000b,TCR             ;stop the timer
        stm     #2000h,SWWSR            ;program 2 WS for IO space
                                        ;require by EVM

        stm     #100h,AR2               ;set receive address for channel A
        stm     #300h,AR4               ;set transmit address for channel B

        stm     #3,BK                   ;circular buffer size is 3
        ssbx    SXM                     ;sign extension mode is ON

        evmsetup                        ;setup comm with host PC


        ld      #-1,A                   ;load -1 (0xfffffffff) in Acc A

                                        ;copy contents of Acc A (40 bits)
                                        ;in this case -1 (0xfffffffff)
        mvkd    AG,*AR4+%               ;copy AG
        sth     A,*AR4+%                ;copy AH
        stl     A,*AR4+%                ;copy AL

        pshm    AG                      ;save context
        pshm    AH
        pshm    AL                      ;context saved

        rdio_a  *(#nbBlocks)            ;get number of blocks
        mvdm    #nbBlocks,AR3           ;update *AR3
        rdio_a  *(#blockSize)           ;get block size
```

```
getBlock
       mvdm    #blockSize,BRC          ;reload BRC
       PIPELINE                        ;due to pipeline latencies for BRC
       rptb    rx_end-1                ;receive each block
         rdio_a        *AR2            ;receive 1 data
rx_end: banz    getBlock,*AR3-          ;loop nbBlocks times


       mvdm    #nbBlocks,AR3           ;update *AR3
sendBlock
       mvdm    #blockSize,BRC          ;reload BRC
       PIPELINE                        ;due to pipeline latencies for BRC
       rptb    tx_end-1                ;transmit 1 block
         wrio_a        *AR4            ;transmit 1 data
tx_end: banz    sendBlock,*AR3-         ;loop nbBlocks times



       wrio_b  *AR4+%                  ;transmit AG
       wrio_b  *AR4+%                  ;transmit AH
       wrio_b  *AR4+%                  ;transmit AL

       popm    AL                      ;restore context
       popm    AH
       popm    AG                      ;context restored

done:   B       done
```

8.7 HOST2.C

```
/*******************************************************
 *                                                     *
 *  (C) Copyright 1996, Texas Instruments Incorporated   *
 *                                                     *
 *******************************************************/


#include <stdio.h>
#include <values.h>
#include <stdlib.h>
#include <time.h>
#include <dos.h>

#include "hostio.h"
#include "utils.h"

void main(void)
{
       int *table;
       int i,j,k;
       unsigned int blockSize,nbBlocks;
       clock_t start,stop;
       double rate;

       puts("\n\n");
       puts("*********************************");
       puts("*     Communication Test        *");
       puts("*     c541 EVM <---> PC         *");
       puts("*********************************");

       /* get size and number of blocks */
       getParam(&nbBlocks,&blockSize);

       table=(int *)malloc(blockSize*sizeof(int));
       for (i=0; i<blockSize; i++)
              /* table[i]=random(MAXINT) */
              table[i]=i;


       /* Reset host/target communications */
       puts("\n\nReset host/target communication");
       reset_HI((porttype)0x4000);

       /* initialize host/target connection */
       printf("Initializing connection...");
       host_setup_com();
       puts("\nConnection established.");
```

```
        printf("\n\nPC reads Host Control Register: 0x%x",inport(BASE+STATUS));
        puts("\nExample 1: Transmission using Channel A - PC->EVM (not interrupt
        driven)");
        /* transmit the number of blocks */
        write_A(nbBlocks);
        /* transmit the block size */
        write_A(blockSize);
        printf("Transmitting...");
        start=clock();
        for (i=0; i<nbBlocks; i++)
                /* transmit 1 block */
                for (j=0; j<blockSize; j++)
                        write_A(table[j]);
        stop=clock();
        printf("Done");
        rate=comSpeed((double)nbBlocks*(double)blockSize,start,stop);
        printf("\nChannel A: Transmission rate is %g Mbps\n",rate);
        printf("PC reads Host Control Register: 0x%x",inport(BASE+STATUS));


        printf("\n\nPC reads Host Control Register: 0x%x",inport(BASE+STATUS));
        puts("\nExample 2: Transmission using Channel B - EVM->PC (not interrupt
        driven)");
        printf("Transmitting...");
        start=clock();
        for (i=0; i<nbBlocks; i++)
                /* receive 1 block */
                for (j=0; j<blockSize; j++)
                        read_B();
        stop=clock();
        printf("Done");
        rate=comSpeed((double)nbBlocks*(double)blockSize,start,stop);
        printf("\nChannel B: Transmission rate is %g Mbps\n",rate);
        printf("PC reads Host Control Register: 0x%x\n\n",inport(BASE+STATUS));

        free(table);
        return;

}
```

8.8 TARGET2.C

```
/*******************************************************
 *                                                     *
 *  (C) Copyright 1996, Texas Instruments Incorporated   *
 *                                                     *
 *******************************************************/

/*************************************************
*       Example of communication with the C541 EVM  *
*************************************************/

#include "evmio.h"


void main(void)
{
        int nbBlocks,blockSize;
        int dummy=0xffff;
        int i,j;


        evm_setup_com();

        nbBlocks=rdio_a(); /* get number of blocks */
        blockSize=rdio_a(); /* get block size */

        /* read from channel A nbBlocks of blockSize */
        for (i=0; i<nbBlocks; i++)
                for (j=0; j<blockSize; j++)
                        rdio_a();

        /* write to channel B nbBlocks of blockSize */
        for (i=0; i<nbBlocks; i++)
                for (j=0; j<blockSize; j++)
                        wrio_b(dummy);

        return;

}
```

References

1. TMS320C54x Evaluation Module - Technical Reference (SPRU135)
2. TMS320C54x CPU & Peripherals Ref set Vol 1 (SPRU131C)
3. TMS320C54x Assembly Language Tools (SPRU102A)