Freescale Semiconductor
Application Note

Document Number: AN4147
Rev. 0, 07/2010

Tips and Tricks with DMA on MPC56xx

by: David Cermak
Applications Engineering

Contents

1 Introduction

This application note presents a set of simple examples using eDMA on MPC56xx and MPC55xx to emulate receiving and transmitting serial data over SPI and SCI. This additionally demonstrates advanced features of eDMA on these examples together with some other techniques to minimize CPU load and perform certain nontrivial tasks, such as SCI/SPI communication using eDMA and other peripherals. This does not include any software task.

In this process seven sample applications were prepared for MPC5607B, employing various DMA channels to emulate

- SCI transmitter
- SCI receiver
- SPI master
- SPI slave

All these examples emulate simple serial communications and thus share one technique to serialize/de-serialize data bits using unused or unusable GPIO pins. More specifically SIUL (System Integration Unit Lite) registers for serial and parallel access.

Other techniques or eDMA features, which are used in these sample applications, include:

- Channel to channel linking
- Scatter-gather feature
- Conditional linking using eMIOS

All these examples are compilable using standard development tools, listed below with the versions used for the testing. A simple *Makefile* is used to build executable for the following three compilers.

- GreenHills version 5.16
- WindRiver version 5.6.1
- CodeWarrior version 2.3

2 eDMA configuration

This section outlines the eDMA configuration options. For details, please refer to the Reference Manual for each device. The programming model defines:

- Global registers
- Priority registers
- Transfer control descriptors

2.1 Transfer Control Descriptor

Transfer Control Descriptor (TCD) is a data field, which defines behaviour of one DMA channel. The TCD contains the following information, see Table 1

- Source address, offset
- Destination address, offset
- Transfer attributes
- Control/Status

Table 1. Transfer Control Descriptor

DMA2 Offset	TCDn Field	
0x1000 + (32 × n) + 0×00	Source Address (saddr)	
0x1000 + (32 × n) + 0×04	Transfer Attributes (smod, ssize, dmod, dsize)	Signed Source Address Offset (soff)
0x1000 + (32 × n) + 0x08	Inner Minor Byte Count (nbytes)	
0x1000 + (32 × n) + 0x0c	Last Source Address Adjustment (slast)	
0x1000 + (32 × n) + 0x10	Destination Address (daddr)	
0x1000 + (32 × n) + 0x14	Current 'Major' Iteration Count (citer)	Signed Destination Address Offset (soff)
0x1000 + (32 × n) + 0x18	Last Destination Address Adjustment/Scatter Gather Address (dlast_sga)	
0x1000 + (32 × n) + 0x1c	Beginning Major Iteration Count (biter)	Channel Control Status (bwc, major.linkch, done, active, major.e_link, e_sg, d_req, int_half, int_maj_start)

3 Features used

This section describes the features that are used in the examples of emulating SPI and SCI.

3.1 GPIO Registers for serial and parallel access

For transmitting and receiving data in a serial manner, it is necessary to provide a bit access or bit shift to the transmitted and received data. Hardware communication peripherals usually utilize shift registers. Software emulation uses a bit shift commonly.

This application note outlines a method of using eDMA and SIUL (System Integration Unit Lite) registers to serialize (deserialize) data bits and sending (receiving) them as one communication word. The simplest example is sending 8-bit data frame over SCI. The peripherals in this case perform the following tasks:

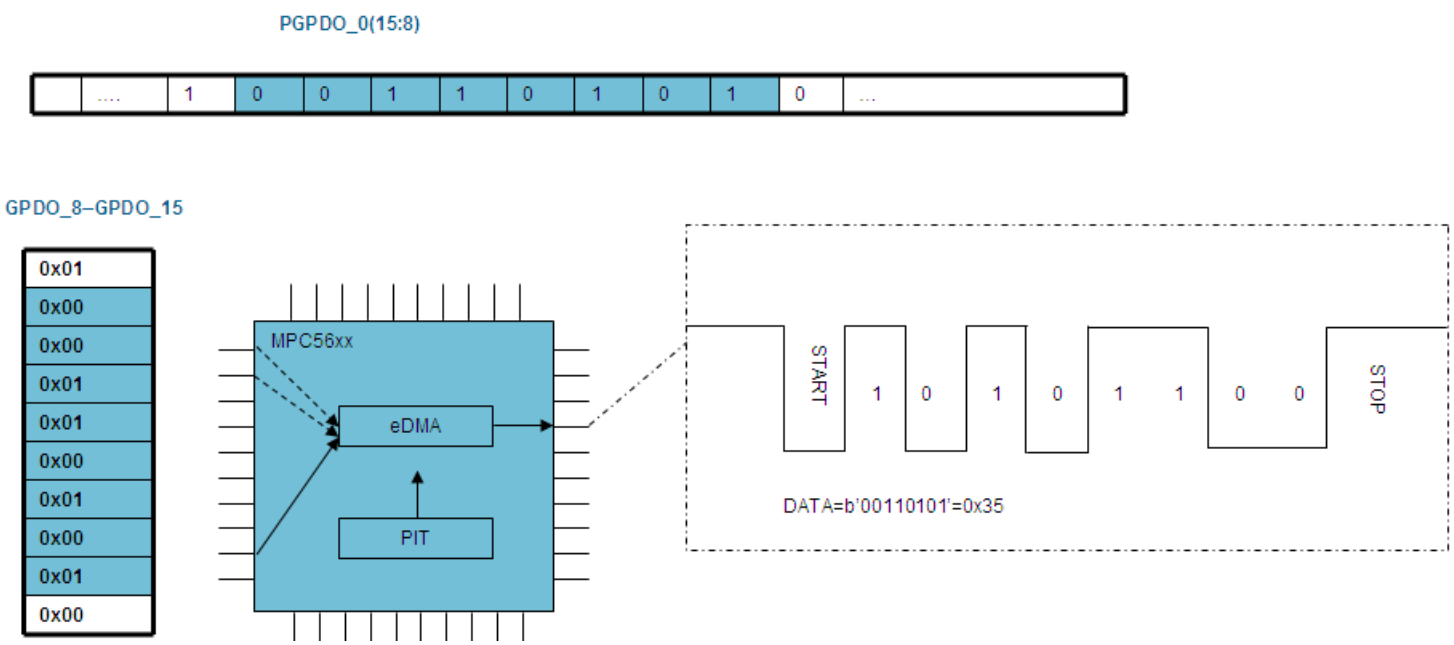
- **eDMA**: Moving 8 bytes in a loop (each byte contains one bit to be sent) into GPIO register for a given pin (UART Tx)
- **SIUL**: It is possible to access the whole port (gate) writing a word (byte) to parallel GPIO registers. It is also possible to access each pin separately writing a byte to serial GPIO registers. These registers mirror the same data, so by writing various data into serial register, the same data would automatically appear in parallel registers and vice-versa. This is beneficial, as the eDMA may use the serial access registers and transfer a byte array in a loop, while the application may use the parallel access registers to write the whole word at once.

In order to utilize SIUL registers for serializing and deserializing data, the user application must reserve some GPIO registers like the following:

- Unused GPIO: Pins, which are normally present on a device, but are not used by customer application in GPIO mode (might be used in different mode).
- Unusable GPIO: Pins which are not present on a device package, but registers for these pins are accessible.

See the principle of this operation in the Figure 1, which refers to the transmission of one byte over SCI.

Figure 1. Principle of Serial and Parallel Access in SCI transmit



3.2 Channel to channel linking

It is possible to link to the other DMA channel after completion of

- Minor Loop
- Major Loop

When both the linkages are enabled and both loops have completed, the major loop link is performed only.

Another option to start a channel from one channel is to write into SSBR register the number of channel to link.

These features are used in SPI communications to split the job into receiving and transmitting, in case if both the edges of the clock signal are sensed (SPI slave) or generated (SPI master).

3.3 Scatter Gather feature

One DMA channel is allowed to use multiple Transfer Control Descriptors and this enables a DMA channel to scatter the DMA data to multiple destinations or gather it from multiple sources.

When scatter gather feature is enabled, on completion of a major loop for certain channel, a pointer to the next TCD is de-referenced and loaded into this channel. The user TCD is usually located in flash memory.

Complex transfers within limited number of channels may use scatter gather operation to service multiple heterogeneous data movements.

3.4 Conditional linkage

In some cases, it is necessary to process the DMA on the data of variable length, i.e. to conditionally link to a DMA channel depending on the processed data.

MPC56xx platform defines very modular peripheral eMIOS, containing time-base, counters and comparators. It is possible in certain modes to use some of these blocks separately. For example, in GPIO mode, the user has a write access to the counter and a comparator may be used. Moreover, the unified channels have usually DMA triggering capability, so the configured DMA channel can be started, on matching of an eMIOS comparator. In case of writing to the compare register by another DMA channel, the system works as a dynamic conditional channel to channel link.

This feature is used, when transmitting variable amount of data, where a special character is used to stop the transmission.

4 SCI Transmitter

The examples in this section emulate the SCI transmitter using eDMA. All these examples are implemented for the device MPC5607B, but are generally applicable to the 5600 family.

4.1 Module sci_app_01.c

eDMA emulating single SCI transmit

This example demonstrates how the eDMA will be used to emulate SCI. It uses LINFlex0 to receive one byte from serial line, and sends the same byte using single DMA channel. The following configuration is used:

- LINFlex0 - Rx only on PCR_19 (PB3)
- DMA channel 0 writes to GPIO PCR_18 (PB2)
- DMA triggered using PIT_0 with always enabled source (periodic triggers only)
- Software task: After a byte is received through UART, the same is transmitted using eDMA
- Connection: Standard wiring on EVB: PB2 and PB3 -> SCI transciever

4.1.1 Function SciInit01

Initializes the first SCI application to transmit one byte at a time using one eDMA channel triggered periodically by PIT.

Prototype: `void SciInit01(void);`

Initialization of the following peripherals:

- SIU: PCR registers for Rx and Tx signals
- LINFLEX_0: As UART reciever on 9.600 kBauds, 8-bit data bits

- PIT_0: Periodically triggers DMA at 9600 Hz (one bit per timer overflow)
- DMAMUX: Routes PIT_0 flag to activate channel 0 DMA
- eDMA0: Setup one byte data movements
  - from GPIO's PG15-PH9 (10 iterations of minor loop)
  - to GPIO PB2 (SCI Tx)
  - disable on completion (after one byte transmitted, turns the eDMA off)

4.1.2 Function SciRun01

Checks the LINFlex receiver and if anything is received, sends it back using eDMA.

Prototype: `void SciRun01(void);`

Data to be transmitted are located at PG15-PH9, where

- PG15 is a start bit
- PH0-PH8 are data bits
  - accessed using GPDO[112-119] using eDMA (serial)
  - accessed using PGPDO[3] using application (parallel)
- PH9 is a stop bit

Data transmission is started by enabling the DMA channel. At this time the flag is cleared and timer is restarted, as one byte at a time is sent. In case of multi-byte transmission this will not be necessary.

4.1.3 Function SciApp01

Function interface to execute SCI communication emulation Module spi_app_01.c.

Prototype: `void SciApp01(void);`

4.2 Module sci_app_02.c

eDMA emulating multiple SCI transmits

This example demonstrates how eDMA could be used to emulate multiple SCI transmits. Sending a string of bytes using chaining two eDMA channels Configuration:

- DMA channel 0 writes to GPIO PCR_18 (PB2)
- DMA channel 0 activated using periodic triggers
- DMA channel 1 takes one byte from buffers and links to the channel 0
- DMA channel 1 activated explicitly by SW to transmit burst of bytes
- Connection: Standard wiring on EVB: PB2 and PB3 -> SCI transceiver

4.2.1 Variable pTxData

This is a reference to GPIO parallel access registers. It acts as a pointer to the data prepared for transmission, i.e. serialization.

4.2.2 Variable stringBuffer

This stores the string that is periodically transmitted over a serial line.

NOTE

The transmission has fixed number of data, which equals to the size of this byte array.

4.2.3 Function SciInit02

Initializes the SCI application to transmit one byte continuously, while this byte is being modified by other channel.

Prototype: `void SciInit02(void);`

Initialization of the following peripherals:

- SIU: PCR registers for UART Tx signals
- PIT_0: Periodically triggers DMA at 9600 Hz (one bit per timer overflow)
- DMAMUX: Routes PIT_0 flag to activate channel 0 DMA
- eDMA0: Setup one byte data movements
  - from GPIO's PG15-PH9 (10 iterations of minor loop)
  - to GPIO PB2 (SCI Tx)
  - link to the channel 1 after major loop.
- eDMA1: Setup one byte transfer from data buffer to PH0-PH7 (data to be sent)

4.2.4 Function SciRun02

Transmits the message "Hello World!" continuously in a serial line.

Prototype: `void SciRun02(void);`

Data to be transmitted are located at PG15-PH9, where

- PG15 is a start bit
- PH0-PH8 are data bits
  - accessed using GPDO[112-119] using eDMA (serial)
  - accessed using PGPDO[3] using application (parallel)
- PH9 is a stop bit

Data transmission starts by enabling the DMA channel 0, which transmits one byte. After the transmission, links to the channel 1, which transmits another byte from transmitting buffer. Channel 0 is not disabled, so transmission continues with all the messages.

These transmissions are performed only by eDMA, so that there is no software task after activation of the two channels and the timer. This function never exits and waits in the empty neverending loop.

4.2.5 Function SciApp02

Prototype: `void SciApp02(void);`

4.3 Module sci_app_04.c

eDMA emulating multiple SCI transmits under the given condition

This example demonstrates how the eDMA is used to transmit SCI data of variable length configuration:

- DMA channel 0 writes to GPIO PCR_18 (PB2) and is activated by PIT
- DMA channel 1 takes one byte from buffer and link from the channel 0
- DMA channel 2 sends one byte from buffer also to the comparator (is linked from the channel 1)
- DMA channel 3 is activated either by software (to start the timer) or by eMIOS-comparator (on match) to stop transmitting the data.
- Connection: Standard wiring on EVB: PB2 and PB3 -> SCI transceiver

4.3.1 Variable pTxData

This is a reference to GPIO parallel access registers. This serves as a pointer to the data prepared for transmission, i.e. serialization.

4.3.2 Variable stringBuffer

This stores the string which is transmitted over serial line. The end of this buffer is indicated by the last character '.'. The data is compared during the transmission with the last character and the transmission is stopped on finding a match.

4.3.3 Variable toggleTimer

This array is used to switch within (start/stop) periodic interrupt timer, to enable or disable the transmission. Enabling is done by the application, whereas disabling is automatic, when a stop character is indicated.

4.3.4 Function SciInit04

Initializes the SCI application to transmit one byte continuously, while this byte is modified by the channel 1. Channel 2 transmits this byte to the comparator, which triggers channel 3 when it finds the matching value. Channel 3 then disables the PIT, i.e. the data transmission.

Prototype: `void SciInit04(void);`

Initialization of the following peripherals:

- SIU: PCR registers for UART Tx signals
- PIT_0: Periodically triggers DMA at 9600 Hz (one bit per timer overflow)
- DMAMUX
  - Routes PIT_0 flag to activate channel 0 DMA
  - Routes eMIOS flag (from comparator) to trigger channel 3
- eDMA0: Setup one byte data movements (to transmit one SCI byte)
- eDMA1: Setup one byte transfer from data buffer to PH0-PH7 (data to be sent)
- eDMA2: Setup one byte transfer from data buffer to eMIOS comparator
- eDMA3: Setup to enable/disable the PIT. This channel is triggered from eMIOS comparator.

4.3.5 Function SciRun04

Transmits the message "Hello World etc" from the designated buffer in a serial line. This message can have variable length, the end of the message is indicated by the last character '.'.

Prototype: `void SciRun04(void);`

Data transmission is started by enabling the DMA channel 0, which transmits one byte. After the transmission, links to the channel 1, which transmits another byte from transmitting buffer. Triggering for channel 0 is disabled by channel 3, when a stop character is found '.'

4.3.6 Function SciApp04

Prototype: `void SciApp04(void);`

4.4 Module sci_app_05.c

eDMA emulating multiple SCI transmits and using scatter gather option

This example demonstrates how the eDMA will be used to emulate SCI transmitter of buffer with fixed length and scatter gather option, using two DMA channels. Given below is the configuration used:

- DMA channel 0 writes to GPIO PCR_18 (PB2)
- DMA channel 0 activated using periodic triggers
- DMA channel 1 takes one byte from buffer and links to the channel 0
- DMA channel 1 activated explicitly by software to transmit burst of bytes
- DMA channel 1 scatters to USER TCD, which disables the timer and hence the transmission
- Connection: Standard wiring on EVB: PB2 and PB3 -> SCI transciever

This example is the same as sci_app_02, uses the same number of channels, but also scatter gather feature to solve buffering and disabling in one channel.

4.4.1 Variable pTxData

This is a reference to GPIO parallel access registers. It is a pointer to the data prepared for transmission, i.e. serialization.

4.4.2 Variable stringBuffer

String which is periodically transmitted over serial line. Note that the transmission has fixed number of data (12 in this case).

4.4.3 Variable stopTimer

This is used to store the number to be moved to PIT control register to stop PIT and hence the SCI transmission.

4.4.4 Function SciInit05

Initializes the SCI application to transmit one byte continuously, while this byte is being modified by other channel.

Prototype: `void SciInit05(void);`

Initialization of the following peripherals:

- SIU: PCR registers for UART Tx signals
- PIT_0: To periodically trigger DMA at 9600 Hz (one bit per timer overflow)
- DMAMUX: Routes PIT_0 flag to activate channel 0 DMA
- eDMA0: Setup one byte data movements
  - from GPIO's PG15-PH9 (10 iterations of minor loop)
  - to GPIO PB2 (SCI Tx)
  - link to the channel 1 after major loop.

- eDMA1: Setup one byte transfer from data buffer to PH0-PH7 (data to be sent)
  - this channel uses scatter gather operation to split the task into buffering the data and disabling the channel.

NOTE

User TCD's are for demonstration purpose only located in the TCD array (elements 11 and 12). In real application this location will be anywhere in the user memory (preferably flash), but aligned to the 32 bytes boundary.

4.4.5 Function SciRun05

Transmits the message "Hello World!" in a serial line. After sending these 12 characters, the transmission is stopped.

Prototype: `void SciRun05(void);`

Data transmission is started by enabling the DMA channel 0, which transmits one byte. After the transmission, by linking to the channel 1, passes the other byte from transmitting buffer. After a fixed number of iterations of channel 1 (major loop), scatter gather operation is performed and the user TCD is loaded to channel 1, which disables the timer, i.e. the transmission. Then, in the same way another TCD is loaded to the channel 1 (the original one), to setup the system for another data transmission.

4.4.6 Function SciApp05

Prototype: `void SciApp05(void);`

5 SCI Receiver

The following example demonstrates receiving SPI signal using the SCI receiver.

5.1 Module sci_app_03.c

eDMA emulating simple SCI receiver

This example demonstrates how the eDMA could be used to receive SCI. Configuration used is as given below:

- DMA channel 0 reads from GPIO PCR_19 (PB3)
- DMA channel 0 activated using periodic triggers
- DMA channel 1 activates PIT to receive data
- DMA channel 1 activated using eMIOS0_CH0 -- input capture
- Connection: Standard wiring on EVB: PB2 and PB3=PA0 -> SCI transceiver
    - needs to connect PB3 and PA0.

5.1.1 Variable pRxData

This is a reference to GPIO parallel access registers. It is a pointer to the data that was received using byte by byte DMA movements (de-serialization).

5.1.2 Function SciInit03

Initializes the SCI application to receive one byte

Prototype: `void SciInit03(void);`

This involves initialization of the following peripherals:

- SIU: PCR registers for DMA UART Rx signals, eMIOS0, and LINFEX UART Tx
- PIT_0: Periodically triggers DMA at 9600 Hz (one bit per timer overflow)
- EMIOS_0: Channel0 -> input capture (as 5607B does not route EINT SIU to DMA)
- DMAMUX: Routes PIT_0 flag to activate channel 0 DMA, and EMIOS_0,ch0 to activate channel 1 DMA
- eDMA0: Setup one byte data movement from DMA UART Rx to PH0-PH7 (data bits)
- eDMA1: Starts the timer (and data acquisition) after trigger (start bit)

5.1.3 Function SciRun03

Receives one byte from serial line.

Prototype: `void SciRun03(void);`

Data transmission is started by eMIOS channel input capture event - falling edge of the START bit. This triggers DMA channel0, which enables PIT0 and that is used for data reception on GPIO pin PB3 (DMA UART Rx). Data received are located at PH0-PH8.

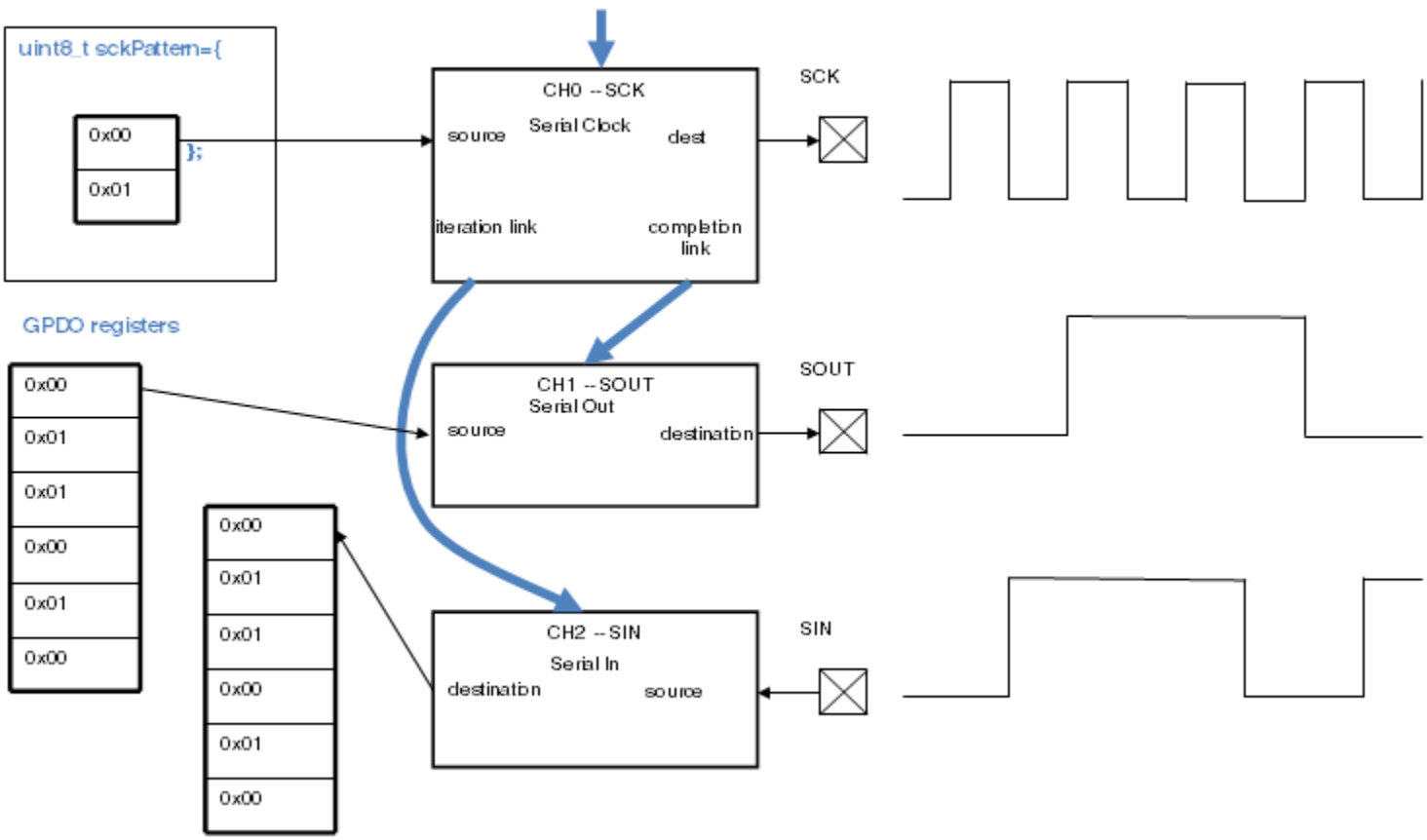
5.1.4 Function SciApp03

Prototype: `void SciApp03(void);`

6 SPI Master

SPI Master example uses the same features, but involves more signals for sending and receiving.

Figure 2. Principle of SPI Master emulation



6.1 Module spi_app_01.c

eDMA emulating SPI master

This example demonstrates how the eDMA will be used to emulate SPI master, sending and receiving data Configuration used is as given below:

- DMA channel 0 SCK generation activated periodically by PIT
- DMA channel 1 processes data in
- DMA channel 2 processes data out

- CS is solved by software, but could be as an additional DMA channel
- Connection: Connect DSPI to GPIO used for eDMA SPI emulation
  - PC4: DSPI1_SIN -> PB0 (DMA SOUT)
  - PC5: DSPI1_SOUT -> PB1 (DMA SIN)
  - PC2: DSPI1_SCK -> PB2 (DMA SCK)
  - PC3: DSPI1_CS0 -> PB3 (DMA CS0)

6.1.1 Variable pInData

Pointer to the input data accessing parallel GPIO registers.

6.1.2 Variable pOutData

Pointer to the output data accessing parallel GPIO registers.

6.1.3 Variable toggleTimer

This array is used to switch between (start/stop) PIT, to enable or disable the transmission. Enabling is done by the application, whereas disabling is automatic, when a stop character is indicated.

6.1.4 Variable generateClock

This array is used to generate clock, where alternately output is 1 and 0.

NOTE

The order depends on the selected SPI properties CPHA and CPOL, which in this example is equal to 1 and 0 respectively.

6.1.5 Function SpiInit01

Initializes the SPI master using eDMA and also the DSPI peripheral as a slave to check the data

Prototype: `void SpiInit01(void);`

Initialization of the following peripherals:

- SIU: PCR registers for DMA SPI master and DSPI SPI slave
- DSPI: SPI slave, 8 bit data.
- PIT_0: Periodically triggers DMA at 9600 Hz (SPI clock rate)
- DMAMUX: Routes PIT_0 flag to activate channel 0 DMA
- eDMA0: Setup to send alternately 0 and 1 to generate clock and link to -channel_1 to receive data (after iteration) -channel_2 to transmit data (after completion)
- eDMA1: Setup one byte transfer from GPIO to PH0-PH7 (reading)
- eDMA2: Setup one byte transfer from PH8-PH15 to GPIO (writing)
- eDMA3: Setup to enable or disable the PIT timer to control the SPI transmission.

6.1.6 Function SpiRun01

Transmits the message "Hello World!" and other messages from emulated SPI master to real SPI slave on DSPI_1.

Prototype: `void SpiRun01(void);`

Data transmission is started by channel 3, which enables PIT0. Timer then triggers channel0, which generates clock and is linked to

- channel 1 after iteration of minor loop - SPI read
- channel 2 after completion of major loop - SPI write
- channel 3 is linked from channel 1 after transmission completes and disables back PIT 0, which turns off the master

NOTE

Driving of CS (chip select) signals is done by software.

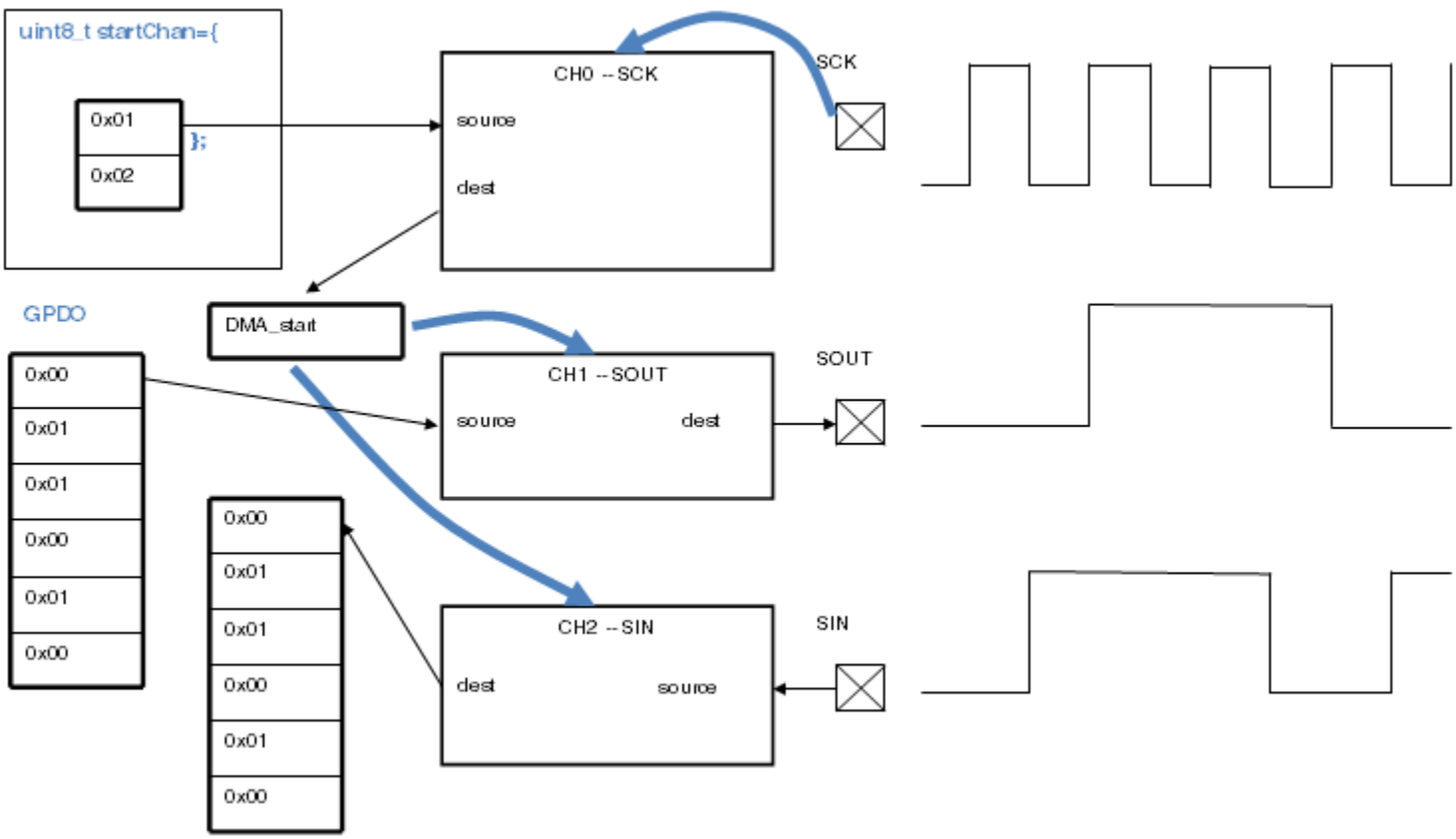
6.1.7 Function SpiApp01

Prototype: `void SpiApp01(void);`

7 SPI Slave

SPI slave example uses the same resources, but the order of triggering and information flow is opposite.

Figure 3. Principle of SPI Slave emulation



7.1 Module spi_app_02.c

eDMA emulating SPI slave

This example demonstrates how the eDMA could be used to emulate SPI slave, sending and receiving data Configuration used is as given below:

- DMA channel 0 SCK activated by SPI master (eMIOS0-CH0-SAIC)
- DMA channel 1 processes data in

- DMA channel 2 processes data out
- CS (chip select) is driven by software, but can be used as an additional DMA channel
- Connection: Connect DSPI to GPIO used for eDMA SPI emulation
  - PC4: DSPI1_SIN -> PB0 (DMA SOUT)
  - PC5: DSPI1_SOUT -> PB1 (DMA SIN)
  - PC2: DSPI1_SCK -> PA0 (DMA SCK)
  - PC3: DSPI1_CS0 -> PB3 (DMA CS0)

7.1.1 Variable pInData

This stores pointer to the input data accessing parallel GPIO registers.

7.1.2 Variable pOutData

This stores pointer to the output data accessing parallel GPIO registers.

7.1.3 Variable startDmaChannel

This array is used explicitly to start DMA channels for data reading and data writing.

NOTE

The order depends on the selected SPI properties CPHA and CPOL, which in this example equals to 1 and 0 respectively (upon the first edge sending, i.e. starting channel 2).

7.1.4 Function SpiInit02

Initializes the SPI slave using eDMA and also the DSPI peripheral as SPI master to check the data.

Prototype: `void SpiInit02(void);`

Initialization of the following peripherals:

- SIU: PCR registers for DMA SPI master and DSPI SPI slave
- DSPI: SPI master, 8 bit data.
- eMIOS: channel 0 as input capture unit (need DMA trigger, 5607B doesn't mux SIU to DMA)
- DMAMUX: Routes eMIOS0_CH0 to activate channel 0 DMA
- eDMA0: Setup to alternately trigger: -channel_1 to receive data (after iteration) -channel_2 to transmit data (after completion)
- eDMA1: Setup one byte transfer from GPIO to PH0-PH7 (reading)
- eDMA2: Setup one byte transfer from PH8-PH15 to GPIO (writing)

7.1.5 Function SpiRun02

Transmits the message "Hello World!" and other message from SPI master (DSPI_1) to SPI slave (emulated) and back.

Prototype: `void SpiRun02(void);`

SPI slave is triggered using eMIOS channels (instead of external interrupt)-channel0. This channel launches other two channels (2 and 1), which perform data transmission and reception.

7.1.6 Function SpiApp02

Prototype: `void SpiApp02(void);`

8 Common System Files

The modules under the common system files are used to setup the MPC5607B system for these examples.

8.1 Module app_main.c

Main function of the application.

8.1.1 Function main

Main function which initializes the system and runs the function interface for the selected example.

Prototype: `int main(void);`

8.2 Module sys_init.c

Initializes the MPC5607B to run on the oscillator clock and enable basic peripherals.

8.2.1 Function DisableWatchdog

Disables the watchdog for MPC5607B device.

Prototype: `void DisableWatchdog(void);`

8.2.2 Function EnableAllPeripherals

Enables all peripherals in DRUN mode for MPC5607B.

Prototype: `void EnableAllPeripherals(void);`

8.2.3 Function VVOscInit

Setup the default 8MHz oscillator (with no PLL), used on MPC56xx evaluation board.

Prototype: `void VVOscInit(void);`

8.2.4 Function SysInit

Initializes the CPU clock on 8MHxz x-tall located on standard EVB(evaluation board).

Prototype: `void SysInit(void);`

Disables the Watchdog, enables all peripheral in DRUN mode, using external 8MH XOSC, enables the clockout to PA[0]

9 Conclusion

MPC56xx device family offers a very flexible eDMA solution, as given in this application note. The eDMA peripheral could be used to emulate completely simple communication nodes (hardware based) for real SPI or SCI data transmission or reception.

More generally one can benefit from the described techniques and features for any other user application to reduce CPU load and balance the application performance.

Demonstrated methods employ advanced eDMA modes, supporting nested loops, links and even conditional branches. This enables a means to dynamically program complex data transfers, which normally need to be performed with some CPU intervention.

10 References

- FTF Americas 2010, AUT-F0451. Session recording available at http://www.freescale.com/ftf.
- MPC5607B Microcontroller Reference Manual. Power Architecture® Controllers (5xx/5xxx), http://www.freescale.com

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4147
Rev. 0, 07/2010